

UNIX - Command-Line Survival Guide

Files, directories, commands, text editors

Simon Prochnik & Lincoln Stein

Lecture Notes

- [What is the Command Line?](#)
 - [Logging In](#)
 - [The Desktop](#)
 - [The Shell](#)
 - [Home Sweet Home](#)
 - [Getting Around](#)
 - [Running Commands](#)
 - [Command Redirection](#)
 - [Pipes](#)
-

What is the Command Line?

Underlying the pretty Mac OSX GUI is a powerful command-line operating system. The command line gives you access to the internals of the OS, and is also a convenient way to write custom software and scripts.

Many bioinformatics tools are written to run on the command line and have no graphical interface. In many cases, a command line tool is more versatile than a graphical tool, because you can easily combine command line tools into automated scripts that accomplish tasks without human intervention.

In this course, we will be writing Perl scripts that are completely command-line based.

Logging into Your Workstation

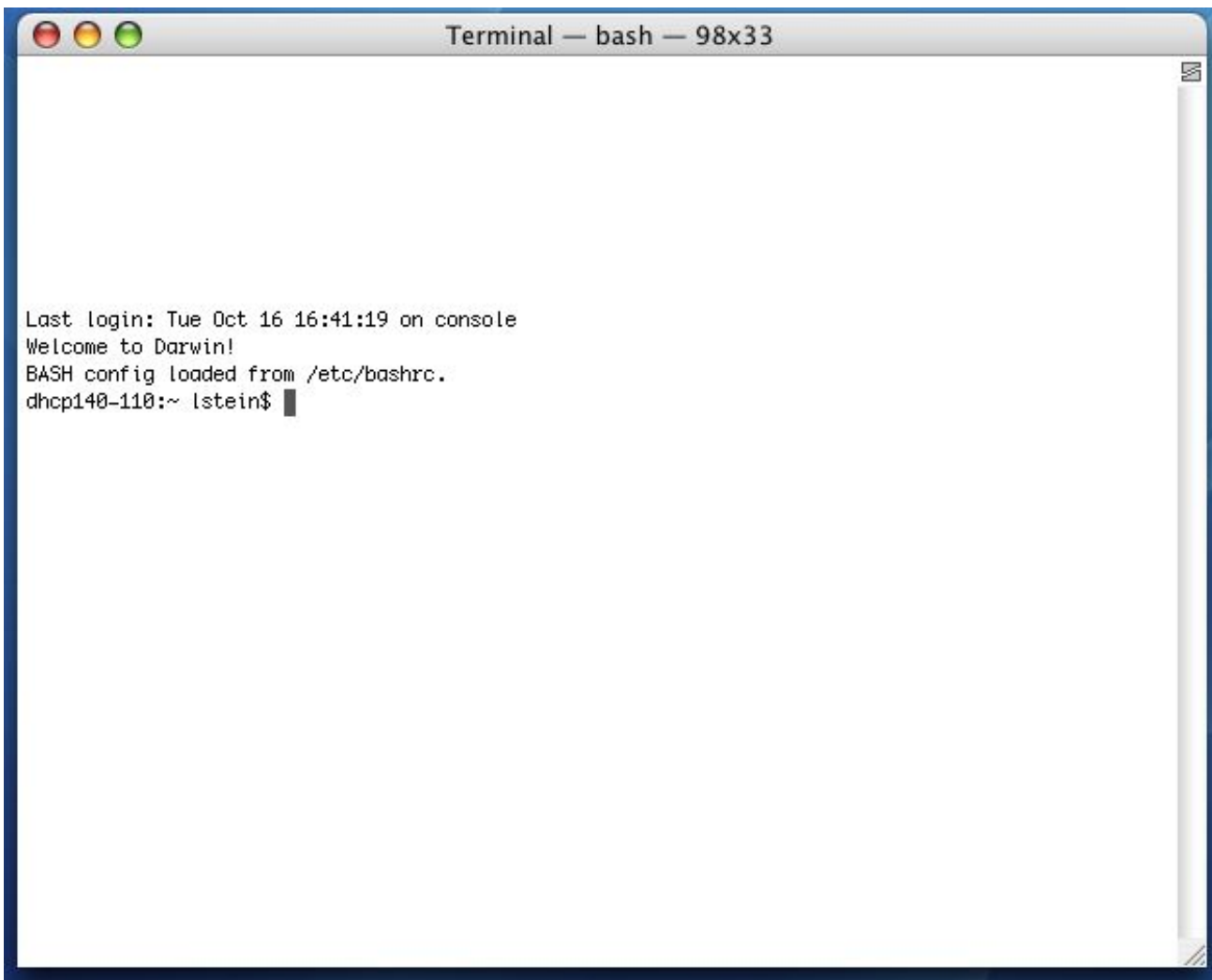
Your workstation is an iMac. To log into it, provide the following information:

Your username: the initial of your first name, followed by your full last name. For example, my username is **srobb** for **sofia robb**

Your password: **changeme**

Bringing up the Command Line

To bring up the command line, use the Finder to navigate to *Applications->Utilities* and double-click on the *Terminal* application. This will bring up a window like the following:



```
Terminal — bash — 98x33

Last login: Tue Oct 16 16:41:19 on console
Welcome to Darwin!
BASH config loaded from /etc/bashrc.
dhcp140-110:~ lstein$
```

OSX Terminal

You will be using this application a lot, so I suggest that you drag the Terminal icon into the shortcuts bar at the bottom of your screen.

OK. I've Logged in. What Now?

The terminal window is running a **shell** called "bash." The shell is a loop that:

1. Prints a prompt
2. Reads a line of input from the keyboard
3. Parses the line into one or more commands
4. Executes the commands (which usually print some output to the terminal)
5. Go back 1.

There are many different shells with bizarre names like **bash**, **sh**, **csh**, **tcsh**, **ksh**, and **zsh**. The "sh" part means shell. Each shell was designed for the purpose of confusing you and tripping you up. We have set up your accounts to use **bash**. Stay with **bash** and you'll get used to it, eventually.

Command-Line Prompt

Most of bioinformatics is done with command-line software, so you should take some time to learn to use the shell effectively.

This is a command line prompt:

```
bush202>
```

This is another:

```
(~) 51%
```

This is another:

```
srobb@bush202 1:12PM>
```

What you get depends on how the system administrator has customized your login. You can customize yourself when you know how.

The prompt tells you the shell is ready to accept a command. When a long-running command is going, the prompt will not reappear until the system is ready to deal with your next request.

Issuing Commands

Type in a command and press the <Enter> key. If the command has output, it will appear on the screen. Example:

```
(~) 53% ls -F
GNUstep/          cool_elegans.movies.txt  man/
INBOX             docs/                   mtv/
INBOX~           etc/                    nsmail/
Mail@            games/                  pcod/
News/            get_this_book.txt       projects/
axhome/          jcod/                   public_html/
bin/             lib/                     src/
build/           linux/                   tmp/
ccod/
(~) 54%
```

The command here is `ls -F`, which produces a listing of files and directories in the current directory (more on which later). After its output, the command prompt appears again.

Some programs will take a long time to run. After you issue their command name, you won't recover the shell prompt until they're done. You can either launch a new shell (from Terminal's File menu), or run the command in the background using the ampersand:

```
(~) 54% long_running_application&
(~) 55%
```

The command will now run in the background until it is finished. If it has any output, the output will be printed to the terminal window. You may wish to redirect the output as described later.

Command Line Editing

Most shells offer command line editing. Up until the moment you press <Enter>, you can go back over the command line and edit it using the keyboard. Here are the most useful keystrokes:

Backspace

Delete the previous character and back up one.

Left arrow, right arrow

Move the text insertion point (cursor) one character to the left or right.

control-A (^A)

Move the cursor to the beginning of the line. Mnemonic: A is first letter of alphabet

control-E (^E)

Move the cursor to the end of the line. Mnemonic: <E> for the End (^Z was already taken for something else).

control-D (^D)

Delete the character currently under the cursor. D=Delete.

control-K (^K)

Delete the entire line from the cursor to the end. K=Kill. The line isn't actually deleted, but put into a temporary holding place called the "kill buffer".

control-Y (^Y)

Paste the contents of the kill buffer onto the command line starting at the cursor. Y=Yank.

Up arrow, down arrow

Move up and down in the command history. This lets you reissue previous commands, possibly after modifying them.

There are also some useful shell commands you can issue:

history

Show all the commands that you have issued recently, nicely numbered.

!*<number>*

Reissue an old command, based on its number (which you can get from *history*)

!!

Reissue the immediate previous command.

!*<partial command string>*

Reissue the previous command that began with the indicated letters. For example *!!* would reissue the *ls -F* command from the earlier example.

bash offers automatic command completion and spelling correction. If you type part of a command and then the tab key, it will prompt you with all the possible completions of the command. For example:

```
(~) 51% fd<tab>
```

```
(~) 51% fd
```

```
fd2ps    fdesign  fdformat fdlist   fdmount  fdmountd fdrawcmd fdumount
```

```
(~) 51%
```

If you hit tab after typing a command, but before pressing <Enter>, **bash** will prompt you with a list of file names. This is because many commands operate on files.

Wildcards

You can use wildcards when referring to files. "*" refers to zero or more characters. "?" refers to any single character. For example, to list all files with the extension ".txt", run **ls** with the pattern "*.txt":

```
(~) 56% ls -F *.txt
```

```
final_exam_questions.txt  genomics_problem.txt
```

```
genebridge.txt             mapping_run.txt
```

There are several more advanced types of wildcard patterns which you can read about in the **tcsh** manual page. For example, you can refer to files beginning with the characters "f" or "g" and ending with ".txt" this way:

```
(~) 57% ls -F [f-g]*.txt
```

```
final_exam_questions.txt  genebridge.txt
```

```
genomics_problem.txt
```

Home Sweet Home

When you first log in, you'll be placed in a part of the system that is your personal domain, called the *home directory*. You are free to do with this area what you will: in particular you can create and delete files and other directories. In general, you cannot create files elsewhere in the system.

Your home directory lives somewhere way down deep in the bowels of the system. On our iMacs, it is a directory with the same name as your login name, located in **/Users**. The full directory path is therefore **/Users/username**. Since this is a pain to write, the shell allows you to abbreviate it as `~username` (where "username" is your user name), or simply as `~`. The weird character (technically called the "twiddle") is usually hidden at the upper left corner of your keyboard.

To see what is in your home directory, issue the command `ls -F`:

```
(~) % ls -F
INBOX          Mail/          News/          nsmail/        public_html/
```

This shows one file "INBOX" and four directories ("Mail", "News") and so on. (The "-F" in the command turns on fancy mode, which appends special characters to directory listings to tell you more about what you're seeing. "/" means directory.)

In addition to the files and directories shown with `ls -F`, there may be one or more hidden files. These are files and directories whose names start with a "." (technically called the "dot" character). To see these hidden files, add an "a" to the options sent to the `ls` command:

```
(~) % ls -aF
./              .cshrc         .login         Mail/
../            .fetchhost    .netscape/   News/
.Xauthority    .fvwmrc       .xinitrc*     nsmail/
.Xdefaults     .history      .xsession@    public_html/
.bash_profile  .less         .xsession-errors
.bashrc        .lessrc       INBOX
```

Whoa! There's a lot of hidden stuff there. But don't go deleting dot files willy-nilly. Many of them are essential configuration files for commands and other programs. For example, the *.profile* file contains configuration information for the **bash** shell. You can peek into it and see all of **bash's** many options. You can edit it (when you know what you're doing) in order to change things like the command prompt and command search path.

Getting Around

You can move around from directory to directory using the `cd` command. Give the name of the directory you want to move to, or give no name to move back to your home directory. Use the `pwd` command to see where you are (or rely on the prompt, if configured):

```
(~/docs/grad_course/i) 56% cd
(~) 57% cd /
(/) 58% ls -F
bin/          docsc/         gmon.out      mnt/          sbin/
```

```

boot/          etc/          home@         net/          tmp/
cdrom/         fastboot     lib/          proc/         usr/
dev/           floppy/     lost+found/  root/         var/
(/) 59% cd ~/docs/
(~/docs) 60% pwd
/usr/home/lstein/docs
(~/docs) 62% cd ../projects/
(~/projects) 63% ls
Ace-browser/      bass.patch
Ace-perl/         cgi/
Foo/              cgi3/
Interface/       computertalk/
Net-Interface-0.02/ crypt-cbc.patch
Net-Interface-0.02.tar.gz fixer/
Pts/              fixer.tcsh
Pts.bak/          introspect.pl*
PubMed/           introspection.pm
SNPdb/            rhmap/
Tie-DBI/          sbbox/
ace/              sbbox-1.00/
atir/             sbbox-1.00.tgz
bass-1.30a/       zhmapper.tar.gz
bass-1.30a.tar.gz
(~/projects) 64%

```

Each directory contains two special hidden directories named "." and "..". "." refers always to the directory in which it is located. ".." refers always to the parent of the directory. This lets you move upward in the directory hierarchy like this:

```
(~/docs) 64% cd ..
```

and to do arbitrarily weird things like this:

```
(~/docs) 65% cd ../../docs
```

The latter command moves upward to levels, and then into a directory named "docs".

If you get lost, the *pwd* command prints out the full path to the current directory:

```
(~) 56% pwd
/Users/lstein
```

Essential Unix Commands

With the exception of a few commands that are built directly into the shell, all Unix commands are standalone executable programs. When you type the name of a command, the shell will search through all the directories listed in the PATH environment variable for an executable of the same name. If found, the shell will execute the command.

Otherwise, it will give a "command not found" error.

Most commands live in `/bin`, `/usr/bin`, or `/usr/local/bin`.

Getting Information About Commands

The **man** command will give a brief synopsis of the command:

```
(~) 76% man wc
Formatting page, please wait...
WC(1)                                                    WC(1)

NAME
    wc - print the number of bytes, words, and lines in files

SYNOPSIS
    wc [-clw] [--bytes] [--chars] [--lines] [--words] [--help]
    [--version] [file...]

DESCRIPTION
    This manual page documents the GNU version of wc.  wc
    counts the number of bytes, whitespace-separated words,
    ...
```

Finding Out What Commands are There

The **apropos** command will search for commands matching a keyword or phrase:

```
(~) 100% apropos column
showtable (1)      - Show data in nicely formatted columns
colrm (1)          - remove columns from a file
column (1)        - columnate lists
fix132x43 (1)     - fix problems with certain (132 column) graphics
modes
```

Arguments and Command Switches

Many commands take arguments. Arguments are often (but not inevitably) the names of one or more files to operate on. Most commands also take command-line "switches" or "options" which fine-tune what the command does. Some commands recognize "short switches" that consist of a single character, while others recognize "long switches" consisting of whole words.

The **wc** (word count) program is an example of a command that recognizes both long and short options. You can pass it the **-c**, **-w** and/or **-l** options to count the characters, words and lines in a text file, respectively. Or you can use the longer but more readable, **--chars**, **--words** or **--lines** options. Both these examples count the number of characters and lines in the text file `/var/log/messages`:

```
(~) 102% wc -c -l /var/log/messages
    23      941 /var/log/messages
```

```
(~) 103% wc --chars --lines /var/log/messages
      23      941 /var/log/messages
```

You can cluster short switches by concatenating them together, as shown in this example:

```
(~) 104% wc -cl /var/log/messages
      23      941 /var/log/messages
```

Many commands will give a brief usage summary when you call them with the **-h** or **--help** switch.

Spaces and Funny Characters

The shell uses whitespace (spaces, tabs and other nonprinting characters) to separate arguments. If you want to embed whitespace in an argument, put single quotes around it. For example:

```
mail -s 'An important message' 'Bob Ghost <bob@ghost.org>'
```

This will send an e-mail to the fictitious person Bob Ghost. The **-s** switch takes an argument, which is the subject line for the e-mail. Because the desired subject contains spaces, it has to have quotes around it. Likewise, my e-mail address, which contains embedded spaces, must also be quoted in this way.

Certain special non-printing characters have *escape codes* associated with them:

Escape Code	Description
<code>\n</code>	new line character
<code>\t</code>	tab character
<code>\r</code>	carriage return character
<code>\a</code>	bell character (ding! ding!)
<code>\nnn</code>	the character whose ASCII code in octal is nnn

Useful Commands

Here are some commands that are used extremely frequently. Use **man** to learn more about them. Some of these commands may be useful for solving the problem set ;-)

Manipulating Directories

ls	Directory listing. Most frequently used as ls -F (decorated listing) and ls -l (long listing).
mv	Rename or move a file or directory.
cp	Copy a file.
rm	Remove (delete) a file.
mkdir	Make a directory
rmdir	Remove a directory
ln	Create a symbolic or hard link.
chmod	

Change the permissions of a file or directory.

Manipulating Files

cat

Concatenate program. Can be used to concatenate multiple files together into a single file, or, much more frequently, to send the contents of a file to the terminal for viewing.

more

Scroll through a file page by page. Very useful when viewing large files. Works even with files that are too big to be opened by a text editor.

less

A version of **more** with more features.

head

View the head (top) of a file. You can control how many lines to view.

tail

View the tail (bottom) of a file. You can control how many lines to view. You can also use **tail** to view a growing file.

wc

Count words, lines and/or characters in one or more files.

tr

Substitute one character for another. Also useful for deleting characters.

sort

Sort the lines in a file alphabetically or numerically.

uniq

Remove duplicated lines in a file.

cut

Remove sections from each line of a file or files.

fold

Wrap each input line to fit in a specified width.

grep

Filter a file for lines matching a specified pattern. Can also be reversed to print out lines that don't match the specified pattern.

gzip (gunzip)

Compress (uncompress) a file.

tar

Archive or unarchive an entire directory into a single file.

emacs

Run the Emacs text editor (good for experts).

Networking

ssh

A secure (encrypted) way to log into machines.

ping

See if a remote host is up.

ftp and the secure version **sftp**

Transfer files using the File Transfer Protocol.

who

See who else is logged in.

lp

Send a file or set of files to a printer.

Standard I/O and Command Redirection

Unix commands communicate via the command line interface. They can print information out to the terminal for you

to see, and accept input from the keyboard (that is, from *you!*)

Every Unix program starts out with three connections to the outside world. These connections are called "streams" because they act like a stream of information (metaphorically speaking):

standard input

This is a communications stream initially attached to the keyboard. When the program reads from standard input, it reads whatever text you type in.

standard output

This stream is initially attached to the command window. Anything the program prints to this channel appears in your terminal window.

standard error

This stream is also initially attached to the command window. It is a separate channel intended for printing error messages.

The word "initially" might lead you to think that standard input, output and error can somehow be detached from their starting places and reattached somewhere else. And you'd be right. You can attach one or more of these three streams to a file, a device, or even to another program. This sounds esoteric, but it is actually very useful.

A Simple Example

The **wc** program counts lines, characters and words in data sent to its standard input. You can use it interactively like this:

```
(~) 62% wc
Mary had a little lamb,
little lamb,
little lamb.

Mary had a little lamb,
whose fleece was white as snow.
^D
      6      20      107
```

In this example, I ran the **wc** program. It waited for me to type in a little poem. When I was done, I typed the END-OF-FILE character, control-D (^D for short). **wc** then printed out three numbers indicating the number of lines, words and characters in the input.

More often, you'll want to count the number of lines in a big file; say a file filled with DNA sequences. You can do this by *redirecting* **wc**'s standard input from a file. This uses the < metacharacter:

```
(~) 63% wc <big_file.fasta
      2943      2998      419272
```

If you wanted to record these counts for posterity, you could redirect standard output as well using the > metacharacter:

```
(~) 64% wc <big_file.fasta >count.txt
```

Now if you **cat** the file *count.txt*, you'll see that the data has been recorded. **cat** works by taking its standard input and copying it to standard output. We redirect standard input from the *count.txt* file, and leave standard output at its default, attached to the terminal:

```
(~) 65% cat <count.txt
```

Redirection Meta-Characters

Here's the complete list of redirection commands for **bash**:

```
<filename      Redirect standard input to file
>filename      Redirect standard output to file
1>filename     Redirect just standard output to file (same as above)
2>filename     Redirect just standard error to file
>filename 2>&1 Redirect both stdout and stderr to file
```

These can be combined. For example, this command redirects standard input from the file named */etc/passwd*, writes its results into the file *search.out*, and writes its error messages (if any) into a file named *search.err*. What does it do? It searches the password file for a user named "root" and returns all lines that refer to that user.

```
(~) 66% grep root </etc/passwd >search.out 2>search.err
```

Filters, Filenames and Standard Input

Many Unix commands act as filters, taking data from a file or standard input, transforming the data, and writing the results to standard output. Most filters are designed so that if they are called with one or more filenames on the command line, they will use those files as input. Otherwise they will act on standard input. For example, these two commands are equivalent:

```
(~) 66% grep 'gatttgc' <big_file.fasta
(~) 67% grep 'gatttgc' big_file.fasta
```

Both commands use the **grep** command to search for the string "gatttgc" in the file *big_file.fasta*. The first one searches standard input, which happens to be redirected from the file. The second command is explicitly given the name of the file on the command line.

Sometimes you want a filter to act on a series of files, one of which happens to be standard input. Many filters let you use "-" on the command line as an alias for standard input. Example:

```
(~) 68% grep 'gatttgc' big_file.fasta bigger_file.fasta -
```

This example searches for "gatttgc" in three places. First it looks in *big_file.fasta*, then in *bigger_file.fasta*, and lastly in standard input (which, since it isn't redirected, will come from the keyboard).

Standard I/O and Pipes

The coolest thing about the Unix shell is its ability to chain commands together into pipelines. Here's an example:

```
(~) 65% grep gatttgc big_file.fasta | wc -l
22
```

There are two commands here. **grep** searches a file or standard input for lines containing a particular string. Lines which contain the string are printed to standard output. **wc -l** is the familiar word count program, which counts words,

lines and characters in a file or standard input. The `-l` command-line option instructs `wc` to print out just the line count. The `|` character, which is known as the "pipe" character, connects the two commands together so that the standard output of `grep` becomes the standard input of `wc`.

What does this pipe do? It prints out the number of lines in which the string "gatttgc" appears in the file *big_file.fasta*.

More Pipe Idioms

Pipes are very powerful. Here are some common command-line idioms.

Count the Number of Times a Pattern does NOT Appear in a File

The example at the top of this section showed you how to count the number of lines in which a particular string pattern appears in a file. What if you want to count the number of lines in which a pattern does **not** appear?

Simple. Reverse the test with the `grep -v` switch:

```
(~) 65% grep -v gatttgc big_file.fasta | wc -l
2921
```

Uniquify Lines in a File

If you have a long list of names in a text file, and you are concerned that there might be some duplicates, this will weed out the duplicates:

```
(~) 66% sort long_file.txt | uniq > unique.out
```

This works by sorting all the lines alphabetically and piping the result to the `uniq` program, which removes duplicate lines that occur together. The output is placed in a file named *unique.out*.

Concatenate Several Lists and Remove Duplicates

If you have several lists that might contain repeated entries among them, you can combine them into a single unique list by **catting** them together, then uniquifying them as before:

```
(~) 67% cat file1 file2 file3 file4 | sort | uniq
```

Count Unique Lines in a File

If you just want to know how many unique lines there are in the file, add a `wc` to the end of the pipe:

```
(~) 68% sort long_file.txt | uniq | wc -l
```

Page Through a Really Long Directory Listing

Pipe the output of `ls` to the `more` program, which shows a page at a time. If you have it, the `less` program is even better:

```
(~) 69% ls -l | more
```

Monitor a Rapidly Growing File for a Pattern

Pipe the output of `tail -f` (which monitors a growing file and prints out the new lines) to `grep`. For example, this will monitor the */var/log/syslog* file for the appearance of e-mails addressed to *mzhang*:

```
(~) 70% tail -f /var/log/syslog | grep mzhang
```

Beginning Perl Scripting

Simple scripts, Expressions, Operators, Statements, Variables

Simon Prochnik & Lincoln Stein

Suggested Reading

Chapters 1, 2 & 5 of *Learning Perl*.

Lecture Notes

1. [What is Perl?](#)
2. [Some simple Perl scripts](#)
3. [Mechanics of creating a Perl script](#)
4. [Statements](#)
5. [Literals](#)
6. [Operators](#)
7. [Functions](#)
8. [Variables](#)
9. [Processing the Command Line](#)

Problems

What is Perl?

Perl is a Programming Language

Written by Larry Wall in late 80's to process mail on Unix systems and since extended by a huge cast of characters. The name is said to stand for:

1. Pathologically Eclectic Rubbish Lister
2. Practical Extraction and Report Language

Perl Properties

1. Interpreted Language
2. "Object-Oriented"
3. Cross-platform
4. Forgiving
5. Great for text
6. Extensible, rich set of libraries
7. Popular for web pages
8. Extremely popular for bioinformatics

Other Languages Used in Bioinformatics

C, C++

Compiled languages, hence very fast.
Used for computation (BLAST, FASTA, Phred, Phrap, ClustalW)
Not very forgiving.

Java

Interpreted, fully object-oriented language.

Built into web browsers.
Supposed to be cross-platform, getting better.

Python , Ruby

Interpreted, fully object-oriented language.
Rich set of libraries.
Elegant syntax.
Smaller user community than Java or Perl.

Some Simple Scripts

Here are some simple scripts to illustrate the "look" of a Perl program.

Print a Message to the Terminal

Code:

```
#!/usr/bin/perl
# file: message.pl
use strict;
use warnings;
print "When that Aprill with his shoures soote\n";
print "The droghte of March ath perced to the roote,\n";
print "And bathed every veyne in swich licour\n";
print "Of which vertu engendered is the flour...\n";
```

Output:

```
(~) 50% perl message.pl
When that Aprill with his shoures soote
The droghte of March ath perced to the roote,
And bathed every veyne in swich licour
Of which vertu engendered is the flour...
```

Do Some Math

Code:

```
#!/usr/bin/perl
# file: math.pl
use strict;
use warnings;
print "2 + 2 =", 2+2, "\n";
print "log(1e23)= ", log(1e23), "\n";
print "2 * sin(3.1414)= ", 2 * sin(3.1414), "\n";
```

Output:

```
(~) 51% perl math.pl
2 + 2 =4
log(1e23)= 52.9594571388631
```

```
2 * sin(3.1414)= 0.000385307177203065
```

Run a System Command

Code:

```
#!/usr/bin/perl
# file: system.pl
use strict;
use warnings;
system "ls";
```

Output:

```
(~/docs/grad_course/perl) 52% perl system.pl
index.html          math.pl~          problem_set.html~  what_is_perl.html
index.html~         message.pl        simple.html        what_is_perl.html~
math.pl             problem_set.html  simple.html~
```

Return the Time of Day

Code:

```
#!/usr/bin/perl
# file: time.pl
use strict;
use warnings;
$time = localtime;
print "The time is now $time\n";
```

Output:

```
(~) 53% perl time.pl
The time is now Thu Sep 16 17:30:02 1999
```

Mechanics of Writing Perl Scripts

Some hints to help you get going.

Creating the Script

A Perl script is just a text file. Use any text (programmer's) editor. Don't use word processors like Word.

By convention, Perl script files end with the extension *.pl*.

The Emacs text editor has a *Perl mode* that will auto-format your Perl scripts and highlight keywords. Perl mode will be activated automatically if you end the script name with *.pl*.

Running the Script

Option 1 (quick)

Run the **perl** program from the command line, giving it the name of the script file to run.

```
(~) 50% perl time.pl
The time is now Thu Sep 16 18:09:28 1999
```

Option 2 (as shown in examples above)

Put the magic comment `#!/usr/bin/perl` at the top of the script. And always add `use strict;`
`use warnings;`
to the top of your script like in the example below

```
#!/usr/bin/perl
# file: time.pl
$time = localtime;
print "The time is now $time\n";
```

Now make the script executable with `chmod +x time.pl`:

```
(~) 51% chmod +x time.pl
```

Run the script as if it were a command:

```
(~) 52% ./time.pl
The time is now Thu Sep 16 18:12:13 1999
```

Note that you have to type `./time.pl` rather than `time.pl` because, by default, **bash** does not search the current directory for commands to execute. To avoid this, you can add the current directory (".") to your search PATH environment variable. To do this, create a file in your home directory named `.profile` and enter the following line in it:

```
export PATH=$PATH:.
```

The next time you log in, your path will contain the current directory and you can type `time.pl` directly.

Common Errors

Every script goes through a few iterations before you get it right. Here are some common errors:

Syntax Errors

Code:

```
#!/usr/bin/perl
# file: time.pl
use strict;
use warnings;
time = localtime;
print "The time is now $time\n";
```

Output:

(~) 53% **time.pl**

Can't modify time in scalar assignment at time.pl line 3, near "localtime;"
Execution of time.pl aborted due to compilation errors.

Runtime Errors

Code:

```
#!/usr/bin/perl
# file: math.pl
use strict;
use warnings;

$six_of_one = 6;
$half_dozen = 12/2;
$result = $six_of_one/($half_dozen - $six_of_one);
print "The result is $result\n";
```

Output:

(~) 54% **math.pl**

Illegal division by zero at math.pl line 6.

Forgetting to Make the Script Executable

(~) 55% **test.pl**

test.pl: Permission denied.

Getting the Path to Perl Wrong on the #! line

Code:

```
#!/usr/local/bin/pearl
# file: time.pl
use strict;
use warnings;
$time = localtime;
print "The time is now $time\n";
```

(~) 55% **time.pl**

time.pl: Command not found.

This gives a very confusing error message because the command that wasn't found is 'pearl' not time.pl

Useful Perl Command-Line Options

You can call Perl with a few command-line options to help catch errors:

-c

Perform a syntax check, but don't run.

-w Turn on verbose warnings. Same as

```
use warnings;
```

-d Turn on the Perl debugger.

Usually you will invoke these from the command-line, as in `perl -cw time.pl` (syntax check `time.pl` with verbose warnings). You can also put them in the top line: `#!/usr/bin/perl -w`.

Perl Statements

A Perl script consists of a series of *statements* and *comments*. Each statement is a command that is recognized by the Perl interpreter and executed. Statements are terminated by the semicolon character (;). They are also usually separated by a newline character to enhance readability.

A *comment* begins with the # sign and can appear anywhere. Everything from the # to the end of the line is ignored by the Perl interpreter. Commonly used for human-readable notes.

Some Statements

```
$sum = 2 + 2; # this is a statement
```

```
$f = <STDIN>; $g = $f++; # these are two statements
```

```
$g = $f  
/  
$sum; # this is one statement, spread across 3 lines
```

The Perl interpreter will start at the top of the script and execute all the statements, in order from top to bottom, until it reaches the end of the script. This execution order can be modified by loops and control structures.

Blocks

It is common to group statements into *blocks* using curly braces. You can execute the entire block conditionally, or turn it into a *subroutine* that can be called from many different places.

Example blocks:

```
{ # block starts  
  my $EcoRI = 'GAATTC';  
  my $sequence = <STDIN>;  
  print "Sequence contains an EcoRI site" if $sequence=~/$EcoRI/;  
} # block ends
```

```
my $sequence2 = <STDIN>;  
if (length($sequence) < 100) { # another block starts  
  print "Sequence is too small. Throw it back\n";  
  exit 0;  
} # and ends
```

```
foreach $sequence (@sequences) { # another block
    print "sequence length = ",length($sequence),"\n";
}
```

Literals

Literals are constant values that you embed directly in the program code. Perl supports both *string literals* and *numeric literals*.

String Literals

String literals are enclosed by single quotes (') or double quotes ("):

```
'The quality of mercy is not strained.'; # a single-quoted string
"The quality of mercy is not strained."; # a double-quoted string
```

The difference between single and double-quoted strings is that variables and certain special escape codes are interpolated into double quoted strings, but not in single-quoted ones. Here are some escape codes:

<code>\n</code>	New line
<code>\t</code>	Tab
<code>\r</code>	Carriage return
<code>\f</code>	Form feed
<code>\a</code>	Ring bell
<code>\040</code>	Octal character (octal 040 is the space character)
<code>\0x2a</code>	Hexadecimal character (hex 2A is the "*" character)
<code>\cA</code>	Control character (This is the ^A character)
<code>\u</code>	Uppercase next character
<code>\l</code>	Lowercase next character
<code>\U</code>	Uppercase everything until \E
<code>\L</code>	Lowercase everything until \E
<code>\Q</code>	Quote non-word characters until \E
<code>\E</code>	End \U, \L or \Q operation

```
"Here goes\n\tnothing!";
# evaluates to:
# Here goes
#   nothing!
```

```
'Here goes\n\tnothing!';
```

```
# evaluates to:  
# Here goes\n\tnothing!
```

```
"Here goes \unothing!";  
# evaluates to:  
# Here goes Nothing!
```

```
"Here \Ugoes nothing\E";  
# evaluates to:  
# Here GOES NOTHING!
```

```
"Alert! \a\a\a";  
# evaluates to:  
# Alert! (ding! ding! ding!)
```

Putting backslashes in strings is a problem because they get interpreted as escape sequences. To include a literal backslash in a string, double it:

```
"My file is in C:\\Program Files\\Accessories\\wordpad.exe";  
  
# evaluates to: C:\Program Files\Accessories\wordpad.exe
```

Put a backslash in front of a quote character in order to make the quote character part of the string:

```
"She cried \"Oh dear! The parakeet has flown the coop!\"";  
  
# evaluates to: She cried "Oh dear! The parakeet has flown the coop!"
```

Numeric Literals

You can refer to numeric values using integers, floating point numbers, scientific notation, hexadecimal notation, and octal. With some help from the `Math::Complex` module, you can refer to complex numbers as well:

```
123;      # an integer  
  
1.23;     # a floating point number  
  
-1.23;    # a negative floating point number  
  
1_000_000; # you can use _ to improve readability  
  
1.23E45;  # scientific notation  
  
0x7b;     # hexadecimal notation (decimal 123)  
  
0173;     # octal notation (decimal 123)
```

```
use Math::Complex; # bring in the Math::Complex module
```

```
12+3*i; # complex number 12 + 3i
```

Backtick Strings

You can also enclose a string in backticks (```). This has the unusual property of executing whatever is inside the string as a Unix system command, and returning its output:

```
`ls -l`;
# evaluates to a string containing the output of running the
# ls -l command
```

Lists

The last type of literal that Perl recognizes is the *list*, which is multiple values strung together using the comma operator (`,`) and enclosed by parentheses. Lists are closely related to *arrays*, which we talk about later.

```
('one', 'two', 'three', 1, 2, 3, 4.2);
# this is 7-member list contains a mixture of strings, integers
# and floats
```

Operators

Perl has numerous *operators* (over 50 of them!) that perform operations on string and numeric values. Some operators will be familiar from algebra (like "+", to add two numbers together), while others are more esoteric (like the "." string concatenation operator).

Numeric & String Operators

The "." operator acts on strings. The "!" operator acts on strings and numbers. The rest act on numbers.

Operator	Description	Example	Result
.	String concatenate	'Teddy' . 'Bear'	TeddyBear
=	Assignment	\$a = 'Teddy'	\$a variable contains 'Teddy'
+	Addition	3+2	5
-	Subtraction	3-2	1
-	Negation	-2	-2
!	Not	!1	0
*	Multiplication	3*2	6
/	Division	3/2	1.5
%	Modulus	3%2	1

**	Exponentiation	3**2	9
<FILEHANDLE>	File input	<STDIN>	Read a line of input from standard input
>>	Right bit shift	3>>2	0 (binary 11>>2=00)
<<	Left bit shift	3<<2	12 (binary 11<<2=1100)
	Bitwise OR	3 2	3 (binary 11 10=11)
&	Bitwise AND	3&2	2 (binary 11&10=10)
^	Bitwise XOR	3^2	1 (binary 11^10=01)

Operator Precedence

When you have an expression that contains several operators, they are evaluated in an order determined by their *precedence*. The precedence of the mathematical operators follows the rules of arithmetic. Others follow a precedence that usually does what you think they should do. If uncertain, use parentheses to force precedence:

2+3*4; # evaluates to 14, multiplication has precedence over addition
 (2+3)*4; # evaluates to 20, parentheses force the precedence

Logical Operators

These operators compare strings or numbers, returning TRUE or FALSE:

Numeric Comparison		String Comparison	
3 == 2	equal to	'Teddy' eq 'Bear'	equal to
3 != 2	not equal to	'Teddy' ne 'Bear'	not equal to
3 < 2	less than	'Teddy' lt 'Bear'	less than
3 > 2	greater than	'Teddy' gt 'Bear'	greater than
3 <= 2	less or equal	'Teddy' le 'Bear'	less than or equal
3 >= 2	greater than or equal	'Teddy' ge 'Bear'	greater than or equal
3 <=> 2	compare	'Teddy' cmp 'Bear'	compare
		'Teddy' =~ /Bear/	pattern match

The <=> and **cmp** operators return:

- -1 if the left side is less than the right side
- 0 if the left side equals the right side
- +1 if the left side is greater than the right side

File Operators

Perl has special *file operators* that can be used to query the file system. These operators generally return TRUE or FALSE.

Example:

```
print "Is a directory!\n" if -d '/usr/home';
print "File exists!\n" if -e '/usr/home/lstein/test.txt';
```

```
print "File is plain text!\n" if -T '/usr/home/lstein/test.txt';
```

There are many of these operators. Here are some of the most useful ones:

-e filename	file exists
-r filename	file is readable
-w filename	file is writable
-x filename	file is executable
-z filename	file has zero size
-s filename	file has nonzero size (returns size)
-d filename	file is a directory
-T filename	file is a text file
-B filename	file is a binary file
-M filename	age of file in days since script launched
-A filename	same for access time

Functions

In addition to its operators, Perl has many *functions*. Functions have a human-readable name, such as **print** and take one or more arguments passed as a list. A function may return no value, a single value (AKA "scalar"), or a list (AKA "array"). You can enclose the argument list in parentheses, or leave the parentheses off.

A few examples:

```
# The function is print. Its argument is a string.
# The effect is to print the string to the terminal.
print "The rain in Spain falls mainly on the plain.\n";

# Same thing, with parentheses.
print("The rain in Spain falls mainly on the plain.\n");

# You can pass a list to print. It will print each argument.
# This prints out "The rain in Spain falls 6 times in the plain."
print "The rain in Spain falls ",2*4-2," times in the plain.\n";

# Same thing, but with parentheses.
print ("The rain in Spain falls ",2*4-2," times in the plain.\n");

# The length function calculates the length of a string,
# yielding 45.
length "The rain in Spain falls mainly on the plain.\n";

# The split function splits a string based on a delimiter pattern
```



```
# yielding the list ('The','rain in Spain','falls mainly','on the plain.')
split '/', 'The/rain in Spain/falls mainly/on the plain.';
```

Often Used Functions (alphabetic listing)

For specific information on a function, use `perldoc -f function_name` to get a concise summary.

abs	absolute value
chdir	change current directory
chmod	change permissions of file/directory
chomp	remove terminal newline from string variable
chop	remove last character from string variable
chown	change ownership of file/directory
close	close a file handle
closedir	close a directory handle
cos	cosine
defined	test whether variable is defined
delete	delete a key from a hash
die	exit with an error message
each	iterate through keys & values of a hash
eof	test a filehandle for end of file
eval	evaluate a string as a perl expression
exec	quit Perl and execute a system command
exists	test that a hash key exists
exit	exit from the Perl script
glob	expand a directory listing using shell wildcards
gmtime	current time in GMT
grep	filter an array for entries that meet a criterion
index	find location of a substring inside a larger string
int	throw away the fractional part of a floating point number
join	join an array together into a string
keys	return the keys of a hash
kill	send a signal to one or more processes
last	exit enclosing loop
lc	convert string to lowercase
lcfirst	lowercase first character of string

length	find length of string
local	temporarily replace the value of a global variable
localtime	return time in local timezone
log	natural logarithm
m//	pattern match operation
map	perform an operation on each member of array or list
mkdir	make a new directory
my	create a local variable
next	jump to the top of enclosing loop
open	open a file for reading or writing
opendir	open a directory for listing
pack	pack a list into a compact binary representation
package	create a new namespace for a module
pop	pop the last item off the end of an array
print	print to terminal or a file
printf	formatted print to a terminal or file
push	push a value onto the end of an array
q/STRING/	generalized single-quote operation
qq/STRING/	generalized double-quote operation
qx/STRING/	generalized backtick operation
qw/STRING/	turn a space-delimited string of words into a list
rand	random number generator
read	read binary data from a file
readdir	read the contents of a directory
readline	read a line from a text file
readlink	determine the target of a symbolic link
redo	restart a loop from the top
ref	return the type of a variable reference
rename	rename or move a file
require	load functions defined in a library file
return	return a value from a user-defined subroutine
reverse	reverse a string or list
rewinddir	rewind a directory handle to the beginning

rindex	find a substring in a larger string, from right to left
rmdir	remove a directory
s///	pattern substitution operation
scalar	force an expression to be treated as a scalar
seek	reposition a filehandle to an arbitrary point in a file
select	make a filehandle the default for output
shift	shift a value off the beginning of an array
sin	sine
sleep	put the script to sleep for a while
sort	sort an array or list by user-specified criteria
splice	insert/delete array items
split	split a string into pieces according to a pattern
sprintf	formatted string creation
sqrt	square root
stat	get information about a file
sub	define a subroutine
substr	extract a substring from a string
symlink	create a symbolic link
system	execute an operating system command, then return to Perl
tell	return the position of a filehandle within a file
tie	associate a variable with a database
time	return number of seconds since January 1, 1970
tr///	replace characters in a string
truncate	truncate a file (make it smaller)
uc	uppercase a string
ucfirst	uppercase first character of a string
umask	change file creation mask
undef	undefine (remove) a variable
unlink	delete a file
unpack	the reverse of pack
untie	the reverse of tie
unshift	move a value onto the beginning of an array
use	import variables and functions from a library module
values	return the values of a hash variable

wantarray	return true in an array context
warn	print a warning to standard error
write	formatted report generation

Creating Your Own Functions

You can define your own functions or redefine the built-in ones using the **sub** function. This is described in more detail in the lesson on creating subroutines, which you'll be seeing soon..

Variables

A variable is a symbolic placeholder for a value, a lot like the variables in algebra. Perl has several built-in variable types:

Scalars: **\$variable_name**

A single-valued variable, always preceded by a \$ sign.

Arrays: **@array_name**

A multi-valued variable indexed by integer, preceded by an @ sign.

Hashes: **%hash_name**

A multi-valued variable indexed by string, preceded by a % sign.

Filehandle: **FILEHANDLE_NAME**

A file to read and/or write from. Filehandles have no special prefix, but are usually written in all uppercase.

We discuss arrays, hashes and filehandles later.

Scalar Variables

Scalar variables have names beginning with \$. The name must begin with a letter or underscore, and can contain as many letters, numbers or underscores as you like. These are all valid scalars:

- \$foo
- \$The_Big_Bad_Wolf
- \$R2D2
- \$_____A23
- \$Once_Upon_a_Midnight_Dreary_While_I_Pondered_Weak_and_Weary

You assign values to a scalar variable using the = operator (not to be confused with ==, which is numeric comparison). You read from scalar variables by using them wherever a value would go.

A scalar variable can contain strings, floating point numbers, integers, and more esoteric things. You don't have to predeclare scalars. A scalar that once held a string can be reused to hold a number, and vice-versa:

Code:

```
$p = 'Potato'; # $p now holds the string "potato"
$bushels = 3; # $bushels holds the value 3
$potatoes_per_bushel = 80; # $potatoes_per_bushel contains 80;

$total_potatoes = $bushels * $potatoes_per_bushel; # 240

print "I have $total_potatoes $p\n";
```

Output:

I have 240 Potato

Scalar Variable String Interpolation

The example above shows one of the interesting features of double-quoted strings. If you place a scalar variable inside a double quoted string, it will be interpolated into the string. With a single-quoted string, no interpolation occurs.

To prevent interpolation, place a backslash in front of the variable:

```
print "I have \$total_potatoes \$p\n";  
  
# prints: I have $total_potatoes $p
```

Operations on Scalar Variables

You can use a scalar in any string or numeric expression like `$hypotenuse = sqrt($x**2 + $y**2)` or `$name = $first_name . ' ' . $last_name`. There are also numerous shortcuts that combine an operation with an assignment:

\$a++

Increment \$a by one

\$a--

Decrement \$a by one

\$a += \$b

Modify \$a by adding \$b to it.

\$a -= \$b

Modify \$a by subtracting \$b from it.

\$a *= \$b

Modify \$a by multiplying \$b to it.

\$a /= \$b

Modify \$a by dividing it by \$b.

\$a .= \$b

Modify the **string** in \$a by appending \$b to it.

Example Code:

```
$potatoes_per_bushel = 80; # $potatoes_per_bushel contains 80;  
  
$p = 'one';  
$p .= ' '; # append a space  
$p .= 'potato'; # append "potato"  
  
$bushels = 3;  
$bushels *= $potatoes_per_bushel; # multiply  
  
print "From $p come $bushels.\n";
```

Output:

From one potato come 240.

String Functions that Come in Handy for Dealing with Sequences

Reverse the Contents of a String

```
$name          = 'My name is Lincoln';
$reversed_name = reverse $name;
print $reversed_name, "\n";
# prints "nlocniL si eman yM"
```

Translating one set of letters into another set

```
$name = 'My name is Lincoln';
# swap a->g and c->t
$name =~ tr/ac/gt/;
print $name, "\n";
# prints "My ngme is LintoIn"
```

Can you see how a combination of these two operators might be useful for computing the reverse complement?

Processing Command Line Arguments

When a Perl script is run, its command-line arguments (if any) are stored in an automatic array called **@ARGV**. You'll learn how to manipulate this array later. For now, just know that you can call the **shift** function repeatedly from the main part of the script to retrieve the command line arguments one by one.

Printing the Command Line Argument

Code:

```
#!/usr/bin/perl
# file: echo.pl
use strict;
use warnings;
$argument = shift;
print "The first argument was $argument.\n";
```

Output:

```
(~) 50% chmod +x echo.pl
(~) 51% echo.pl tuna
The first argument was tuna.
(~) 52% echo.pl tuna fish
```

The first argument was tuna.

```
(~) 53% echo.pl 'tuna fish'
```

The first argument was tuna fish.

```
(~) 53% echo.pl
```

The first argument was.

Computing the Hypotenuse of a Right Triangle

Code:

```
#!/usr/bin/perl
# file: hypotenuse.pl
use strict;
use warnings;
$x = shift;
$y = shift;
$x>0 and $y>0 or die "Must provide two positive numbers";

print "Hypotenuse=",sqrt($x**2+$y**2),"\n";
```

Output:

```
(~) 82% hypotenuse.pl
Must provide two positive numbers at hypotenuse.pl line 6.
(~) 83% hypotenuse.pl 1
Must provide two positive numbers at hypotenuse.pl line 6.
(~) 84% hypotenuse.pl 3 4
Hypotenuse=5
(~) 85% hypotenuse.pl 20 18
Hypotenuse=26.9072480941474
(~) 86% hypotenuse.pl -20 18
Must provide two positive numbers at hypotenuse.pl line 6.
```

Perl II

Operators, truth, control structures, functions, and
processing the command line

Dave Messina

say

Most of the time when you print, you will end the print statement with a newline (`\n`). `say` is shorthand for that.

These statements are equivalent:

```
print "x is $x\n";  
say   "x is $x";
```

say

But for `say` to work, you have to have the line

```
use 5.10.0;
```

in your script. There are other things we will teach you that need `use 5.10.0;`, too,

```
use strict;  
use warnings;  
use 5.10.0;
```

```
say "x is $x";
```

Math

1 + 2 = 3 # kindergarten

$x = 1 + 2$ # algebra

```
my $x = 1 + 2; # Perl
```

What are the differences between the algebra version and the Perl version?

Math

my \$x = 5;

my \$y = 2;

my \$z = \$x + \$y;

Math

my \$sum = \$x + \$y;

my \$difference = \$x - \$y;

my \$product = \$x * \$y;

my \$quotient = \$x / \$y;

my \$remainder = \$x % \$y;

Math

```
my $x = 5;
```

```
my $y = 2;
```

```
my $sum = $x + $y;
```

```
my $product = $x - $y;
```

Variable names are arbitrary. Pick good ones!

What are these called?

my \$sum = \$x + \$y ;
my \$difference = \$x - \$y ;
my \$product = \$x * \$y ;
my \$quotient = \$x / \$y ;
my \$remainder = \$x % \$y ;

Numeric operators

Operator

Meaning

+	add 2 numbers
-	subtract left number from right number
*	multiply 2 numbers
/	divide left number from right number
%	divide left from right and take remainder
**	take left number to the power of the right number

Numeric comparison operators

Operator

Meaning

<	Is left number smaller than right number?
>	Is left number bigger than right number?
<=	Is left number smaller or equal to right?
>=	Is left number bigger or equal to right?
==	Is left number equal to right number?
!=	Is left number not equal to right number?

Why == ?

Comparison operators are **yes** or **no** questions

or, put another way, **true** or **false** questions

True or false:

> Is left number smaller than right number?

2 > 1 # true

1 > 3 # false

Comparison operators are **true** or **false** questions

5 > 3

-1 <= 4

5 == 5

7 != 4

What is truth?

`0` the number 0 is false

`"0"` the string 0 is false

`""` and `' '` an empty string is false

`my $x;` an undefined variable is false

everything else is **true**

Examples of truth

```
my $a;           # FALSE (not yet defined)
$x = 1;          # TRUE
$x = 0;          # FALSE
$x = "";         # FALSE
$x = 'true';     # TRUE
$x = 'false';    # TRUE (watch out! "false" is a nonempty string)
$x = ' ';        # TRUE (a single space is non-empty)
$x = "\n";       # TRUE (a single newline is non-empty)
$x = 0.0;        # FALSE (converts to string "0")
$x = '0.0';      # TRUE (watch out! The string "0.0" is not the
                  # same as "0")
```

Sidebar: = vs ==

- 1 equals sign to *make* the left side equal the right side.
- 2 equals signs to *test* if the left side is equal to the right.

```
my $x;           # x is undefined
my $x = 1;       # x is now defined
if ($x == 1)     # is $x equal to 1?
if ($x = 1)      # (wrong)
```

use warnings will catch this error.

Logical operators

Use and and or to combine comparisons.

Operator

Meaning

and

TRUE if left side is TRUE and right side is TRUE

or

TRUE if left side is TRUE or right side is TRUE

Logical operator examples

```
if ($i < 100 and $i > 0) {  
    say "$i is the right size";  
}  
else {  
    say "out of bounds error!";  
}
```

```
if ($age < 10 or $age > 65) {  
    say "Your movie ticket is half price!";  
}
```

Let's test some more

Logical operators

Use `not` to reverse the truth.

```
$ok = ($i < 100 and $i > 0);  
print "a is too small\n" if not $ok;
```

```
# same as this:  
print "a is too small\n" unless $ok;
```

defined and undef

defined lets you test whether a variable is defined.

```
if (defined $x) {  
    say "$x is defined";  
}
```

undef lets you empty a variable, making it undefined.

```
undef $x;  
say $x if defined $x;
```

if not

Testing for defined-ness:

```
if (defined $x) {  
    say "$x is defined";  
}
```

What if you wanted to test for undefined-ness?

```
if (not defined $x) {  
    say "x is undefined";  
}
```

if not

or you could use unless:

```
unless (defined $x) {  
    say "$x is undefined";  
}
```

Sidebar: operator precedence

Some operators have higher precedence than others.

```
my $result = 3 + 2 * 5;
```

```
# force addition before multiplication  
my $result = (3 + 2) * 5 = 25;
```

The universal precedence rule is this:
multiplication comes before addition,
use parentheses for everything else.

String operators

Operator

Meaning

eq	Is the left string same as the right string?
ne	Is the left string not the same as the right string?
lt	Is the left string alphabetically before the right?
gt	Is the left string alphabetically after the right?
.	add the right string to the end of the left string

String operator examples

```
my $his_first = 'Barry';
my $his_last  = 'White';
my $her_first = 'Betty';
my $her_last  = 'White';

my $his_full = $his_first . ' ' . $his_last;
if ($his_last eq $her_last) {
    print "Same\n";
}
if ($his_first lt $her_first) {
    print "$his_first before $her_first\n";
}
```

Comparing numeric and string operators

Numeric	Meaning	String
==	equal to	eq
!=	not equal to	ne
>	greater than	gt
<	less than	lt
+	addition/concatenation	.

Control structures

Control structures allow you to control if and how a line of code is executed.

You can create alternative branches in which different sets of statements are executed depending on the circumstances.

You can create various types of repetitive loops.

Control structures

So far you've seen a basic program, where every line is executed, in order, and only once.

```
my $x = 1;  
my $y = 2;  
my $z = $x + $y;  
print "$x + $y = $z\n";
```

Control structures

Here, the print statement is only executed some of the time.

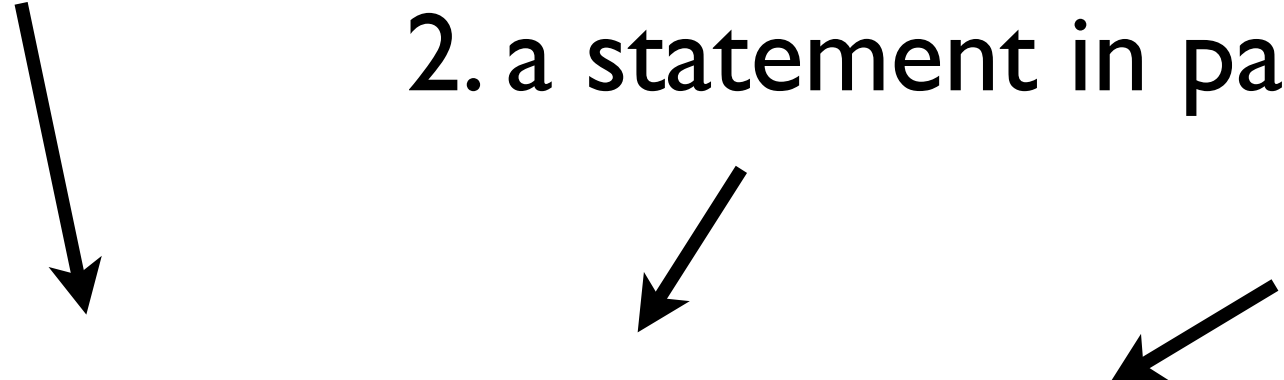
```
my $x = 1;  
my $y = 2;  
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

Components of a control structure

1. a keyword

2. a statement in parentheses

3. squiggly brackets



```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

The part enclosed by the squiggly brackets is called a **block**.

Components of a control structure

When you program, build the structure first and then fill in.

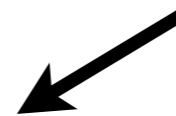
1. a keyword



2. a statement in parentheses



3. squiggly brackets



```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

4. now add the print statement

if

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}
```

If $\$x$ is the same as $\$y$, then the print statement will be executed.

or said another way:

If $(\$x == \$y)$ is **true**, then the print statement will be executed.

if — a common mistake

```
if ($x = $y) {  
    print "$x and $y are equal\n";  
}
```

What happens if we write it this way?

else

If the `if` statement is **false**, then the first `print` statement will be skipped and only the second `print` statement will be executed.

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}  
else {  
    print "$x and $y aren't equal\n";  
}
```


elseif

Sometimes you want to test a series of conditions.

```
if ($x == $y) {  
    print "$x and $y are equal\n";  
}  
elseif ($x > $y) {  
    print "$x is bigger than $y\n";  
}  
elseif ($x < $y) {  
    print "$x is smaller than $y\n";  
}
```

elseif

What if more than one condition is true?

```
if (1 == 1) {  
    print "$x and $y are equal\n";  
}  
elseif (2 > 0) {  
    print "2 is positive\n";  
}  
elseif (2 < 10) {  
    print "2 is smaller than 10\n";  
}
```

given-when

is another way to test a series of conditions
(whose full power you'll learn later).

```
my $x = 3;
given($x) {
    when ($x % 2 == 0) {
        say '$x is even';
    }
    when ($x < 10) {
        say '$x is less than 10';
    }
    default {
        die q(I don't know what to do with $x);
    }
}
```

unless

It's exactly the opposite of `if (something) *`
These statements are equivalent:

```
if ($x > 0) {  
    print "$x is positive\n";  
}  
unless ($x < 0) {  
    print "$x is positive\n";  
}
```

If the statement `($x < 0)` is **false**, then the print statement will be executed.

*except you can't `unless..else` or `unless..elsif`

while

As long as `($x == $y)` is **true**, the print statement will be executed over and over again.

```
while ( $x == $y ) {  
    print "$x and $y are equal\n";  
}
```

Why might you want to execute a block repeatedly?

one line conditionals

An alternative form that sometimes reads better. The conditional comes at the end and parentheses are optional.

```
print "x is less than y\n" if $x < $y;  
print "x is less than y\n" unless $x >= $y;
```

However, you can execute only one statement because there's no longer brackets to enclose multiple lines. Only works for `if` and `unless`.

functions

Functions are like operators — they do something with the data you give them. They have a human-readable name, such as print and take one or more arguments.

```
print "The rain in Spain falls mainly on the plain.\n";
```

functions

The function is `print`. Its argument is a string. The effect is to print the string to the terminal.

```
print "The rain in Spain falls mainly on the plain.\n";
```


functions

You can enclose the argument list in parentheses, or leave the parentheses off.

```
# Same thing, with parentheses.  
print("The rain in Spain falls mainly on the plain.\n");
```

function examples

You can pass multiple values separated by commas to print, and it will print each argument.

```
# This prints out "The rain in Spain falls 6 times in the plain."  
print "The rain in Spain falls ", 2*4-2, " times in the plain.\n";
```

```
# Same thing, but with parentheses.  
print ("The rain in Spain falls ", 2*4-2, " times in the plain.\n");
```

functions

A function may return no value, a single value, or multiple values.

```
# print returns nothing.  
print "The rain in Spain falls mainly on the plain.\n";  
  
# The length function calculates the length of a string  
# and returns the answer.  
  
my $length = length "The rain in Spain falls mainly on the  
plain.\n";
```

processing the command line

Often when you run a program, you want to pass it some information. For example, some numbers, or a filename.

These are called **arguments**.

```
$ add 1 2
```

```
$ parse_blast.pl mydata.blast
```

What are the command-line arguments in these examples?

processing the command line

You can give arguments to Perl programs you write, and you can see those arguments inside your script using the shift function.

```
#!/usr/bin/perl  
  
my $arg1 = shift;  
my $arg2 = shift;  
say "my command-line arguments were $arg1 and $arg2";
```

Perl III

File I/O, more on system calls

Dave Messina

File I/O

I/O stands for input/output.

It's how get computer programs talk to the rest of the world.

Perl has magic

Perl has a magic way that makes it super easy to get data from files and into your program.

It looks like this: <>



<> will:

read filenames that are arguments on the
command line

open each file in turn

read each line from the file



```
#!/usr/bin/perl
# how to read a file with <>
use warnings;
use strict;

while (my $line = <>) {
    chomp $line;
    print "Here's a line: ", $line, "\n";
}
```

Sidebar: chomp

chomp removes the newline from the end of a string (if there is a newline).

```
my $string = "hey there!\n";  
print "my string is: ", $string, "\n";  
chomp $string;  
print "after chomp : ", $string, "\n";
```

When you read a file, the first thing you always want to do is chomp.



Let's make a file and read from it.
We'll call it myfile.txt

```
% perl read_from_file.pl myfile.txt
```

And now we're giving the name myfile.txt as a
command-line argument to our Perl script.

<> line count

Let's do something more interesting than printing the line back out. Let's count how many lines there are in the file.

```
my $line_count;
while (my $line = <>) {
    chomp $line;
    $line_count++;
}
say "There are $line_count lines";
```

Sidebar: increment operators

Yesterday we learned several numeric operators. Here are a couple more common ones:

`++` the increment operator

```
my $x = 1;  
$x++;          # add 1 to $x
```

```
# exactly the same as  
$x = $x + 1;
```

Sidebar: decrement operators

-- the decrement operator

```
my $x = 1;
```

```
$x--;          # subtract 1 from $x
```

exactly the same as

```
$x = $x - 1;
```

<> line count

With ++, we're counting each time we go through the loop.

```
my $line_count;
while (my $line = <>) {
    chomp $line;
    $line_count++;
}
say "There are $line_count lines";
```


<> multiple files

If there is more than one argument, each one is opened and read completely, one after the other.

```
% perl read_from_file.pl myfile.txt another.txt
```

So let's create another file and try it.

<> mistakes

Remember how yesterday we had command-line arguments that were numbers?

Does Perl know that the arguments are files?

```
% perl read_from_file.pl 2 9
```

Let's try it and see what happens.

the input loop

Let's step back for a moment and think about why `<>` works. What is `while`? What is it testing?

```
my $line_count;
while (my $line = <>) {
    chomp $line;
    $line_count++;
}
say "There are $line_count lines";
```

the input loop

What exactly is going on on this line?

```
while (my $line = <>) {
```

The `<>` is a function.

It returns a line of input.

We assign that line to a variable, `$line`.

While tests that assignment for truth:

"Can we assign a value to `$line`?"

the input loop

If there is another line in the file, the answer is "yes, we can, it's TRUE."

If we've hit the end of the file, there are no more lines to read, and so the answer is "no", or FALSE.

When the expression in parentheses is false, we exit the loop.

the input loop

Once we've exited the loop, the say statement gets executed.

```
my $line_count;
while (my $line = <>) {
    chomp $line;
    $line_count++;
}
say "There are $line_count lines";
```

the input loop

To summarize:

The while loop will read one line of text after another. At the end of input, the `<>` operator returns undef and the while loop terminates.

Remember that even blank lines in a file are TRUE, because they consist of a single newline character.

STDOUT and STDERR

Every Perl script by default has two places it knows where to write to:

STDOUT and STDERR

STDOUT and STDERR

STDOUT

Standard output, used to write data out. Initially connected to the terminal, but can be redirected to a file or other program from the shell using redirection or pipes.

STDOUT and STDERR

STDERR

Standard error, used for diagnostic messages. Initially connected to the terminal.

STDOUT and STDERR

You've actually been usually `STDOUT` all along. It's the default place where your program's output goes.

When you use `say` or `print`, you're actually writing to `STDOUT`.

These are equivalent:

```
say "Well, how did I get here?";
```

```
say STDOUT "Well, how did I get here?";
```

STDOUT and STDERR

But you can also specify other places to write to.

Like STDERR:

```
say STDOUT "You may ask yourself:";  
say STDERR "Well, how did I get here?";
```

STDOUT and STDERR

At first it looks exactly the same as STDOUT, but if we use output redirection on the command line, we can see that the output is actually going to a different place:

```
$ perl test.pl > output.txt
```

Well, how did I get here?

open for reading

<> is great, but often you want to read from a specific file. You can do that using open.

```
my $file = shift;  
open(FILE, '<', $file) or die "can't open $file: $!\n";
```

open

```
my $file = shift;  
open(FILE, '<', $file) or die "can't open $file: $!\n";
```

Let's break this down into pieces:

```
my $file = shift;
```

reads the filename from the command line.

open

```
open(FILE, '<', $file)
```

open is a function, which is taking 3 arguments:

The first argument is a filehandle. Filehandles are how you refer to a file within Perl. **STDOUT** and **STDERR** are filehandles.

open

```
open(FILE, '<', $file)
```

The second argument is a mode. The modes are borrowed from redirection on the command line.

- < for reading from a file
- > for writing to a file

open

```
open(FILE, '<', $file)
```

The third argument is the name of a file to open. It can either be a literal name:

```
open(FILE, '<', 'myfile.txt')
```

or a variable containing a filename:

```
open(FILE, '<', $file)
```

Where can you go for more information on open?

open or die

```
or die "can't open $file: $!\n";
```

open or die is a Perl idiom. die is a function that exits the program immediately and prints the specified string to STDERR.

Why or? What is being tested for truth?

open — \$!

```
or die "can't open $file: $!\n";
```

`$!` is a special Perl variable that contains error messages from the system. If there was a problem with opening your file, there will be an error message in `$!`, and we can include it in *our* error string.

Let's try it.

open for writing

Open also can be used to open files for *writing* by using '>' as the second argument to open.

```
my $out = shift;  
open(FILE, '>', $out) or die "can't open $out: $!\n";
```

Now specify that filehandle when you say or print:

```
say FILE "I'm writing to a file!";
```

Be careful! If you open an existing file for writing, you will erase everything inside that file!

open

You can open more than one file in a script — just give them different filehandles.

```
my $in = shift;  
my $out = shift;  
open( IN, '<', $in ) or die "can't open $in: $!\n";  
open( OUT, '>', $out ) or die "can't open $out: $!\n";
```

open

To read from a filehandle line by line, you put the name of the filehandle inside `<>`, like this:

```
my $in = shift;
open( IN, '<', $in ) or die "can't open $in: $!\n";

while (my $line = <IN>) {
    chomp $line;
    print "This line is from the file $in: $line\n";
}
```

a quick word on system

We saw yesterday that there were two ways of executing a command line from within Perl:

```
# with system  
system("sort $file");
```

```
# or with backticks  
`sort $file`;
```


a quick word on system

With backticks, you can capture the output from the command into a variable:

```
open(OUT, '>', 'sorted.txt') or die "error:$!";  
my $sorted_output = `sort $file`;  
print OUT "sorted output:\n", $sorted_output;
```

Arrays

Sofia Robb

1

What is an Array?

- An array is a named list.
- What is a list?
 - ('cat', 'dog', 'narwhal')
- A named list:
 - @animals = ('cat', 'dog', 'narwhal');

2

Arrays

- Arrays are denoted with '@' symbol

3

Arrays

- Each element of an array is a scalar variable
 - number
 - letter
 - word
 - sentence
 - \$scalar_variable

4

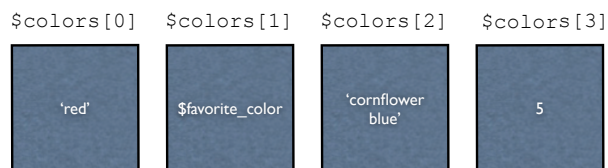
An array of colors.



```
my @colors = ('red', $favorite_color,  
'cornflower blue', 5);
```

5

Accessing each element of an array by its index.



```
my @colors = ('red', $favorite_color,  
'cornflower blue', 5);
```

```
my $first  = $color[0];  
my $second = $color[1];  
my $third  = $color[2];  
my $last   = $color[-1];
```

6

Each element of the array is a scalar variable therefore we use the '\$' when we refer to an individual element.

```
my $first = $color[0];  
my $second = $color[1];  
my $third = $color[2];  
my $last = $color[-1];
```

7

A common MISTAKE is to try to access an element in array context (meaning using the '@').

```
my @colors = ('red', $favorite_color,  
             'cornflower blue', 5);
```

This is wrong:

```
my $first = @color[0];
```

This is correct:

```
my $first = $color[0];
```

8

Changing values at a specific array index.



```
$color[1] = 'black';
```



The value at position `$color[1]` is changed from `$favorite_color` to 'black'.

9

Calculate length of an array with `scalar`

```
my @colors = ('red', $favorite_color,  
'cornflower blue', 5);
```

```
my $length = scalar @colors;  
print "$length\n";  
4
```

```
my $length = @colors;  
print "$length\n";  
4
```

10

Quick print of an array

```
my @colors = ('red', $favorite_color,  
             'cornflower blue', 5);
```

```
print "@colors";  
red purple cornflower blue 5
```

When double quotes are used around “@array” in a print statement, the array elements are printed with a single space separating each element.

11

Converting an array to a string using `join()`

```
my $new_string = join(string , @array);
```

```
my @colors = ('red', $favorite_color,  
             'cornflower blue', 5);
```

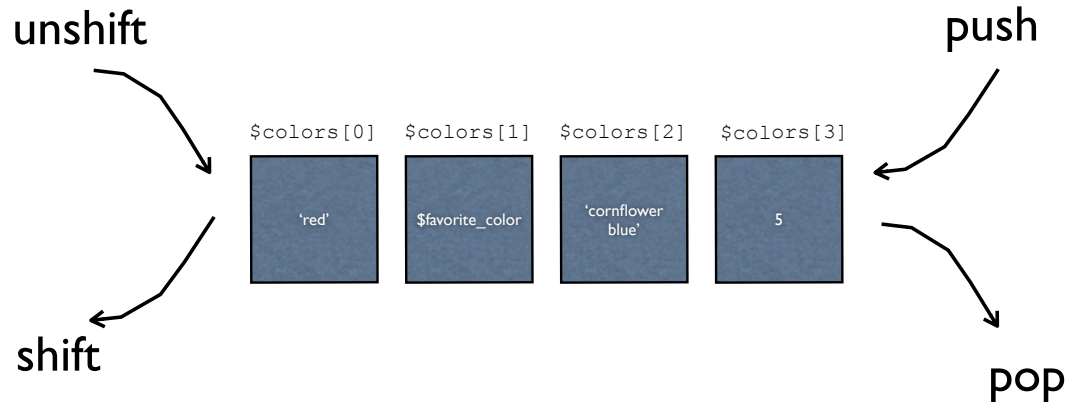
```
my $new_string = join ('--' , @colors);  
print "$new_string\n";  
red--purple--cornflower blue--5
```

The `join()` function concatenates each element of the array with the provided 1st argument ‘--’ into a string.

12

Arrays are Dynamic

Arrays can grow and shrink



13

Add elements to the end with `push()`;



```
push (@array, list of values);
```

```
#add one element to the end  
push (@colors, 'black');
```

```
print join ('--', @colors) , "\n";  
red--purple--cornflower blue--5--black
```

14

Add elements to the end with push();

push



```
push (@array, list of values);
```

```
#add two elements to the end  
push (@colors, 'black' , 'blue');
```

```
print join('--',@colors), "\n";  
red--purple--cornflower blue--black--blue
```

15

Add elements to the end with push();

push



```
push (@array, list of values);
```

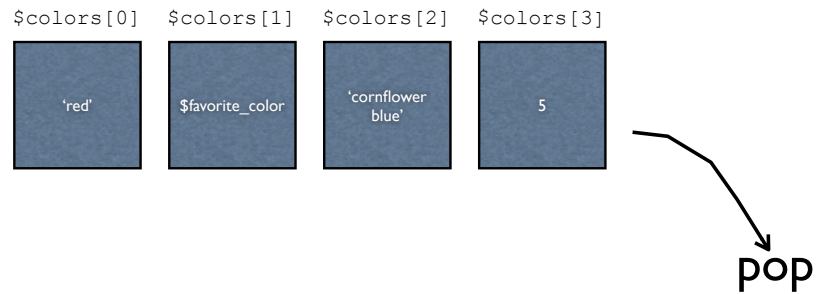
```
#add an array of elements  
my @more_colors =  
( 'yellow', 'pink', 'white', 'orange' );
```

```
push (@colors, @more_colors);
```

```
print join('--',@colors) , "\n";  
red--purple--cornflower blue--5--yellow--pink--white--orange
```

16

Remove an element from the end with pop();



```
my $last_element = pop @colors;

print "$last_element\n";
5
print join ('--', @colors) , "\n";
red--purple--cornflower blue
```

17

Remove an element from the beginning with shift();



```
my $first_element = shift(@colors);

print "$first_element\n";
red

print join ('--', @colors) , "\n";
purple--cornflower blue--5
```

18

Add elements to the beginning with unshift();



```
unshift (@array, list of values);
```

```
#add one element to the beginning  
unshift (@colors, 'black');
```

```
print join ('--', @colors) , "\n";  
black--red--purple--cornflower blue--5
```

19

Add elements to the beginning with unshift();



```
unshift (@array, list of values);
```

```
#add one element to the beginning  
unshift (@colors, 'black');
```

```
print join ('--', @colors) , "\n";  
black--red--purple--cornflower blue--5
```

20

Add elements to the beginning with unshift();



```
unshift (@array, list of values);
```

```
#add two elements to the beginning  
unshift (@colors, 'black' , 'blue');
```

```
print join('--',@colors), "\n";  
black--blue--red--purple--cornflower blue
```

21

Add elements to the beginning with unshift();



```
unshift (@array, list of values);
```

```
#add an array of elements to the beginning  
my @more_colors =  
( 'yellow', 'pink', 'white', 'orange' );
```

```
unshift (@colors, @more_colors);
```

```
print join('--',@colors) , "\n";  
yellow--pink--white--orange--red--purple--cornflower blue--5
```

22

Dynamic Arrays

Function	Meaning
<code>push(@array, a list of values)</code>	add value(s) to the end of the list
<code>\$popped_value = pop(@array)</code>	remove a value from the end of the list
<code>\$shifted_value = shift(@array)</code>	remove a value from the front of the list
<code>unshift(@array, a list of values)</code>	add value(s) to the front of the list
<code>splice(...)</code>	everything above and more!

23

Converting a string into an array

```
my @array = split(pattern , string);
```

```
my $string = "I do not like green eggs and ham";  
my @words = split(' ', $string);
```

```
print join('--', @words), "\n";  
I--do--not--like--green--eggs--and--ham
```

`split()` is splitting the string on ' ' (a single white space) into individual array elements.

24

Sorting the elements of an array

```
my @words = qw(I do not like green eggs and ham);  
  
my @sorted_words = sort @words;  
  
print join(@sorted_words), "\n";  
I--and--do--eggs--green--ham--like--not  
##ascii sort order. 0-9 then A-Z then a-z
```

The array sorts in ascii order not ABC order.

25

Sorting using the `cmp` operator

```
my @words = qw(I do not like green eggs and ham);  
  
##sort {$a cmp $b} is default sort behavior  
my @sorted_words = sort {$a cmp $b} @words;  
  
print join(@sorted_words), "\n";  
I--and--do--eggs--green--ham--like--not
```

26

The comparison operator and strings

```
my $x = 'sid';  
my $y = 'nancy';  
my $result = $x cmp $y;
```

\$result is:

- 1 if the ascii value of the left side is less than the right side
- 0 if the ascii value of the left side equals the right side
- +1 if the ascii value of the left side is greater than the right side

27

Reverse sorting of arrays using the `cmp` operator

```
my @words = qw(I do not like green eggs and ham);  
my @sorted_words = sort {$b cmp $a} @words;  
print join('--', @sorted_words), "\n";  
not--like--ham--green--eggs--do--and--I
```

28

The comparison operator for numbers

```
my $x = 2;  
my $y = 3.14;  
my $result = $x <=> $y;
```

\$result is:

- 1 if the value of the left side is less than the right side
- 0 if the value of the left side equals the right side
- +1 if the value of the left side is greater than the right side

29

Numeric sorting of arrays using the <=> operator

```
my @numbers = (15,2,10,20,11,1);  
  
## default sorting is ascii  
my @sorted_numbers = sort @numbers;  
print "@sorted_numbers\n";  
1 10 11 15 2 20  
  
@sorted_numbers = sort {$a <=> $b}@numbers;  
print "@sorted_numbers\n";  
1 2 10 11 15 20
```

With the <=> the numbers of the array sort by numeric value and not ascii value.

30

Using the `map` function with arrays

```
my @words = qw(I do not like green eggs and ham);  
  
my @ABC_words = map { uc }@words;  
print join('--',@ABC_words),"\n";  
I--DO--NOT--LIKE--GREEN--EGGS--AND--HAM  
  
my @sorted_words = sort (@ABC_words);  
print join('--',@sorted_words),"\n";  
AND--DO--EGGS--GREEN--HAM--I--LIKE--NOT
```

After converting to uppercase the array sorts in ABC order.

31

Accessing Each Element of an Array

- Loops
 - foreach
 - for
 - while

32

foreach loop

```
## iterate thru @array
foreach my $one_element (@array) {
    ##do something to each $one_element
}
```

33

Iterating through an array with a foreach loop

```
my @words = qw(I do not like green eggs and ham);

foreach my $word (@words) {
    print "$word\n";
}
I
do
not
like
green
eggs
and
ham
```

34

Sorting an array using `cmp` and iterating through each element

```
my @words = qw(I do not like green eggs and ham);

foreach my $word (sort {uc($a)cmp uc($b)}@words){
    print "$word\n";
}
and
do
eggs
green
ham
I
like
not
```

35

for loop iterations

```
for(initialization; test; increment){
    statements;
}
```

```
for (my $i=0; $i<5 ; $i++){
    print "$i\n";
}
0
1
2
3
4
```

36

for loop iterations

```
for (my $i=0; $i<5 ; $i++){  
  print "$i\n";  
}
```

0
1
2
3
4

\$i	\$i<5	print "\$i\n";	\$i++
0	yes	0	1
1	yes	1	2
2	yes	2	3
3	yes	3	4
4	yes	4	5
5	no		

37

while loop iterations

```
while (condition) {  
  statements;  
}
```

```
my $i = 0;  
while ($i<5){  
  print "$i\n";  
  $i++;  
}
```

0
1
2
3
4

38

while loop iterations

```
my $i = 0;
while ($i<5){
    print "$i\n";
    $i++;
}
```

	\$i	\$i<5	print "\$i\n";	\$i++
0	0	yes	0	1
1	1	yes	1	2
2	2	yes	2	3
3	3	yes	3	4
4	4	yes	4	5
	5	no		

39

Loop Control: next

execution of next() will cause the loop to jump to the next iteration.

```
my @words = qw(I do not like green eggs and ham);

foreach my $word (sort {uc($a) cmp uc($b)}@words){
    next if $word eq 'and';
    print "$word\n";
}
do
eggs
green
ham
I
like
not
```

40

Loop Control: last

execution of `last()` will cause the loop to exit the loop.

```
my @words = qw(I do not like green eggs and ham);

foreach my $word (sort {uc($a) cmp uc($b)}@words){
    print "$word\n";
    last if $word eq 'and';
}
and
```

41

Example use of a loop to count the occurrences of a specific strings

```
my @seqs = qw(TTT CGG ATG TAA CCC ACC TGA);

my $count = 0;
foreach my $seq (@seqs){
    if ($seq eq 'TAA' or $seq eq 'TGA' or $seq eq 'TAG'){
        print "*\n";
    }else {
        $count++;
    }
}
print "$count non-stop codons\n";
```

42

@ARGV holds command line arguments

```
./sample_usr_input.pl 5 five
```

```
print "@ARGV\n";  
  
print "\$ARGV[0]: $ARGV[0]\n";  
print "\$ARGV[1]: $ARGV[1]\n";  
  
my $arg1 = shift;  
my $arg2 = shift;  
  
print "arg1: $arg1\n";  
print "arg2: $arg2\n";  
  
print "\$ARGV[0]: $ARGV[0]\n";  
print "\$ARGV[1]: $ARGV[1]\n";
```

@ARGV contains the 2 command line arguments 5 and five

The 2 command line arguments 5 and five are shifted off sequentially

@ARGV now is empty

```
5 five  
$ARGV[0]: 5  
$ARGV[1]: five  
arg1: 5  
arg2: five $ARGV  
[0]:  
$ARGV[1]:
```

Hashes

Sofia Robb

1

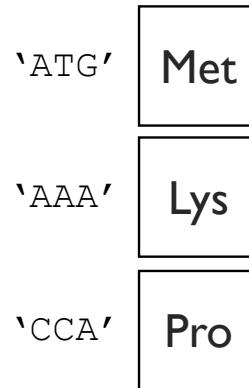
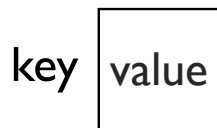
Hashes

- Perl hashes are denoted with a '%' symbol like this
%data
- Each key and each value contains a scalar value for example this could be
 - a number
 - a letter
 - a word
 - a sentence
 - a scalar variable like \$scalar_variable
 - a gene ID
 - a sequence

2

What is a hash?

- A hash is an associative array made up of key/value pairs.
- Like a dictionary
- And unlike an array a hash is unordered.



3

A key is like a descriptive array index.

An array

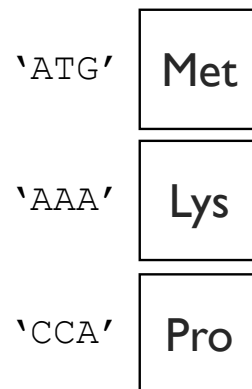


The array index [0] is similar to the key 'ATG'.

The key 'ATG' is used to access the value 'Met', just as [0] is used to access 'red'

But the key/value pairs are not stored in order

A hash



4

Creating a hash

The hash `%genetic_code` is built with key/value pairs

```
my %genetic_code = (  
  'ATG' => 'Met',  
  'AAA' => 'Lys',  
  'CCA' => 'Pro',  
);
```

key	value
'ATG'	Met

5

Accessing a hash value using a key

```
my %genetic_code = (  
  'ATG' => 'Met',  
  'AAA' => 'Lys',  
  'CCA' => 'Pro',  
);  
  
my $aa = %genetic_code{'ATG'};  
print "ATG translates to $aa\n";  
ATG translates to Met
```

Each value of the hash is a scalar therefore we use the '\$' when we refer to an individual value.

Hash keys are surrounded by squiggly brackets `{ }`

6

keys() returns an unordered list of the keys of a hash

```
@array_of_keys = keys (%hash);

my %genetic_code = (
  'ATG' => 'Met',
  'AAA' => 'Lys',
  'CCA' => 'Pro',
);

my @codons = keys (%genetic_code);
print join('--', @codons), "\n";
CCA--AAA--ATG
```

7

Iterating through a hash by looping through an list of hash keys.

```
my %genetic_code = (
  'ATG' => 'Met',
  'AAA' => 'Lys',
  'CCA' => 'Pro',
);

Remember: the key is used to access
the value
$value = $hash{$key}

foreach my $codon (keys %genetic_code) {
  my $aa = $genetic_code{$codon};
  print "$codon translates to $aa\n";
}
CCA translates to Pro
AAA translates to Lys
ATG translates to Met
```

8

Sorting and iterating through the keys of a hash

```
my %genetic_code = (  
  'ATG' => 'Met',  
  'AAA' => 'Lys',  
  'CCA' => 'Pro',  
);
```

Remember: hash keys are unordered so we use `sort` to be sure that the order is always the same.

```
foreach my $codon (sort keys %genetic_code) {  
  my $aa = $genetic_code{$codon};  
  print "$codon translates to $aa\n";  
}  
AAA translates to Lys  
ATG translates to Met  
CCA translates to Pro
```

9

Iterating through a hash and sorting by the values

```
my %genetic_code = (  
  'ATG' => 'Met',  
  'AAA' => 'Lys',  
  'CCA' => 'Pro',  
);
```

```
foreach my $codon (sort {$genetic_code{$a} cmp $genetic_code{$b}}  
keys %genetic_code) {  
  my $aa = $genetic_code{$codon};  
  print "$codon translates to $aa\n";  
}  
AAA translates to Lys  
ATG translates to Met  
CCA translates to Pro
```

Remember: the key is used to access the value
`$value = $hash{$key}`

we can create a custom sort function using `{$a cmp $b}`

10

values() returns an unordered list of values

```
@array_of_values = values(%hash);
```

```
my %genetic_code = (  
  'ATG' => 'Met',  
  'AAA' => 'Lys',  
  'CCA' => 'Pro',  
);
```

You can use `sort values` to be sure that the order of the values is always the same.

```
my @amino_acids = values(%genetic_code);  
print join('--', @amino_acids), "\n";  
Pro--Lys--Met
```

11

Adding additional key/value pairs

```
my %genetic_code = (  
  'ATG' => 'Met',  
  'AAA' => 'Lys',  
  'CCA' => 'Pro',  
);
```

```
$genetic_code{'TGT'} = 'Cys';
```

```
foreach my $codon (keys %genetic_code){  
  print "$codon -- $genetic_code{$codon}\n";  
}  
CCA -- Pro  
AAA -- Lys  
ATG -- Met  
TGT -- Cys
```

12

Deleting key/value pairs

```
my %genetic_code = (  
  'ATG' => 'Met',  
  'AAA' => 'Lys',  
  'CCA' => 'Pro',  
);  
  
delete $genetic_code{'AAA'};  
  
foreach my $codon (keys %genetic_code){  
  print "$codon -- $genetic_code{$codon}\n";  
}  
  
CCA -- Pro  
ATG -- Met
```

13

Use exists() to test if a key exists.

```
my %genetic_code = (  
  'ATG' => 'Met',  
  'AAA' => 'Lys',  
  'CCA' => 'Pro',  
);
```

key exists?	return value
yes	1
no	' ' empty string is false

```
my $codon = 'ATG';  
if (exists $genetic_code{$codon}){  
  print "$codon -- $genetic_code{$codon}\n";  
}else{  
  print "key: $codon does not exist\n";  
}  
  
ATG -- Met  
##when $codon='TTT', code prints "key: TTT does not exist"
```

14

Using hashes for keeping count

```
my $seq = "ATGGGCGTATGCAATT";
my @nucs = split "", $seq;
print "@nucs\n";
#A T G G G C G T A T G C A A T T

my %nt_count;
foreach my $nt (@nucs){
    $nt_count{$nt}++;
}

foreach my $nt (keys %nt_count){
    my $count = $nt_count{$nt};
    print "$nt\t$count\n";
}

A      4
T      5
C      2
G      5
```

A lot happens here: \$hash{key}++;

If a key/value does not exist and perl sees it in a script, it creates the key/value pair and sets the value to undef.

If we add 1 to undef with ++, the resulting value will be 1.

This is equivalent to perl code
\$hash{\$key} = undef;
\$value = \$hash{\$key};
\$hash{\$key} = \$value + 1;
\$hash{\$key} is now 1

If a key exists and its value is a number the value will be incremented by 1.

15

Creating a hash from variable input like data from a file

```
my $file = shift;
open (INFILE, '<', $file)
    or die "can't open file $file $!\n";
my %hash;
while (my $line = <INFILE>){
    chomp $line;
    my ($key, $value) = split /\t/, $line;
    $hash{$key} = $value;
}
foreach my $key (sort keys %hash){
    my $value = $hash{$key};
    print "key:$key value:$value\n";
}
```

16

Regular Expressions

Sofia Robb

1

What is a regular expression?

A regular expression is a string template against which you can match a piece of text.

They are something like shell wildcard expressions, but **much** more powerful.

2

Examples of Regular Expressions

This bit of code loops through @ARGV files or STDIN. Finds all lines containing an EcoRI site, and bumps up a counter:

```
my $sites = 0;
while (my $line = <>) {
    chomp $line;
    if ($line =~ /GAATTC/){
        print "Found an EcoRI site!\n";
        $sites++;
    }
}
print "$sites EcoRI sites total.\n"
```

3

Examples of Regular Expressions

This does the same thing, but counts one type of methylation site (Pu-C-X-G) instead:

```
my $sites = 0;
while (my $line = <>) {
    chomp $line;
    if ($line =~ /[GA]C.?G/) { # more conventional if block
        print "Found a methylation site!\n";
        $sites++;
    }
}
print "$sites methylation sites total.\n"
```

4

Specifying the String to Search

To specify which string variable to search, use the `=~` operator:

```
my $h = "Who's afraid of Virginia Woolf?";  
print "I'm afraid!\n" if $h =~ /Woo?lf/;
```

5

Regular Expression Atoms

A regular expression is normally delimited by two slashes ("`/`"). Everything between the slashes is a pattern to match. A pattern is composed of one or more atoms:

1. Ordinary characters: `a-z`, `A-Z`, `0-9` and some punctuation. These match themselves.
2. The `.` character, which matches everything except the newline.
3. A bracket list of characters, such as `[AaGgCcTtNn]`, `[A-F0-9]`, or `[^A-Z]` (the last means anything BUT A-Z).
4. Certain predefined character sets:
 - `\d`
The digits `[0-9]`
 - `\w`
A word character `[A-Za-z_0-9]`
 - `\s`
White space `[\t\n\r]`
 - `\D`
A non-digit
 - `\W`
A non-word
 - `\S`
Non-whitespace
5. Anchors:
 - `^`
Matches the beginning of the string
 - `$`
Matches the end of the string
 - `\b`
Matches a word boundary (between a `\w` and a `\W`)

6

Regular Expression Atoms

Examples

- `/g..t/` matches "gaat", "goat", and "gotta get a goat" (twice)
- `/g[gatc][gatc]t/` matches "gaat", "gttt", "gatt", and "gotta get an agatt" (once)
- `/\d\d\d-\d\d\d\d/` matches 376-8380, and 5128-8181, but not 055-98-2818.
- `/^\d\d\d-\d\d\d\d/` matches 376-8380 and 376-83801, but not 5128-8181.
- `/^\d\d\d-\d\d\d\d$/` only matches telephone numbers.
- `/\bcat/` matches "cat", "catsup" and "more catsup please" but not "scat".
- `/\bcat\b/` only text containing the word "cat".

7

Quantifiers

By default, an atom matches once. This can be modified by following the atom with a quantifier:

<code>?</code>	atom matches zero or exactly once
<code>*</code>	atom matches zero or more times
<code>+</code>	atom matches one or more times
<code>{3}</code>	atom matches exactly three times
<code>{2,4}</code>	atom matches between two and four times, inclusive
<code>{4,}</code>	atom matches at least four times

Examples:

- `/goa?t/` matches "goat" and "got". Also any text that contains these words.
- `/g.+t/` matches "goat", "goot", and "grant", among others.
- `/g.*t/` matches "gt", "goat", "goot", and "grant", among others.
- `/^\d{3}-\d{4}$/` matches US telephone numbers (no extra text allowed).

8

Alternatives and Grouping

A set of alternative patterns can be specified with the | symbol:

```
/wolf|sheep/;    # matches "wolf" or "sheep"  
  
/big bad (wolf|sheep)/;    # matches "big bad wolf" or "big bad sheep"
```

You can combine parenthesis and quantifiers to quantify entire subpatterns:

```
/Who's afraid of the big (bad )?wolf\?/;  
  
# matches "Who's afraid of the big bad wolf?" and  
#           "Who's afraid of the big wolf?"
```

This also shows how to literally match the special characters -- put a backslash (\) in front of them. There's also an equivalent "not match" operator !~, which reverses the sense of the match:

```
$h = "Who's afraid of Virginia Woolf?";  
print "I'm not afraid!\n" if $h !~ /Woo?lf/;
```

9

Matching with a Variable Pattern

You can use a scalar variable for all or part of a regular expression. For example:

```
$pattern = '/usr/local';  
print "matches" if $file =~ /^$pattern/;
```

Look up o flag or important information about using variables inside patterns.

10

Subpatterns

You can extract and manipulate subpatterns in regular expressions.

To designate a subpattern, surround its part of the pattern with parenthesis (same as with the grouping operator). This example has just one subpattern, (.+) :

```
/Who's afraid of the big bad w(.+)f/
```

11

Matching Subpatterns

Once a subpattern matches, you can refer to it later within the same regular expression. The first subpattern becomes \1, the second \2, the third \3, and so on.

```
while (my $line = <>) {  
    chomp $line;  
    print "I'm scared!\n" if $line =~ /Who's afraid of the big bad w(.)\1f/  
}
```

This loop will print "I'm scared!" for the following matching lines:

- Who's afraid of the big bad woof
- Who's afraid of the big bad weef
- Who's afraid of the big bad waaf

but not

- Who's afraid of the big bad wolf
- Who's afraid of the big bad wife

In a similar vein,

```
/\b(\w+)s love \1 food\b/
```

will match "dogs love dog food", but not "dogs love monkey food".

12

Using Subpatterns Outside the Regular Expression Match

Outside the regular expression match statement, the matched subpatterns (if any) can be found the variables **\$1**, **\$2**, **\$3**, and so forth.

Example. Extract 50 base pairs upstream and 25 base pairs downstream of the TATTAT consensus transcription start site:

```
while (my $line = <>) {
    chomp $line;
    next unless $line =~ /(.{50})TATTAT(.{25})/;
    my $upstream = $1;
    my $downstream = $2;
}
```

13

Extracting Subpatterns Using Arrays

If you assign a regular expression match to an **array**, it will return a list of all the subpatterns that matched. Alternative implementation of previous example:

```
while (my $line = <>) {
    chomp $line;
    my ($upstream, $downstream) = $line =~ /(.{50})TATTAT(.{25})/;
}
```

If the regular expression doesn't match at all, then it returns an empty list. Since an empty list is **FALSE**, you can use it in a logical test:

```
while (my $line = <>) {
    chomp $line;
    next unless my ($upstream, $downstream) = $line =~ /(.{50})TATTAT(.{25})/;
    print "upstream = $upstream\n";
    print "downstream = $downstream\n";
}
```

14

Grouping without Making Subpatterns

Because parentheses are used both for grouping (a|b|c) and for matching subpatterns, you may match subpatterns that don't want to. To avoid this, group with (?:pattern):

```
/big bad (?:wolf|sheep)/;
```

```
# matches "big bad wolf" or "big bad sheep",  
# but doesn't extract a subpattern.
```

15

Subpatterns and Greediness

By default, regular expressions are "greedy". They try to match as much as they can. For example:

```
$h = 'The fox ate my box of doughnuts';  
$h =~ /(f.+x)/;  
$subpattern = $1;
```

Because of the greediness of the match, **\$subpattern** will contain "fox ate my box" rather than just "fox".

To match the minimum number of times, put a **?** after the qualifier, like this:

```
$h = 'The fox ate my box of doughnuts';  
$h =~ /(f.+?x)/;  
$subpattern = $1;
```

Now **\$subpattern** will contain "fox". This is called *lazy matching*. Lazy matching works with any quantifier, such as +?, *?, ?? and {2,50}?

16

String Substitution

String substitution allows you to replace a pattern or character range with another one using the **s///** and **tr///** functions.

The s/// Function

s/// has two parts: the regular expression and the string to replace it with: *s/expression/replacement/*.

```
$h = "Who's afraid of the big bad wolf?";  
$i = "He had a wife.";  
  
$h =~ s/w.+f/goat/; # yields "Who's afraid of the big bad  
goat?"  
$i =~ s/w.+f/goat/; # yields "He had a goate."
```

If you extract pattern matches, you can use them in the replacement part of the substitution:

```
$h = "Who's afraid of the big bad wolf?";  
  
$h =~ s/(\w+) (\w+) wolf/$2 $1 wolf/;  
# yields "Who's afraid of the bad big wolf?"
```

17

Using a Variable in the Substitution Part

Yes you can:

```
$animal = 'hyena';  
$h =~ s/(\w+) (\w+) wolf/$2 $1 $animal/;  
# yields "Who's afraid of the bad big hyena?"
```

18

Translating Character Ranges

The **tr///** function allows you to translate one set of characters into another. Specify the source set in the first part of the function, and the destination set in the second part:

```
$h = "Who's afraid of the big bad wolf?";  
$h =~ tr/ao/AO/; # yields "WhO's AfrAid Of the big bAd  
wOlf?";
```

tr/// returns the number of characters transformed, which is sometimes handy for counting the number of a particular character without actually changing the string.

19

This example counts N's in a series of DNA sequences:

Code:

```
while (my $line = <>) {  
    chomp $line; # assume one sequence per line  
    my $count = $line =~ tr/Nn/Nn/;  
    print "Sequence $line contains $count Ns\n";  
}
```

Input:

```
AGCTGGGAAAGT  
AGCNGNNAAGT  
TAGCNGTTAAAT  
GAATCAGCTGGG  
...
```

Output:

```
(~) 50% count_Ns.pl  
sequence_list.txt  
Sequence 1 contains 0 Ns  
Sequence 2 contains 3 Ns  
Sequence 3 contains 1 Ns  
Sequence 4 contains 0 Ns  
...
```

20

Common Regular Expression Modifiers

Regular expression matches and substitutions have a whole set of options which you can use by appending one or more modifiers to the end of the operation.

i
Case insensitive match.

g
Global match.

21

Case insensitive Matches

```
my $string = 'Big Bad WOLF!';  
print "There's a wolf in the closet!" if $string =~ /wolf/i;  
#case insensitive match
```

22

Global Matches

Adding the `g` modifier to the pattern causes the match to be global. Called in a scalar context (such as an `if` or `while` statement), it will match as many times as it can.

This will match all codons in a DNA sequence, printing them out on separate lines:

Code:

```
my $sequence = 'GTTGCCTGAAATGGCGGAACCTTGAA';
while ( $sequence =~ /(.{3})/g ) {
    print $1, "\n";
}
```

Output:

```
GTT
GCC
TGA
AAT
GGC
GGA
ACC
TTG
```

The `pos()` function retrieves the position where the next attempt begins

```
$position_of_next_attempt = pos($sequence)
```

23

If you perform a global match in a **list** context (e.g. assign its result to an array), then you get a list of all the subpatterns that matched from left to right. This code fragment gets arrays of codons in three reading frames:

```
@frame1 = $sequence =~ /(.{3})/g;
@frame2 = substr($sequence,1) =~ /(.{3})/g;
@frame3 = substr($sequence,2) =~ /(.{3})/g;
```

24

Additional regular expression modifiers

o

Only compile variable patterns once.

m

Treat string as multiple lines. `^` and `$` will match at start and end of internal lines, as well as at beginning and end of whole string. Use `\A` and `\Z` to match beginning and end of whole string when this is turned on.

s

Treat string as a single line. `.` will match any character at all, including newline.

x

Allow extra whitespace and comments in pattern.

Subroutines

Ed Lee

```
#!/usr/bin/perl

use strict;
use warnings;

my $seq1 = "ac ggTtAa";
my $seq2 = "tTcC aaA tgg";

# clean up $seq1
# 1) make it all lower case
$seq1 = lc $seq1;
# 2) remove white space
$seq1 =~ s/\s//g;

# clean up $seq2
# 1) make it all lower case
$seq2 = lc $seq2;
# 2) remove white space
$seq2 =~ s/\s//g;

# print cleaned up sequences
print "seq1: $seq1\n";
print "seq2: $seq2\n";
```

Problems With This Code

- The same cleanup statements are run for \$seq1 and \$seq2
- Duplication of code (BAD!)
- Subroutines to the rescue

Subroutines

- Blocks of code that you can call in different places
- Code resides in one place
 - Only need to write the code once
 - Easier to maintain
- Take arguments and return results
- Make code easier to read
- Like a mini-program within your program

Creating a Subroutine

1. Turn the code of interest into a block

```
{
  # clean up $seq
  # 1) make it all lower case
  $seq = lc $seq;
  # 2) remove white space
  $seq =~ s/\s//g;
}
```

Creating a Subroutine

2. Label the block with

```
sub subroutine_name
```

```
sub cleanup_sequence {
  # clean up $seq
  # 1) make it all lower case
  $seq = lc $seq;
  # 2) remove white space
  $seq =~ s/\s//g;
}
```

Creating a Subroutine

3. Add statements to read the subroutine argument(s) and return the subroutine result(s)

```
sub cleanup_sequence {
    # get the sequence argument to the
    # subroutine - note that just like shift gets
    # an argument for your program, shift gets an
    # argument to your subroutine
    my $seq = shift;

    # clean up $seq
    # 1) make it all lower case
    $seq = lc $seq;
    # 2) remove white space
    $seq =~ s/\s//g;

    # return cleaned up sequence
    return $seq;
}
```


Passing Arguments to a Subroutine

- Arguments are passed in `@_` a special array created by Perl
 - Analogous to `@ARGV` for program arguments
- Can use `shift` to take one argument at a time

```
# take the first argument
my $arg1 = shift;
# take the second argument
my $arg2 = shift;
```

Passing Arguments to a Subroutine

- Can copy the contents of `@_` into a list of named variables

```
my ($arg1, $arg2) = @_;
```

Returning Subroutine Results

- Use `return` operator to return results
 - Usually return at the end of the subroutine but can use it to exit the subroutine earlier
 - Return a single value

```
return $single_value; #scalar
```

- Return a list

```
return ($variable, "string", 3); #list  
return @array_of_values; #array
```

Returning Subroutine Results

- Return an empty list or `undef` depending on context

```
return; #empty list or undef
```

Calling a Subroutine

- Calling our subroutine is just like calling an existing built-in Perl function

```
my $result = my_sub($arg1, $arg2, $arg3, ...);
```

Location of Subroutines

- Usually at the bottom of the script
 - Allows to visually separate main program from the subroutines

```
#!/usr/bin/perl

use strict;
use warnings;

my $seq1 = "ac ggTtAa";
my $seq2 = "tTcC aaA tgg";

# call cleanup_sequence for each sequence
$seq1 = cleanup_sequence($seq1);
$seq2 = cleanup_sequence($seq2);

# print cleaned up sequences
print "seq1: $seq1\n";
print "seq2: $seq2\n";

sub cleanup_sequence {
    # get the sequence argument
    my $seq = shift;
    # cleanup $seq
    # 1) make it all lower case
    $seq = lc $seq;
    # 2) remove white space
    $seq =~ s/\s//g;
    # return cleaned up sequence
    return $seq;
}
```

Scope

```
#!/usr/bin/perl

use strict;
use warnings;

my $x = 100;
my $y = 20;

if ($x > $y) {
    my $z = 10;
    $x = 30;
    print "x (inside if block): $x\n";
    print "y (inside if block): $y\n";
    print "z (inside if block): $z\n";
}

print "x (outside if block): $x\n";
print "y (outside if block): $y\n";
print "z (outside if block): $z\n";
```

Global symbol "\$z" requires explicit package name at ./scope.pl line 19.
Execution of ./scope.pl aborted due to compilation errors.

Blocks

- That's because \$z was declared *inside* the if block, so it's only accessible inside that block
- Any time we see { }, we're creating a block
- Blocks are like boxes that have one way mirrors – you can see outside the box from inside, but not inside the box from the outside
- To fix that error, we need to declare \$z outside the if block

Blocks

- Variables declared inside of a block *only* exist inside the block – once the block is finished, they will be destroyed

```
#!/usr/bin/perl

use strict;
use warnings;

my $x = 100;
my $y = 20;
my $z = 5;

if ($x > $y) {
    my $z = 10;
    $x = 30;
    print "x (inside if block): $x\n";
    print "y (inside if block): $y\n";
    print "z (inside if block): $z\n";
}

print "x (outside if block): $x\n";
print "y (outside if block): $y\n";
print "z (outside if block): $z\n";
```

Output:

```
x (inside if block): 30
y (inside if block): 20
z (inside if block): 10
x (outside if block): 30
y (outside if block): 20
z (outside if block): 5
```

Scope

- Does the program give the expected behavior?
- By declaring “my \$z = 10;” inside the if block, we’re creating a *new* variable called \$z only accessible within the block
- This new variable will *not* modify the outside variable!
- Note that we can create a new \$z variable inside the block with no problems – if we do it outside, we’ll get a warning

Scope

- If we remove “my” from that line, the modification to \$z will show outside the block

```
#!/usr/bin/perl

use strict;
use warnings;

my $x = 100;
my $y = 20;
my $z = 5;

if ($x > $y) {
    $z = 10;
    $x = 30;
    print "x (inside if block): $x\n";
    print "y (inside if block): $y\n";
    print "z (inside if block): $z\n";
}

print "x (outside if block): $x\n";
print "y (outside if block): $y\n";
print "z (outside if block): $z\n";
```

Output:

```
x (inside if block): 30
y (inside if block): 20
z (inside if block): 10
x (outside if block): 30
y (outside if block): 20
z (outside if block): 10
```


Using Modules

Programming for Biology 2011

Why use modules?

- Sometimes you may want to use the same functions over and over again in different programs
- Bad way: Copy and paste
- Good way: Make a module
- There are also many many modules that other people have written that you can use!
- To use modules they must be properly installed and called with the “use” command

File::Basename

basename

- Input = long UNIX path name
 - i.e. '/bush_home/bush1/lstein/dna.fa'
- Output = file name
 - i.e. 'dna.fa'

• *dirname*

- Input = long UNIX path name
 - i.e. '/bush_home/bush1/lstein/dna.fa'
- Output = directory
 - '/bush_home/bush1/lstein/'

File::Basename

```
#!/usr/bin/perl  
# file: basename.pl
```



```
use strict;  
use File::Basename;
```

```
my $path = '/bush_home/bush1/lstein/dna.fa';  
my $base = basename($path);  
my $dir  = dirname($path);
```

```
print "The base is $base and the directory is $dir.\n";
```

Output:

```
The base is dna.fa and the directory is /bush_home/bush1/  
lstein.
```

Common error: Undefined subroutine &main::basename called at basename.pl
line 8.

Env

- This standard module imports a set of scalar variables that describe your environment

- \$HOME
- \$PATH
- \$USER

Env

```
#!/usr/bin/perl
# file env.pl

use strict;
use Env;

print "My home is $HOME\n";
print "My path is $PATH\n";
print "My username is $USER\n";
```

Output:

```
My home is /bush_home/bush1/lstein
My path is /bush_home/bush1/lstein/pfb2011
My username is lstein
```

Installed modules

- perldoc perlmodlib
- modules installed with basic perl installation
- <http://perldoc.perl.org/perlmodlib.html>
- perldoc perllocal
- Tells you modules that are installed on your machine

Installing modules manually

```
% tar zxvf bioperl-1.6.1.tar.gz
bioperl-1.6.1/
bioperl-1.6.1/Bio/
...

% perl Makefile.PL
Generated sub tests. go make show_tests to see available subtests
...
Writing Makefile for Bio

% make
cp Bio/Tools/Genscan.pm blib/lib/Bio/Tools/Genscan.pm
...
Manifying blib/man3/Bio::Location::CoordinatePolicyI.3
Manifying blib/man3/Bio::SeqFeature::Similarity.3

% make test
PERL_DL_NONLAZY=1 /net/bin/perl -Iblib/arch -Iblib/lib
-I/net/lib/perl5/5.6.1/i686-linux -I/net/lib/perl5/5.6.1 -e 'use
Test::Harness qw(&runtests $verbose); $verbose=0; runtests @ARGV;' t/*.t
t/AChange.....ok
...
All tests successful, 95 subtests skipped.
Files=60, Tests=1011, 35 wallclock secs (25.47 cusr + 1.60 csys = 27.07 CPU)

% make install
Installing /net/lib/perl5/site_perl/5.6.1/bioback.pod
Installing /net/lib/perl5/site_perl/5.6.1/biostart.pod
...
```


Installing Modules Using the CPAN Shell

Perl has a CPAN module installer built into it. You run it like this:

```
% cpan
```

```
cpan shell -- CPAN exploration and modules installation (v1.59_54)  
ReadLine support enabled
```

```
cpan>
```

From this shell, there are commands for searching for modules, downloading them, and installing them.

[The first time you run the CPAN shell, it will ask you a lot of configuration questions. Generally, you can just hit return to accept the defaults. The only trick comes when it asks you to select CPAN mirrors to download from. Choose any ones that are in your general area on the Internet and it will work fine.]

To search for a module:

```
cpan> i /Wrap/
```

```
Going to read /bush_home/bush1/lstein/.cpan/sources/authors/01mailrc.txt.gz
```

```
CPAN: Compress::Zlib loaded ok
```

```
Going to read /bush_home/bush1/lstein/.cpan/sources/modules/02packages.details.txt.gz
```

```
Database was generated on Tue, 16 Oct 2001 22:32:59 GMT
```

```
...
```

```
Module          Text::Wrap      (M/MU/MUIR/modules/Text-Tabs+Wrap-2001.0929.tar.gz)
```

```
...
```

```
41 items found
```

```
cpan> install Text::Wrap
```

```
Running install for module Text::Wrap
```

```
quit
```

Where are module installed?

Module files end with the extension `.pm`. If the module name is a simple one, like **Env**, then Perl will look for a file named **Env.pm**. If the module name is separated by `::` sections, Perl will treat the `::` characters like directories. So it will look for the module **File::Basename** in the file **File/Basename.pm**

Perl searches for module files in a set of directories specified by the Perl library path. This is set when Perl is first installed. You can find out what directories Perl will search for modules in by issuing **perl -V** from the command line:

```
% perl -V
Summary of my perl5 (revision 5.0 version 6 subversion 1) configuration:
 Platform:
   osname=linux, osvers=2.4.2-2smp, archname=i686-linux
 ...
 Compiled at Oct 11 2001 11:08:37
 @INC:
   /usr/lib/perl5/5.6.1/i686-linux
   /usr/lib/perl5/5.6.1
   ...
```

You can modify this path to search in other locations by placing the **use lib** command somewhere at the top of your script:

```
#!/usr/bin/perl

use lib '/home/lstein/lib';
use MyModule;
...
```

This tells Perl to look in **/home/lstein/lib** for the module **MyModule** before it looks in the usual places. Now you can install module files in this directory and Perl will find them.

Sometimes you really need to know where on your system a module is installed. `Perldoc` to the rescue again -- use the `-l` command-line option:

```
% perldoc -l File::Basename
/System/Library/Perl/5.8.8/File/Basename.pm
```

Making modules

Programming for Biology 2011

What is a module?



```
52 sub _get_description {
53     my $service = shift;
54     my $protocol = shift;
55
56     my $description;
57     if ($protocol) {
58         $description = " ($protocol";
59         if ($service) {
60             $description .= ", $service)";
61         }
62     } else {
63         $description .= ")";
64     }
65 }
66 return ($description);
67 }
```

```
4  ### The Central Dogma ###
5  #####
6
7  ### Genome
8  $DNA_seq = "ATGACCCTACTAGATCATCTATGATAGCTCAT";
9
10 ### Transcription
11 $RNA_seq = $DNA_seq;
12 $RNA_seq =~ s/T/U/gi;
13 print "$RNA_seq\n";
14
15 ### Translation
16 $position = 0;
17 while (substr $RNA_seq,$position,3) {
18     $codon = substr $RNA_seq,$position,3;
19     print translate_codon($codon);
20     $position = $position + 3;
21 }
22 sub translate_codon {
23     if ($_[0] =~ /GC./i) {return Ala;}
24     if ($_[0] =~ /UGC|UGU/i) {return Cys;}
```

continue

Module

```
package MySequence;  
#file: MySequence.pm  
  
use strict;  
our $EcoRI = 'ggatcc';
```

A package (or namespace) is an abstract **container** or **environment** created to hold a logical grouping of unique symbols (i.e., subroutines).

```
sub reverseq {  
    my $sequence = shift;  
    $sequence = reverse $sequence;  
    $sequence =~ tr/gatcGATC/ctagCTAG/;  
    return $sequence;  
}  
  
sub seqlen {  
    my $sequence = shift;  
    $sequence =~ s/[^gatcnGATCN]//g;  
    return length $sequence;  
}  
  
1;
```

A Perl module must end with a true value.

Script

```
#!/usr/bin/perl
#file: sequence.pl

use strict;
use warnings;
use MySequence;

my $sequence = 'gattccggatttccaaagggttcccaatttggg';
my $complement = MySequence::reverseq($sequence);

print "original = $sequence\n";
print "complement = $complement\n";
```

Must explicitly *qualify* each MySequence function by using the notation

MySequence::function_name

Module using Exporter

```
package MySequence;
#file: MySequence.pm

use strict;
use base 'Exporter';

our @EXPORT = qw(reverseseq seqlen);
our @EXPORT_OK = qw($EcoRI);
our $EcoRI = 'ggatcc';

sub reverseseq {
    my $sequence = shift;
    $sequence = reverse $sequence;
    $sequence =~ tr/gatcGATC/ctagCTAG/;
    return $sequence;
}

sub seqlen {
    my $sequence = shift;
    $sequence =~ s/[^gatcnGATCN]//g;
    return length $sequence;
}

1;
```

*

Script using Exporter

```
#!/usr/bin/perl
#file: sequence.pl

use strict;

use warnings;
use MySequence;

my $sequence = 'gattccggatttccaaagggttcccaatttggg';
my $complement = reverseseq($sequence);

print "original = $sequence\n";
print "complement = $complement\n";
```


Exporter - Implements default import method
for modules

```
use base 'Exporter';  
  
our @EXPORT = qw(reverseq seqlen);  
our @EXPORT_OK = qw($EcoRI);
```

use base 'Exporter' Tells Perl that this module is
a type of "Exporter" module

our @EXPORT = qw(reverseq seqlen) line tells
Perl to export the functions **reverseq** and **seqlen**
automatically.

Also, can export **qw(afunc \$scalar @array %hash);**

our @EXPORT_OK = qw(\$EcoRI) tells Perl that it
is OK for the user to import the \$EcoRI
variable, but not to export it automatically.

Getopt::Long - Extended processing of command line options

Command line operated programs traditionally take their arguments from the command line, for example filenames.

Besides arguments, these programs often take command line *options* as well. Options are not necessary for the program to work, hence the name 'option', but are used to modify its default behaviour.

Example:

```
courses:~ srynearson$ grep -i 'AGCG' > capture.txt
```

```
courses:~ srynearson$ perl GVF_Parser.pl -data file.txt
```

Script using Getopt::Long

```
#!/usr/bin/perl -w
```

```
use strict;  
use lib '/Users/srbio/GVF_DB_Variant/lib';  
use Utils;  
use GVF_DB_Connect;  
use IO::File;  
use GVF::DB::Variant;  
use Getopt::Long;
```

```
my $usage = "\n
```

DESCRIPTION:

Parsing script which takes gvf file and stores metadata and gvf line in data structures.

Options allow you to added to specific/all table in GVF_Variant database or none if just working with data structures.

USAGE:

```
./Gvf_Parser.pl -option <GVF_file>
```

OPTIONS(required):

Each option corresponds to a table in the database.

-- all Option will add all areas of GVF file to database.

-- data Will print out the data structures to view.

```
\n";
```

```
my ($all, $data);  
my $input = $ARGV[1] || die $usage;
```

```
GetOptions(  
    'all' => \$all,  
    'data' => \$data,
```

```
) || die $usage;
```

*

References and multidimensional data

Simon Prochnik, Dave Messina, Lincoln Stein, Steve Rozen
PfB 2011

What good are references?

Sometimes you need a more complex data structure than a list.

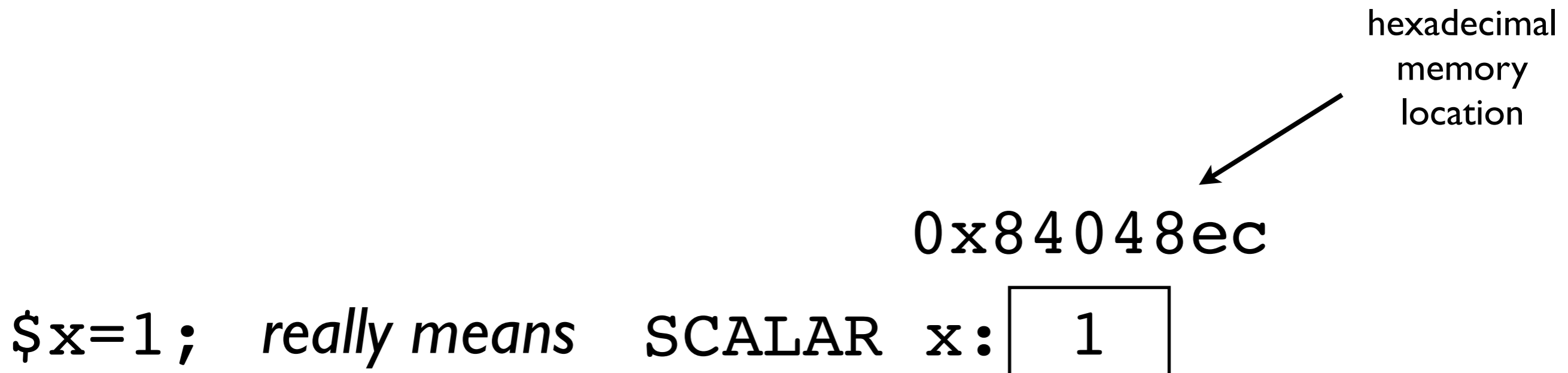
What if you want to keep together several related pieces of information?

Gene	Sequence	Organism
HOXB2	ATCAGCAATATACAATTATAAAGGCCTAAATTTAAAA	mouse
HDAC1	GAGCGGAGCCGCGGGCGGGAGGGGCGGACGGAC	human

What is a reference?

Well first, what is a variable?

A variable is a labeled memory address that holds a value. The location's label is the name of the variable.



What is a list?

```
@y = (1, 'a', 23);
```

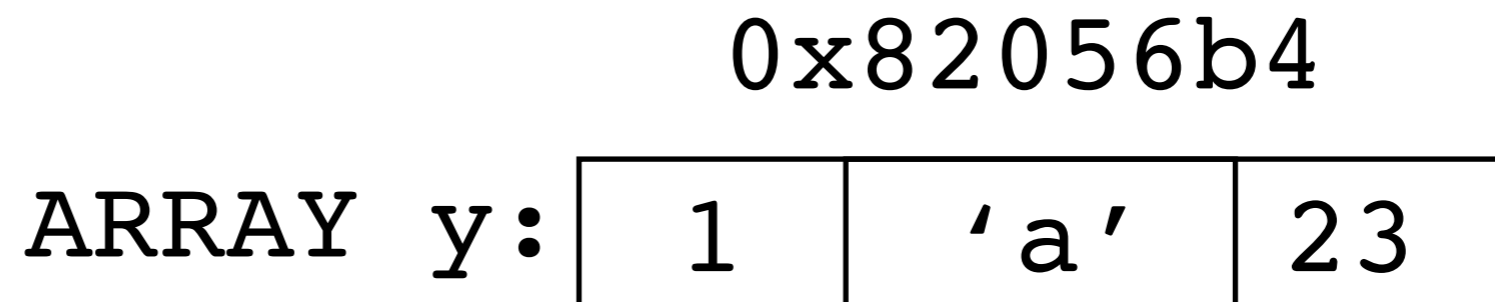
really means

0x82056b4
ARRAY y:

1	'a'	23
---	-----	----

A variable is a labeled memory address.

When we read the contents of the variable, we are reading the contents of the memory address.



So, what is a reference?

A reference is a variable that contains the memory address of some data.

It does not contain the data itself. It contains the memory address where some data is stored.

Making a reference to an array

We can create a reference to named variable `@y` this way:

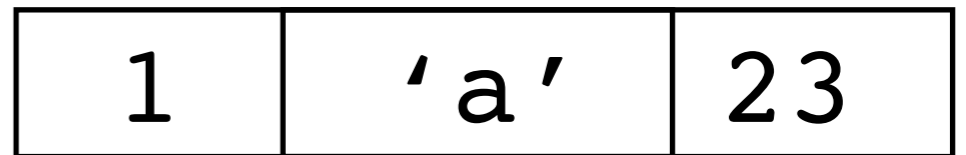
SCALAR

`ref_to_y: 0x82056b4`



ARRAY

`y:`



0x82056b4

Printing a reference

```
SCALAR ref_to_y: 0x82056b4
```

If we try to print out `$ref_to_y`, we see the raw memory address:

```
print $ref_to_y, "\n";  
ARRAY(0x82056b4)
```

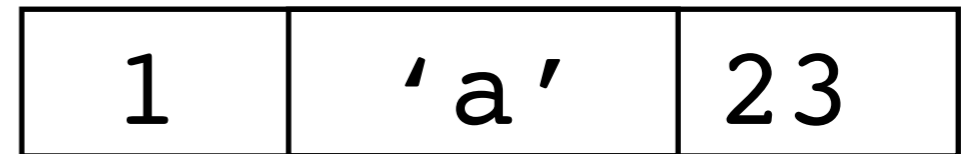
SCALAR

ref_to_y: 0x82056b4



ARRAY

y:



To see the **contents** of what \$ref_to_y **points to**, we have to **dereference** it:

```
print join ' ', @{$ref_to_y};  
1 a 23
```

You can create references to scalars, arrays and hashes

```
# create some references
$scalar_ref = \ $count;
$array_ref  = \@array;
$hash_ref   = \%hash;
```

To dereference a reference, place the appropriate symbol (\$, @, %) in front of the reference:

```
# dereference your references:
$count_copy = ${$scalar_ref};
@array_copy = @{$array_ref};
%hash_copy  = %{$hash_ref};
```

References are pointers

A reference is a pointer to the data. It isn't a copy of the data.

When you make a reference to a variable, you have only created another way to get at the data.

There is still only one copy of the data.

```
@y                = (1, 'a', 23);  
$ref_to_y         = \@y;  
print join ' ', @{$ref_to_y};  
1 a 23
```

```
push @{$ref_to_y}, 'new1', 'new2';  
  
print join ' ', @$ref_to_y;  
1 a 23 new1 new2
```

This is in contrast to doing a direct copy from one variable to another, which creates a new data structure in a new memory location.

```
@y    = (1, 'a', 23);  
@z    = @y;  
push @y, 'new1', 'new2';
```

```
print join ' ', @y;  
1 a 23 new1 new2
```

```
print join ' ', @z;  
1 a 23
```

If you have a reference to an array or a hash, you can access any element.

```
$value = $y[2];
```

directly access the 3rd element in @y

```
$value = ${$ref_to_y}[2];
```

dereference the reference, then access the 3rd element in @y

```
`${$ref_to_y}[2] = 'new';  
print join ' ', @y;  
1 a new
```

change the value of the 3rd element in @y


```
%z = ( 'dog' => 'animal',  
      'potato' => 'vegetable',  
      'quartz' => 'mineral',  
      'tomato' => 'vegetable' );
```

```
$ref_to_z = \%z;
```

```
$value = $z{ 'dog' }; ←
```

directly access the value associated with the key 'dog' in the hash %z

```
$value = ${$ref_to_z}{ 'dog' };
```

dereference the reference, then get the value associated with the key 'dog' in the hash %z

```
${$ref_to_z}{ 'tomato' } = 'fruit';
```

```
print join ' ', values %z;
```

```
animal vegetable mineral fruit
```

change the value associated with the key 'tomato' in the hash %z

Anonymous Hashes and Arrays

You will not usually make references to existing variables. Instead you will create anonymous hashes and arrays. These have a memory location, but no symbol or name, i.e. you can't write `@my_data`. The reference is the only way to address them.

To create an anonymous array use the form:

```
$ref_to_array = ['item1', 'item2' ...]
```

To create an anonymous hash, use the form:

```
$ref_to_hash =  
{key1=>'value1', key2=>'value2', ...}
```

Remember
[] goes with arrays
\$a[0] etc and
{ } goes with
hashes \$hash
{ \$key } etc

```
$y_gene_families = [ 'DAZ', 'TSPY', 'RBMV', 'CDY1',  
'CDY2' ];
```

```
$y_gene_family_counts = { 'DAZ' => 4,  
                           'TSPY' => 20,  
                           'RBMV' => 10,  
                           'CDY2' => 2 };
```

```
$third_item_of_array = $y_gene_families->[2];  
$daz_count            = $y_gene_family_counts->{DAZ};
```

`$y_gene_families` gets (i.e. is assigned) a reference to an array, and `$y_gene_family_counts` gets a reference to a hash.

Multidimensional Data: Making a Hash of Hashes

The beauty of anonymous arrays and hashes is that you can nest them:

```
my %y_gene_data = ( 'DAZ' => { 'family_size' => 4,  
    'description' => 'deleted in azoospermia' },  
    'TSPY' => { 'family_size' => 20,  
    'description' => 'testis specific protein Y-  
linked' },  
    'RBMV' => { 'family_size' => 10,  
    'description' => 'RNA-binding motif Y' },  
    'CDY2' => { 'family_size' => 2,  
    'description' => 'chromodomain protein, Y-linked' }  
    );  
  
# what is the size of the RBMY family?  
my $size = $y_gene_data{'RBMV'}{'family_size'};  
  
# what is the description of TSPY?  
my $desc = $y_gene_data{'TSPY'}{'description'};
```

Multidimensional Data: Making an Array of Arrays

```
my @spotarray = (  
    [0.124, 43.2, 0.102, 80.4],  
    [0.113, 60.7, 0.091, 22.6],  
    [0.084, 112.2, 0.144, 35.3]  
);  
my $cell_1_0 = $spotarray[1][0];  
print $cell_1_0;
```

0.113

Examining References

Inside a Perl script, the `ref` function tells you what kind of value a reference points to:

```
print ref($y_gene_data), "\n";  
HASH
```

```
print ref($spotarray), "\n";  
ARRAY
```

```
$x = 1;  
print ref($x), "\n";  
(empty string)
```

Examining complex data structures in the debugger

Inside the Perl debugger, the "x" command will pretty-print the contents of a complex reference:

```
DB<3> x $y_gene_data
0  HASH(0x8404bb0)
   'CDY2' => HASH(0x8404b80)
       'description' => 'chromodomain protein, Y-linked'
       'family_size' => 2
   'DAZ' => HASH(0x84047fc)
       'description' => 'deleted in azoospermia'
       'family_size' => 4
   'RBMV' => HASH(0x8404b50)
       'description' => 'RNA-binding motif Y'
       'family_size' => 10
   'TSPY' => HASH(0x8404b20)
       'description' => 'testis specific protein Y-linked'
       'family_size' => 20
```

Scripting Example: Creating a Hash of Hashes

We are presented with a table of sequences in the following format: the ID of the sequence, followed by a tab, followed by the sequence itself.

```
2L52.1      atgtcaatggtaagaaatgtatcaaatacagagcgaaaaattggaagtaag...
4R79.2      tcaaatacagcaccagctcctttttttatagttcgaattaatgtccaact...
AC3.1       atggctcaaactttactatcacgtcatttccgtgggtgtcaactgttattt...
...
```

For each sequence calculate the length of the sequence and the count for each nucleotide. Store the results into hash of hashes in which the outer hash's key is the ID of the sequence, and the inner hashes' keys are the names and counts of each nucleotide.


```

#!/usr/bin/perl -w

use strict;

# tabulate nucleotide counts, store into %sequences

my %seqs; # initialize hash
while (my $line = <>) {
    chomp $line;
    my ($id,$sequence) = split "\t",$line;
    my @nucleotides = split '', $sequence; # array of base pairs
    foreach my $n (@nucleotides) {
        $seqs{$id}{$n}++; # count nucleotides and keep tally
    }
}

# print table of results
print join("\t",'id','a','c','g','t'),"\n";

foreach my $id (sort keys %seqs) {
    print join("\t",$id,
                $seqs{$id}{a},
                $seqs{$id}{c},
                $seqs{$id}{g},
                $seqs{$id}{t},
                ), "\n";
}

```

The output will look something like this:

id	a	c	g	t
2L52.1	23	4	12	11
4R79.2	15	12	5	18
AC3.1	11	11	8	20
...				

Object Oriented Programming and Perl

Prog for Biol 201 I
Simon Prochnik

Friday, October 21, 2011

1

Why do we teach you about objects?

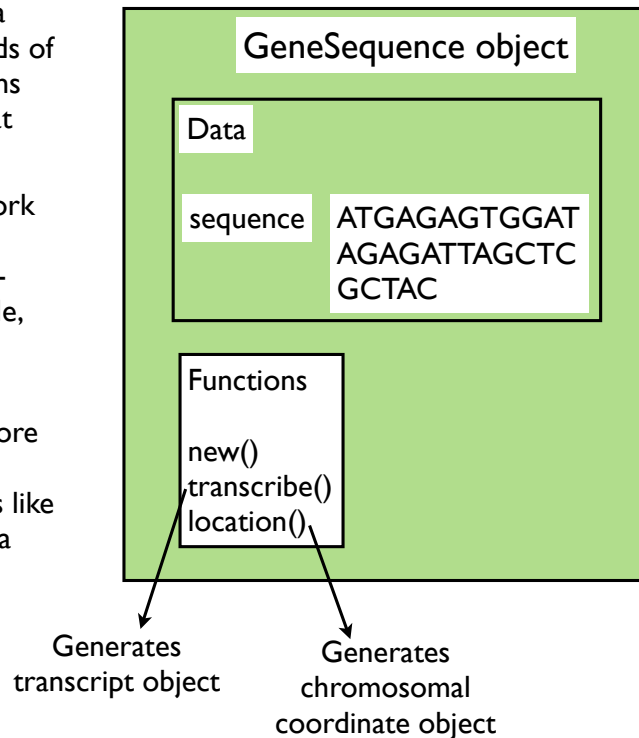
- Objects allow you to use other people's code to do a lot in just a few lines.
- For example, in the lecture on bioperl, you will see how to search GenBank by a sequence Accession, parse the results and reformat the sequence into any format you need in less than a dozen lines of object-oriented perl. Cool!
- Someone else has already written and tested the code, so you don't have to.
- Most people don't ever write an object of their own: only create your own modules and objects if you have to
- check CPAN (www.cpan.org) to see if see if someone has already done it for you. There were 18,534 modules on Oct 14th 2010, this has grown to 100,575 (Oct 20, 2011)! Surely you can find a module to do what you want.

Friday, October 21, 2011

2

What are objects? A programming paradigm

- An object is a special kind of data structure that stores specific kinds of data and provides special functions that can do useful things with that data
- Objects are often designed to work with data and functions that you would find associated with a real-world object or thing, for example, we might design gene sequence objects.
- A gene sequence object might store its chromosomal position and sequence data and have functions like transcribe() and new() to create a new object.



Friday, October 21, 2011

3

Quick example with microarrays

```
#!/usr/bin/perl
#File: 00_script.pl
use strict;
use warnings;
use Microarray;
my $microarray = Microarray->new( gene => 'CDC2',
                                  expression => 45,
                                  tissue => 'liver',
                                  );
my $gene_name = $microarray->gene();
print "Gene for this microarray is $gene_name\n";
my $tissue = $microarray->tissue();
print "The tissue is $tissue\n";
```

Tell perl you want to use Microarray class objects →

Create a new object and load data →

Query the data in the object →

Print the results →

Friday, October 21, 2011

4

Another example with statistics

Make new object
with new()

Add data

Calculate mean

Calculate variance

```
#!/usr/bin/perl
#File: mean_and_variance.pl
use strict;
use warnings;

use Statistics::Descriptive;

$stat = Statistics::Descriptive::Full->new();
$stat->add_data(1,2,3,4);
$mean = $stat->mean();
$var = $stat->variance();
print "mean is $mean\n";
print "variance is $variance\n";
```

Friday, October 21, 2011

5

Object Oriented Programming and Perl

- To understand object-oriented syntax in perl, we need to recap three things: **references, subroutines, packages**.
- These three elements of perl are recycled with slightly different uses to provide object-oriented programming
- The OOP paradigm provides i) a solid framework for sharing code -- reuse
- and ii) a guarantee or contract or specification for how the code will work and how it can be used -- an interface
- and iii) hides the details of implementation so you only have to know how to use the code, not how it works -- saves you time, quick to learn, harder to introduce bugs
- Here we are briefly introducing you to OOP and objects so that you can quickly add code that's already written into your scripts, rather than spend hours re-inventing wheels. Many more people use objects than write them.

Friday, October 21, 2011

6

I: Recap references

example of syntax

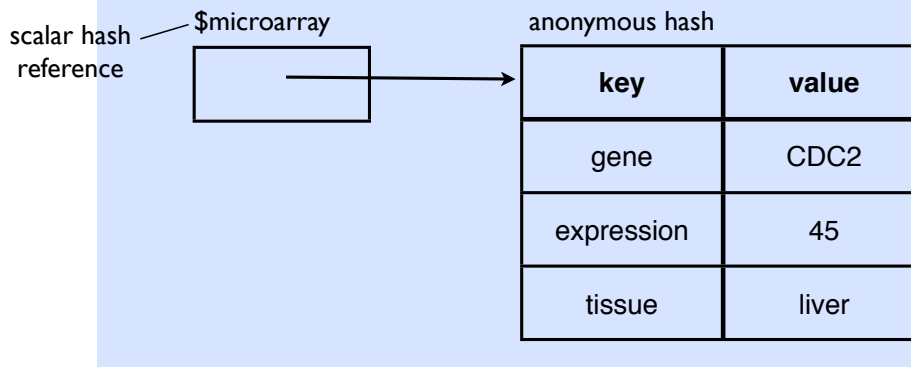
```
$ref_to_hash = {key1=>'value1',key2=>'value2',...}
```

code example

```
my $microarray = {gene => 'CDC2',  
                  expression => 45,  
                  tissue => 'liver',  
                  };
```

We can store any pieces of data we would like to keep together in a hash

Here is the data structure in memory



Friday, October 21, 2011

7

II: recap subroutines

- solve a problem, write code once, and re-use the code
- reusing a single piece of code instead of copying, pasting and modifying reduces the chance you'll make an error and simplifies bug fixing.

```
#!/usr/bin/perl -w  
use strict;  
my $seq;  
while (my $seqline = <>) { # read sequence from standard in  
    my $clean = cleanup_sequence($seqline); # clean it up  
    $seq .= $clean; # add it to full sequence  
}  
sub cleanup_sequence {  
    my ($sequence) = @_; # set $sequence to first argument  
    $sequence = lc $sequence; # translate everything into lower case  
    $sequence =~ s/[\s\d]//g; # remove whitespace and numbers  
    $sequence =~ m/^[gactcn]+$/ or die "Sequence contains invalid  
                                     characters!";  
    return $sequence;  
}
```

Friday, October 21, 2011

8

III: now let's recap packages

- organise code that goes together into reusable modules, packages

```
#!/usr/bin/perl -w                                     read_clean_sequence.pl
#File: read_clean_sequence.pl
use strict;
use Sequence;
my $seq;
while (my $seqline = <>) { # read sequence from standard in
    my $clean    = cleanup_sequence($seqline); # clean it up
    $seq        .= $clean;                   # add it to full sequence
}
```

```
#file: Sequence.pm                                     Sequence.pm
package Sequence;
use strict;
use base Exporter;
our @EXPORT = ('cleanup_sequence');
sub cleanup_sequence {
    my ($sequence) = @_; # set $sequence to first argument
    $sequence = lc $sequence; # translate everything into lower case
    $sequence =~ s/[\s\d]//g; # remove whitespace and numbers
    $sequence =~ m/^[gactcn]+$ / or die "Sequence contains invalid
characters!";
    return $sequence;
}
1;
```

Friday, October 21, 2011

9

Let's recap subroutines: new example with references

```
#!/usr/bin/perl
use strict;
use warnings;
my $microarray = { gene => 'CDC2',
                  expression => 45,
                  tissue => 'liver',
                  };
...
my $gene_name = gene($microarray);
...
sub gene {
    my ($ref) = @_;
    return $ref->{gene};
}
sub tissue {
    my ($ref) = @_;
    return $ref->{tissue};
}
```

Friday, October 21, 2011

10

recap packages

main script
file

```
#!/usr/bin/perl
#File: script.pl
use strict; use warnings;
use Microarray;

my $microarray = {gene => 'CDC2',
                  expression => 45,
                  tissue => 'liver',
}
my $gene_name = gene($microarray);
print "Gene for this microarray is
$gene\n";
```

perl module file

```
Microarray.pm

#File: Microarray.pm
package Microarray;
use strict;
use base Exporter;

our @EXPORT = ('gene', 'tissue');

sub gene {
    my ($ref) = @_;
    return $ref->{gene};
}

sub tissue {
    my ($ref) = @_;
    return $ref ->{tissue};
}

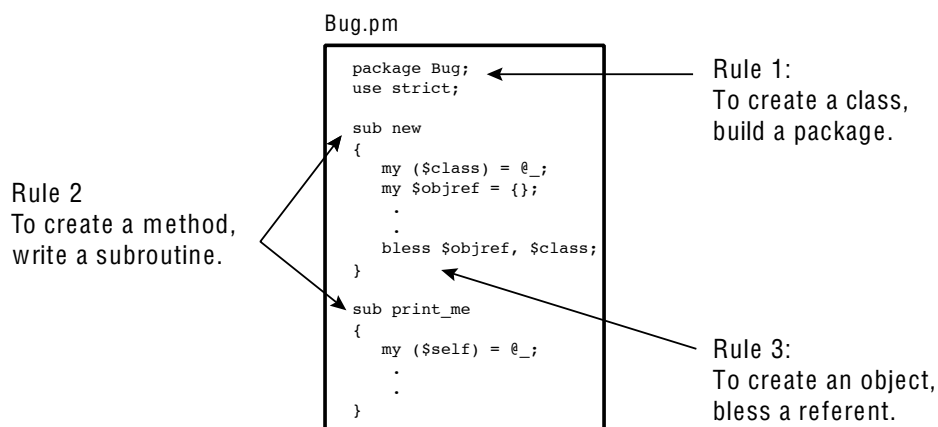
1;
```

Friday, October 21, 2011

11

Three Little Rules

- Rule 1: To create a class, build a package
- Rule 2: To create a method, write a subroutine
- Rule 3: To create an object, bless a reference



Friday, October 21, 2011

12

Rule 1: To create a class, build a package

- all the code that goes with an object (methods, special variables) goes inside a special package
 - perl packages are just files whose names end with '.pm' e.g. `Microarray.pm`
 - package filenames should start with a capital letter
 - the name of the perl package tells us the class of the object. This is really the type or kind of object we are dealing with.
- `Micorarray.pm` is a package, so it will be easy to convert into object-oriented code

Friday, October 21, 2011

13

Rule 2: To create a method, write a subroutine

- we already have `gene()` in `Microarray.pm`
- this can be turned into a method
- we need one extra subroutine to create new objects
- the creator method is called `new()` and has one piece of magic...

Friday, October 21, 2011

14

Rule 3: To create an object, bless a reference

- The `new()` subroutine uses the `bless` function to create an object
- full details coming up... but here's the skeleton of a `new()` method

```
sub new {  
    ...  
    my $self = {};  
    bless $self, $class;  
    ...  
}
```

create a reference, a hashref {} is the most common seen in perl

bless a reference into a class

Friday, October 21, 2011

15

Let's recap packages

```
#!/usr/bin/perl -w  
#File: script.pl  
use strict;  
use Microarray;  
  
my $microarray = { gene => 'CDC2',  
                  expression => 45,  
                  tissue => 'liver',  
                  };  
my $gene_name = gene($microarray);  
print "Gene for this microarray is $gene\n";
```

```
#File: Microarray.pm  
package Microarray;  
use strict;  
use base Exporter;  
  
our @EXPORT = ('gene', 'tissue');  
  
sub gene {  
    my $ref = shift;  
    return $ref->{gene};  
}  
sub tissue {  
    my $ref = shift;  
    return $ref->{tissue};  
}  
1;
```

Friday, October 21, 2011

16

Transforming a package into an object-oriented module or class

procedural perl package
(what you saw yesterday)

...transforming the package into a class...

```
#File: Microarray.pm
package Microarray;
use strict;
use base Exporter;

our @EXPORT = ('gene', 'tissue');

sub gene {
    my ($ref) = @_;
    return $ref->{gene};
}
sub tissue {
    my ($ref) = @_;
    return $ref ->{tissue};
}
1;
```



```
#File: Microarray.pm
package Microarray;
use strict;

sub gene {
    my $self = shift; # same as my ($self) = @_;
    return $self->{gene};
}
sub tissue {
    my $self = shift;
    return $self ->{tissue};
}
1;
```

Friday, October 21, 2011

17

The new() method is a subroutine that creates a new object

```
sub new {
    my $class = shift;
    my %args = @_;
    my $self = {};
    foreach my $key (keys %args) {
        $self -> {$key} =
            $args{$key};
    }
    # the magic happens here
    bless $self, $class;
    return $self;
}
```

the first argument is always the class of the object you are making. **perl gives you this as the first argument automatically**

a hash reference is the data structure you build an object from in perl

here we initialize variables in the object (in case there are any)

bless makes the object \$self (which is a hash reference) become a member of the class \$class

Friday, October 21, 2011

18

bless associates an object with its class

Make an anonymous hash in the debugger

```
$a = {};  
p ref $a;  
HASH
```

Make a MySequence object in the debugger

```
$self = {};  
$class = 'MySequence';  
bless $self , $class;  
  
x $self  
0 MySequence=HASH(0x18bd7cc)  
    empty hash  
p ref $a  
MySequence
```

Friday, October 21, 2011

19

final step

object-oriented module or class

```
#File: Microarray.pm  
package Microarray;  
use strict;  
  
sub new {  
    my $class = shift;  
    my %args = @_;  
    my $self = {};  
    foreach my $key (keys %args) {  
        $self -> {$key} = $args{$key};  
    }  
    # the magic happens here  
    bless $self, $class;  
    return $self;  
}  
  
sub gene {  
    my $self = shift;  
    return $self->{gene};  
}  
  
sub tissue {  
    my $self = shift;  
    return $self ->{tissue};  
}  
1;
```

Friday, October 21, 2011

20

OOP script

```
#!/usr/bin/perl
use strict; use warnings;          procedural version
#File: script.pl
my $microarray = { gene => 'CDC2',
                  expression => 45,
                  tissue => 'liver',
                  };
my $gene_name = gene($microarray);
print "Gene for this microarray is $gene\n";
```

```
#!/usr/bin/perl
#File: OO_script.pl                OO version
use strict; use warnings;
use Microarray;
my $microarray = Microarray->new( gene => 'CDC2',
                                  expression => 45,
                                  tissue => 'liver',
                                  );
my $gene_name = $microarray->gene();
print "Gene for this microarray is $gene_name\n";
my $tissue = $microarray->tissue();
print "The tissue is $tissue\n";
```

Friday, October 21, 2011

21

Lastly, did I mention “code lazy”?

- This lecture has introduced you to object-oriented programming
- You only need to **use** other people’s objects (beg, borrow, buy, steal).
- Only create your own modules and objects if you **have to**.

Friday, October 21, 2011

22

Aside on inheritance

- If you want to make an object that is a special case or subclass of another, more general, object, you can have it inherit all the general data storage and functions of the more general object.
- This saves coding time by re-using existing code. This also avoids copying and pasting existing code into the new object, a process that makes code harder to maintain and debug.
- For example, a `MicroRNA_gene` object is a special case of a `Gene` object and might have some specific functions like `cut_RNA_hairpin()` as well as general functions like `transcribe()` it can **inherit** from the general gene object.
- More formally, a subclass inherits variables and functions from its superclass (like a child and a parent). Here are some examples

```
package MicroRNA;
use base 'Gene'; # Gene is a parent
use base 'Exporter'; # Exporter is another parent
```

Friday, October 21, 2011

23

Problems

1. Take a look at the `Statistics::Descriptive` module on cpan here <http://search.cpan.org/~shlomif/Statistics-Descriptive-3.0202/lib/Statistics/Descriptive.pm>

2. Write a script that uses the methods in `Statistics::Descriptive` to calculate the standard deviation, median, min and max of the following numbers

12,-13,-12,7,11,-4,-12,9,6,7,-9

Optional questions

4. Add a method to `Microarray.pm` called `expression()` which returns the expression value

5. Currently calling `$a = $m->gene()` gets the value of gene in the object `$m`. Modify the `gene()` method so that it can also set the value of gene if you call `gene()` with an argument, e.g.

```
$m->gene('FOXP1'); # this should set the gene name to
'FOXP1'
print $m->gene(); # this should print the value 'FOXP1'
```

Friday, October 21, 2011

24

Perl Pipelines

Using perl as bioinformatics glue

Simon Prochnik
with code from Scott Cain

Sunday, October 23, 2011

1

`perldoc -f <command>` to get help

```
% perldoc -f split
```

```
split /PATTERN/,EXPR,LIMIT
```

```
split /PATTERN/,EXPR
```

```
split /PATTERN/
```

```
split Splits the string EXPR into a list of strings and returns that list. By default, empty leading fields are preserved, and empty trailing ones are deleted. (If all fields are empty, they are considered to be trailing.)
```

Sunday, October 23, 2011

2

perldoc <perl topic> to get help

```
% perldoc perlref
```

```
PERLREF(1)          User Contributed Perl Documentation          PERLREF
(1)
```

NAME

perlref - Perl references and nested data structures

NOTE

This is complete documentation about all aspects of references. For a shorter, tutorial introduction to just the essential features, see `perlreftut`.

DESCRIPTION

Before release 5 of Perl it was difficult to represent complex data structures, because all references had to be symbolic--and even then it was difficult to refer to a variable instead of a symbol table entry. Perl now not only makes it easier to use symbolic references to variables, but also lets you have "hard" references to any piece of data or code. Any scalar may hold a hard reference. Because arrays and hashes contain scalars, you can now easily build arrays of arrays, arrays of hashes, hashes of arrays, arrays of hashes of functions, and so on.

Sunday, October 23, 2011

3

Get online help from perldoc.perl.org

<http://perldoc.perl.org/functions/split.html>

The screenshot shows the perldoc.perl.org website. The header includes the Perl logo and the text "perldoc.perl.org Perl Programming Documentation". A navigation sidebar on the left has sections for "Perl version" (with a dropdown menu), "Manual" (with links to Overview, Tutorials, FAQs, History / Changes, and License), and "Reference" (with links to Language, Functions, Operators, Special Variables, Pragmas, Utilities, Internals, and Platform Specific). The main content area shows the "split" function documentation for Perl 5 version 12.2. It includes a breadcrumb trail: "Home > Language reference > Functions > split". The title "split" is underlined. Below the title are links for "Perl functions A-Z", "Perl functions by category", and "The 'perifunc' manpage". A list of function signatures is shown:

- `split /PATTERN/,EXPR,LIMIT`
- `split /PATTERN/,EXPR`
- `split /PATTERN/`
- `split`

The description states: "Splits the string EXPR into a list of strings and returns that list. By default, empty leading fields and trailing ones are deleted. (If all fields are empty, they are considered to be trailing.) In scalar context, returns the number of fields found." The final sentence reads: "If EXPR is omitted, splits the \$_ string. If PATTERN is also omitted, splits on whitespace (after stripping whitespace). Anything matching PATTERN is taken to be a delimiter separating the fields. (Note longer than one character.)"

Sunday, October 23, 2011

4

Running your script in the perl debugger

```
> perl -d myScript.pl
Loading DB routines from perl5db.pl version 1.28
Editor support available.
Enter h or `h h' for help, or `man perldebug' for more help.
main::(myScript.pl:3): print "hello world\n";
DB<1>
```

```
h          help
q          quit
n or s     next line or step through next line
<return>  repeat last n or s
c 45      continue to line 45
b 45      break at line 45
b 45 $a == 0  break at line 45 if $a equals 0
p $a      print the value of $a
x $a      unpack or extract the data structure in $a
```

Sunday, October 23, 2011

5

The interactive perl debugger

```
> perl -de 4
Loading DB routines from perl5db.pl version 1.28
Editor support available.

Enter h or `h h' for help, or `man perldebug' for more help.
```

```
main::(-e:1):4
DB<1> $a = {foo => [1,2] , boo => [2,3] , moo => [6,7]}
DB<2> x $a
0 HASH(0x8cd314)
  'boo' => ARRAY(0x8c3298)
    0 2
    1 3
  'foo' => ARRAY(0x8d10d4)
    0 1
    1 2
  'moo' => ARRAY(0x815a88)
    0 6
    1 7
```

Sunday, October 23, 2011

6

More perl tricks: one line perl

```
> perl -e <COMMAND>
```

```
> perl -e '@a = (1..4);print join("\t",@a),"\n"'
1      2      3      4
```

```
#print IDs from fasta file
> perl -ne 'if (/^>(\S+)/) {print "$1\n"}' volvox_AP2EREBP.fa
vca4886446_93762
vca4887371_120236
vca4887497_89954
```

- see Chapter 19, p. 492-502 Perl book 3rd ed.

Sunday, October 23, 2011

7

Is a module installed?

one-line perl program with '-e'

this is the program in quotes

```
% perl -e 'use Bio::AlignIO::clustalw'
```

all ok: no errors

The module in the next example hasn't been installed (it doesn't actually exist)

```
% perl -e 'use Bio::AlignIO::myformat'
Can't locate Bio/AlignIO/myformat.pm in @INC (@INC contains: /sw/lib/perl5 /sw/lib/perl5/darwin /Users/simonp/lib /Users/simonp/Library/Perl/5.8.1/darwin-thread-multi-2level /Users/simonp/Library/Perl/5.8.1 /Users/simonp/com_lib /Users/simonp/cvs/bdgp/software/perl-modules ...)
```

perl can't find the module in any of the paths in the PERL5LIB list (which is in the perl variable @INC)
You can add directories with use lib '/Users/yourname/lib'; after the use strict; at the beginning of your script

To install a module

```
% sudo cpan
install Bio::AlignIO::clustalw
```

Sunday, October 23, 2011

8

Toy example: Finding out how to run a small task

- Let's assume we have a multiple fasta file and we want to use perl to run the program clustalw to make a multiple sequence alignment and read in the results.
- Here are some sequences

```
>vca4886446_93762
MSPPPTHSTTESRMAPPSQSSTPSGDVDGS
>vca4887371_120236
MAGLHSVPKLSARRPDWELPELHGDLQLAP
>vca4887497_89954
MAYKLFGTAAVLNYDLPAERRAELDAMSME
>vca4888938_93984
MLHTDLQPPRCRTSGPRPDPLRMETRER
```

Sunday, October 23, 2011

9

Looking for help with Google

- Google
 - <program name> documentation / docs / command line
 - eg google 'clustal command line'

USE OF OPTIONS

All parameters of Clustalw can be used as options with a "-" That permits to use Clustalw in a script or in batch.

```
$ clustalw -options
```

```
CLUSTAL W (1.7) Multiple Sequence Alignments
```

```
clustalw option list:-
```

```
-help
```

```
-options
```

```
-infile=filename
```

```
-outfile=filename
```

```
-type=protein OR dna
```

```
-output=gcg OR gde OR pir OR phylip
```

Sunday, October 23, 2011

10

Build a command line from the options you need

USE OF OPTIONS

All parameters of Clustalw can be used as options with a "-" That permits to use Clustalw in a script or in batch.

```
$ clustalw -options
CLUSTAL W (1.7) Multiple Sequence Alignments
clustalw option list:-
    -help
    -options
    -infile=filename
    -outfile=filename
    -type=protein OR dna
    -output=gcg OR gde OR pir OR phylip
```

Command line would be:

```
% clustalw -infile=ExDNA.fasta -outfile=ExDNA.aln -type=dna
```

Sunday, October 23, 2011

11

Running a command line from perl

Command line

```
clustalw -infile=ExDNA.fasta -outfile=ExDNA.aln -type=dna
```

Script

```
#!/usr/bin/perl
use strict; use warnings;

my $file = 'ExDNA.fasta';
my $clustFile = 'ExDNA.aln';
# build command line
my $cmd = "clustalw -infile=$file -outfile=$clustFile -type=dna";
print "Call to clustalw $cmd\n";      # show command
my $oops = system $cmd;             # system call and save return
                                     # value in $oops
die "FAILED $!" if $oops;          # $oops true if failed
```

Sunday, October 23, 2011

12

Util.pm package

```
package Util;
use strict;
our @EXPORT = qw(do_or_die);    # allow do_or_die() to be exported
                                # without specifying
                                # Util::do_or_die()

use Exporter;
use base 'Exporter';

# -----
sub do_or_die {
    my $cmd = shift;
    print "CMD: $cmd\n";
    my $oops = system $cmd;
    die "Failed" if $oops;
}
# -----

1;
```

Sunday, October 23, 2011

13

Util.pm in a script

```
#!/usr/bin/perl
use strict; use warnings;
use lib 'lib'; # you might need to tell perl where to find
Util.pm
                # or with something like this
                # use lib '/Users/simonp/lib';
use Util;

my $file = 'ExDNA.fasta';
my $clustFile = 'ExDNA.aln';
my $cmd = "clustalw -infile=$file -outfile=$clustFile
           -type=dna";          # build command line
print "Call to clustalw $cmd\n"; # show command

do_or_die($cmd);    # I use this all the time
```

Sunday, October 23, 2011

14

How do we find out how to parse the clustalw alignment file?

The output is a clustalw multiple sequence alignment in the file ExDNA.aln

Look in bioperl documentation for help.

See HOWTOs

<http://www.bioperl.org/wiki/HOWTOs>

BioPerl HOWTOs

Beginners HOWTO

An introduction to [BioPerl](#), including reading and writing sequence files, running and parsing [BLAST](#), retrieving from databases, and more.

SeqIO HOWTO

Sequence file I/O, with many script examples.

...

AlignIO and SimpleAlign HOWTO

A guide on how to create and analyze alignments using [BioPerl](#).

Sunday, October 23, 2011

15

Help on AlignIO from bioperl

Abstract

This is a HOWTO that talks about using [AlignIO](#) and [SimpleAlign](#) to create and analyze alignments. It also discusses how to run various applications that create alignment files.

AlignIO

Data files storing multiple sequence alignments appear in varied formats and [Bio::AlignIO](#) is the Bioperl object for conversion of alignment files. [AlignIO](#) is patterned on the [Bio::SeqIO](#) object and its commands have many of the same names as the commands in [SeqIO](#). Just as in [SeqIO](#) the [AlignIO](#) object can be created with "-file" and "-format" options:

```
use Bio::AlignIO;
my $io = Bio::AlignIO->new(-file => "receptors.aln",
                          -format => "clustalw" );
```

If the "-format" argument isn't used then Bioperl will try and determine the format based on the file's suffix, in a case-insensitive manner. Here is the current set of input formats:

Format	Suffixes	Comment
bl2seq		
clustalw	aln	

Sunday, October 23, 2011

16

More help on AlignIO from bioperl

Here's a more useful synopsis

```
use Bio::AlignIO;

$in = Bio::AlignIO->new(-file => "inputfilename" ,
                       -format => 'fasta');
$out = Bio::AlignIO->new(-file => ">outputfilename",
                       -format => 'pfam');

while ( my $aln = $in->next_aln ) {
    $out->write_aln($aln);
}
```

Let's add this to our script

Sunday, October 23, 2011

17

Use bioperl to parse the clustalw alignment

Command line

```
clustalw -infile=ExDNA.fasta -outfile=ExDNA.aln -type=dna
```

Script

```
#!/usr/bin/perl
use strict; use warnings;
use Bio::AlignIO;
my $file = 'ExDNA.fasta';
my $clustFile = 'ExDNA.aln';
my $cmd = "clustalw -infile=$file -outfile=$clustFile
           -type=dna";          # build command line
print "Call to clustalw $cmd\n";      # show command
my $oops = system $cmd;             # system call and save return
                                     # value in $oops
die "FAILED $!" if $oops;          # $oops true if failed
my $in = Bio::AlignIO->new(-file => $clustFile,
                          -format => 'clustalw');
while ( my $aln = $in->next_aln() ) {
    ...
}
```

Sunday, October 23, 2011

18

Wait, I haven't told you what a clustalw file looks like

- That's the point of bioperl
- You don't need to know the details of the file format to be able to work with it
- Here's a sample file in case you are curious

CLUSTAL W (1.74) multiple sequence alignment

```
seq1 -----KSKERYKDENGGNYFQLREDWWDANRETVWKAITCNA
seq2 -----YEGLTTANGXKEYYQDKNGGNFFKLREDWWTANRETVWKAITCGA
seq3 ----KRIYKKIFKEIHSGLSTKNGVKDRYQN-DGDNYFQLREDWWTANRSTVWKALTCSD
seq4 -----SQRHYKD-DGGNYFQLREDWWTANRHTVWEAITCSA
seq5 -----NVAALKTRYEK-DGQNFYQLREDWWTANRATIWEAITCSA
seq6 -----FSKNIX--QIEELQDEWLLLEARYKD--TDNYVELREHWWTENRHTVWEALTCEA
seq7 -----KELWEALTCSR

seq1 --GGGKYFRNTCDG--GQNPTEQNNCRGIG-----ATVPTYFDYVPQYLRWSDE
seq2 P-GDASYFHATCDSDGDRGGAQAPHKCRCDG-----ANVPTYFDYVPQFLRWPEE
seq3 KLSNASYFRATC--SDGQSGAQANNYCRCNGDKPDDDKP-NTDPPTYFDYVPQYLRWSEE
seq4 DKGNA-YFRRTCNSADGKSQSQARNQCRC---KDENGKN-ADQVPTYFDYVPQYLRWSEE
seq5 DKGNA-YFRATCNSADGKSQSQARNQCRC---KDENGKN-ADQVPTYFDYVPQYLRWSEE
seq6 P-GNAQYFRNACS----EGKTATKGKCRGISGDP-----PTYFDYVPQYLRWSEE
seq7 P-KGANVFVYKLD-----RPKFSSDRCGHNYNGDP-----LTNLDYVPQYLRWSDE
```

Sunday, October 23, 2011

19

bioperl-run can run clustalw and many other programs

- The Run package (bioperl-run) provides wrappers for executing some 60 common bioinformatics applications (bioperl-run in the repository system Git, see link below)
 - Bio::Tools::Run::Alignment::clustalw
- There are several pieces to bioperl these are all listed here
- http://www.bioperl.org/wiki/Using_Git
 - bioperl-live Core modules including parsers and main objects
 - bioperl-run Wrapper modules around key applications
 - bioperl-ext Ext package has C extensions including alignment routines and link to staden IO library for sequence trace reads.
 - bioperl-pedigree
 - bioperl-microarray
 - bioperl-gui
 - bioperl-db

Sunday, October 23, 2011

20

Smart Essential coding practices

- use strict; use warnings. ALWAYS. Do it!
- Put all the hard stuff in subroutines.
 - This makes the code easy to read and understand.
 - It keeps the code on a single screen, which prevents bugs.
 - Each subroutine should have similar design.
 - If you want to re-use a subroutine several times, put it in a module and re-use the module eg Util.pm
 - don't copy and paste code: bugs multiply, corrections get complicated;
- #comments (ESC-; makes a comment in EMACS)
 - what a subroutine expects and returns
 - anything new to you or unusual
- Use tab indentation for loops, logic, subroutines
 - it's so much easier to spot bugs and follow the code

Sunday, October 23, 2011

21

Coding strategy

- Use the simplest tool for the job: it will be faster to code
- Re-use and modify existing code as much as possible
- Turn to bigger/more complicated tools if and only if you need them:
 - is it going to take less time to wait for your code to finish than learning about a complex tool?
 - is it going to take more time to write a complex tool or search for it on the web or ask your friends what they use?
- Write your code in small pieces and test each piece as you go.
- Check your input data
 - weird characters, line returns (`\r` or `\n` ?), whitespace at the end of lines, spaces instead of tabs. You can use
 - `% od -c mydatafile | more`
 - are there missing pieces, duplicated IDs?
- use a small piece of (real or fake) data to test your code
- Is the output **exactly** what you expect?

Sunday, October 23, 2011

22

gene_pred_pipe.pl (by Scott Cain) part I

```
#!/usr/bin/perl -w

use strict;

use Bio::DB::GenBank;
use Bio::Tools::Run::RepeatMasker;
use Bio::Tools::Run::Genscan;
use Bio::Tools::GFF;

my $acc = $ARGV[0]; # read argument from command line

# main functions in simple subroutines
my $seq_obj = acc_to_seq_obj($acc);
my $masked_seq = repeat_mask($seq_obj);
my @predictions = run_genscan($masked_seq);
predictions_to_gff(@predictions);
warn "Done!\n";
exit(0);
#-----
```

Sunday, October 23, 2011

23

gene_pred_pipe.pl (by Scott Cain) part II

```
sub acc_to_seq_obj {
    #takes a genbank accession, fetches the seq from
    #genbank and returns a Bio::Seq object
    #parent script has to `use Bio::DB::Genbank`
    my $acc = shift;
    my $db = new Bio::DB::GenBank;
    return $db->get_seq_by_id($acc);
}
sub repeat_mask {
    #takes a Bio::Seq object and runs RepeatMasker locally.
    #Parent script must `use Bio::Tools::Run::RepeatMasker`
    my $seq = shift;
    #BTRRM->new() takes a hash for configuration parameters
    #You'll have to set those up appropriately
    my $factory = Bio::Tools::Run::RepeatMasker->new();
    return $factory->masked_seq($seq);
}
```

Sunday, October 23, 2011

24

gene_pred_pipe.pl (by Scott Cain) part III

```
sub run_genscan {
    #takes a Bio::Seq object and runs Genscan locally and returns
    #a list of Bio::SeqFeatureI objects
    #Parent script must `use Bio::Tools::Run::Genscan`
    my $seq = shift;
    #BTRG->new() takes a hash for configuration parameters
    #You'll have to set those up appropriately
    my $factory = Bio::Tools::Run::Genscan->new();
    #produces a list of Bio::Tools::Prediction::Gene objects
    #which inherit from Bio::SeqFeature::Gene::Transcript
    #which is a Bio::SeqFeatureI with child features
    my @genes = $factory->run($seq);
    my @features;
    for my $gene (@genes) {
        push @features, $gene->features;
    }
    return @features;
}
sub predictions_to_gff {
    #takes a list of features and writes GFF2 to a file
    #parent script must `use Bio::Tools::GFF`
    my @features = @_;
    my $gff_out = Bio::Tools::GFF->new(-gff_version => 2,
                                       -file          => '>prediction.gff');
    $gff_out->write_feature($_) for (@features);
    return;
}
```

Sunday, October 23, 2011

25

Getting arguments from the command line with Getopt::Long and GetOptions()

- complicated.pl -flag --pie -start 4
-expect 1e-50 -value=0.00423 -pet cat -pet dog
- order of arguments doesn't matter
- deals with flags, integers, decimals, strings, lists
- an example:-

```
use Getopt::Long;
my ($flag, $count, $price, $string);
GetOptions( "flag" => \$flag,
           "count=i", \$count, # integer
           "price=f", \$price, # floating point 0.12,3e-49
           "name=s", \$string, # always use trailing ','
           );
```

Sunday, October 23, 2011

26

genbank_to_blast.pl (by Scott Cain) part I

```
#!/usr/bin/perl -w
use strict;
use lib "/home/scott/cvs_stuff/bioperl-live"; # this will change depending
# on your machine

use Getopt::Long;
use Bio::DB::GenBank;
#use Bio::Tools::Run::RepeatMasker; # running repeat masked first is a good
# idea, but takes a while

use Bio::Tools::Run::RemoteBlast;
use Bio::SearchIO;
use Bio::SearchIO::Writer::GbrowseGFF;
use Bio::SearchIO::Writer::HTMLResultWriter;
use Data::Dumper; # print out contents of objects etc
#take care of getting arguments
my $usage = "$0 [--html] [--gff] --accession <GB accession number>";
my ($HTML,$GFF,$ACC);
GetOptions ("html" => \$HTML,
           "gff" => \$GFF,
           "accession=s" => \$ACC);
unless ($ACC) {
    warn "$usage\n";
    exit(1);
}
#This will set GFF as the default if nothing is set but allowing both to be set
$GFF ||=1 unless $HTML;
#Now do real stuff ...
```

Sunday, October 23, 2011

27

genbank_to_blast.pl (by Scott Cain) part II

```
# Now do real stuff
# nice and neat subroutine calls
# easy to understand logic of code
my $seq_obj = acc_to_seq_obj($ACC);
my $masked_seq = repeat_mask($seq_obj);
my $blast_res = blast_seq($masked_seq);
gff_out($blast_res, $ACC) if $GFF;
html_out($blast_res, $ACC) if $HTML;
#-----
```

Sunday, October 23, 2011

28

genbank_to_blast.pl (by Scott Cain) part III

```
sub acc_to_seq_obj {
    print STDERR "Getting record from GenBank\n";
    my $acc = shift;
    my $db = new Bio::DB::GenBank;
    return $db->get_seq_by_id($acc);
}
sub repeat_mask {
    my $seq = shift;
    return $seq; #short circuiting RM since we
                #don't have it installed, but this would be where
                # you would run it
#    my $factory = Bio::Tools::Run::RepeatMasker-
>new();
#    return $factory->masked_seq($seq);
}
```

Sunday, October 23, 2011

29

genbank_to_blast.pl (by Scott Cain) part IV

```
sub blast_seq {
    my $seq = shift;
    my $prog = 'blastn';
    my $e_val = '1e-10';
    my $db = 'refseq_rna';
    my @params = (
        -prog => $prog,
        -expect => $e_val,
        -readmethod => 'SearchIO',
        -data => $db
    );
    my $factory = Bio::Tools::Run::RemoteBlast->new(@params);
    $factory->submit_blast($seq);
    my $v = 1; # message flag
    print STDERR "waiting for BLAST..." if ( $v > 0 );
    while ( my @rids = $factory->each_rid ) {
        foreach my $rid ( @rids ) {
            my $rc = $factory->retrieve_blast($rid);
            if ( !ref($rc) ) { #waiting...
                if ( $rc < 0 ) {
                    $factory->remove_rid($rid);
                }
                print STDERR "." if ( $v > 0 );
                sleep 25;
            }
            else {
                print STDERR "\n";
                return $rc->next_result();
            }
        }
    }
}
```

Sunday, October 23, 2011

30

```

sub gff_out {
  my ($result, $acc) = @_;
  my $gff_out = Bio::SearchIO->new(
    -output_format => 'GbrowseGFF',
    -output_signif => 1,
    -file           => ">$acc.gff",
    -reference      => 'query',
    -hsp_tag       => 'match_part',
  );
  $gff_out->write_result($result);
}

sub html_out {
  my ($result, $acc) = @_;
  my $writer = Bio::SearchIO::Writer::HTMLResultWriter->new();
  my $html_out = Bio::SearchIO->new(
    -writer => $writer,
    -format => 'blast',
    -file   => ">$acc.html"
  );
  $html_out->write_result($result);
}

```

Sunday, October 23, 2011

31

HTML version of blast report: NM_000492.html

Bioperl Reformatted HTML of BLASTN Search Report for NM_000492

BLASTN 2.2.12 [Aug-07-2005]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", *Nucleic Acids Res.* 25:3389-3402.

Query= NM_000492 Homo sapiens cystic fibrosis transmembrane conductance regulator, ATP-binding cassette (sub-family C, member 7) (CFTR), mRNA.

(6,129 letters)

Database: NCBI Transcript Reference Sequences

311,041 sequences; 606,661,208 total letters

Sequences producing significant alignments:		Score (bits)	E value
refNM_000492.2	Homo sapiens cystic fibrosis transmembrane conductance re...	1.201e+04	0
refNM_001032938.1	Macaca mulatta cystic fibrosis transmembrane conductance ...	8187	0
refNM_001007143.1	Canis familiaris cystic fibrosis transmembrane conductanc...	5019	0
refNM_174018.2	Bos taurus cystic fibrosis transmembrane conductance regu...	3253	0
refNM_001009781.1	Ovis aries cystic fibrosis transmembrane conductance regu...	3229	0
refNM_021050.1	Mus musculus cystic fibrosis transmembrane conductance re...	888	0
refXM_342645.2	PREDICTED: Rattus norvegicus cystic fibrosis transmembran...	714	0
refXM_347229.2	PREDICTED: Rattus norvegicus similar to cystic fibrosis t...	682	0

Sunday, October 23, 2011

32

GFF output: NM_000492.gff

```
ref|NM_000492.2|      BLASTN  match  1      6129  1.201e+04  +      .      ID=match_sequence1;Target=EST:NM_000492+1+6129
ref|NM_000492.2|      BLASTN  HSP      1      6129  6060      +      .      ID=match_hsp1;Parent=match_sequence1;Target=EST:NM_000492+1+6129
ref|NM_001032938.1|   BLASTN  match  1      4446  8187      +      .      ID=match_sequence2;Target=EST:NM_000492+133+4575
ref|NM_001032938.1|   BLASTN  HSP      1      4446  4130      +      .      ID=match_hsp2;Parent=match_sequence2;Target=EST:NM_000492+133+4575
ref|NM_001007143.1|   BLASTN  match  1      4332  5019      +      .      ID=match_sequence3;Target=EST:NM_000492+133+4455
ref|NM_001007143.1|   BLASTN  HSP      1      4332  2532      +      .      ID=match_hsp3;Parent=match_sequence3;Target=EST:NM_000492+133+4455

ref|NM_000492.2|      BLASTN  match  1      6129  1.201e+04  +      .      ID=match_sequ
ref|NM_000492.2|      BLASTN  HSP      1      6129  6060      +      .      ID=match_hsp1;Parent=
ref|NM_001032938.1|   BLASTN  match  1      4446  8187      +      .      ID=match_sequence2;Tc
ref|NM_001032938.1|   BLASTN  HSP      1      4446  4130      +      .      ID=match_hsp2;Parent=
ref|NM_001007143.1|   BLASTN  match  1      4332  5019      +      .      ID=match_sequence3;Tc
ref|NM_001007143.1|   BLASTN  HSP      1      4332  2532      +      .      ID=match_hsp3;Parent=
ref|NM_174018.2|      BLASTN  match  54     5760  3253      +      .      ID=match_sequence4;Tc
ref|NM_174018.2|      BLASTN  HSP      54     2705  1641      +      .      ID=match_hsp4;Parent=
```

Sunday, October 23, 2011

33

How to approach perl pipelines

- use strict and warnings
- use (bio)perl as glue
- http://www.bioperl.org/wiki/Main_Page
- google.com
- test small pieces as you write them (debugger: `perl -d`)
- construct a command line and test it (catch failure `...or die...`)
- convert into system call, check it worked with small sample dataset
- extend to more complex code only as needed
- if you use code more than once, put it into a subroutine in a module e.g. Util.pm
- get command line arguments with `GetOptions()`

Sunday, October 23, 2011

34

Bioperl I

Sofia Robb

What is Bioperl?

Collection of tools to help you get your work done

Open source, contributed by users

Used by GMOD, wormbase, flybase, me, you

<http://www.bioperl.org>

Why use BioPerl?

Code is already written.

Manipulate sequences.

Run programs (e.g., blast, clustalw and phylip).

Parsing program output (e.g., blast and alignments).

And much, much more. (<http://www.bioperl.org/wiki/Bptutorial.pl>)

Learning about bioperl

Manipulation of sequences from a file

Query a local fasta file

Creating a sequence record

File format conversions

Retrieving annotations

Parsing Blast output

Manipulating Multiple Alignments

Other Cool Things

Learning about Bioperl:

Navigating Bioperl website

Deobfuscator

Bioperl docs

www.bioperl.org Main Page



BioPerl

main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)
- [Tutorials](#)
- [Deobfuscator](#)
- [Browse Modules](#)

[page](#)

[discussion](#)

[view source](#)

[history](#)

Main Page

Welcome to BioPerl, a community effort to produce Perl code which is useful in biology.

For more background on the BioPerl project please see the [History of BioPerl](#).

BioPerl is distributed under the [Perl Artistic License](#). For more information, see [licensing BioPerl](#).

Installation	Documentation	Support
<ul style="list-style-type: none">■ Linux■ Windows■ Mac OSX■ Ubuntu Server	<ul style="list-style-type: none">■ API Docs and BioPerl docs ↗■ HOWTO■ Scrapbook■ The (in)famous Deobfuscator	<ul style="list-style-type: none">■ FAQ■ BioPerl mailing list ↗■ #bioperl 💬■ BioPerl Media options
Developers	How Do I...?	BioPerl-related Distributions
<ul style="list-style-type: none">■ Using Subversion■ Advanced BioPerl■ The SeqIO	<ul style="list-style-type: none">■ ...learn Perl?■ ...find a nice, readable BioPerl overview?	<ul style="list-style-type: none">■ Core■ BioSQL adaptors (BioPerl-db)

OIBIF News

- [Release 1 network](#)
- [BioPerl 1.](#)
- [BioPerl 1.](#)
- [BioPerl br](#)
- [BioPerl 1.](#)
- [Bioperl 1.](#)
- [new Biopo](#)
- [Bioperl 1.](#)
- [Bioperl 1.](#)
- [PopGen t](#)

See also our

HOWTOs

HOWTOs are [narrative](#)-based descriptions of [BioPerl](#) modules focusing more on a concept or a task than one specific module.

BioPerl HOWTOs

[Beginners HOWTO](#) ←

An introduction to [BioPerl](#), including reading and writing sequence files, running and parsing [BLAST](#), retrieving from databases, and more.

[SeqIO HOWTO](#)

Sequence file I/O, with many script examples.

[SearchIO HOWTO](#)

Parsing reports from sequence comparison programs like [BLAST](#) and writing custom reports.

[Tiling HOWTO](#)

Using search reports parsed by SearchIO to obtain robust overall alignment statistics

[Feature-Annotation HOWTO](#)

Reading and writing detailed data associated with sequences.

[SimpleWebAnalysis HOWTO](#)

Submitting sequence data to Web forms and retrieving results.

[Flat Databases HOWTO](#)

Indexing local sequence files for fast retrieval.

[PAML HOWTO](#)

Using the [PAML](#) package using [BioPerl](#).

[OBDA Access HOWTO](#)



BioPerl

main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#) ←
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)

[howto](#)[discussion](#)[view source](#)[history](#)

HOWTO:Beginners

main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)
- [Tutorials](#)
- [Deobfuscator](#)
- [Browse Modules](#)

community

- [News](#)
- [Mailing lists](#)
- [Supporting BioPerl](#)

Contents [\[hide\]](#)

- 1 [Authors](#)
- 2 [Copyright](#)
- 3 [Abstract](#)
- 4 [Introduction](#)
- 5 [Installing Bioperl](#)
- 6 [Getting Assistance](#)
- 7 [Perl Itself](#)
- 8 [Writing a script in Unix](#)
- 9 [Creating a sequence, and an Object](#)
- 10 [Writing a sequence to a file](#)
- 11 [Retrieving a sequence from a file](#)
- 12 [Retrieving a sequence from a database](#)
- 13 [Retrieving multiple sequences from a database](#)
- 14 [The Sequence Object](#)
- 15 [Example Sequence Objects](#)
- 16 [BLAST](#)
- 17 [Indexing for Fast Retrieval](#)
- 18 [More on Bioperl](#)
- 19 [Perl's Documentation System](#)
- 20 [The Basics of Perl Objects](#)
- 21 [A Simple Procedural Example](#)
- 22 [A Simple Object-Oriented Example](#)



main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)
- [Tutorials](#)
- [Deobfuscator](#)
- [Browse Modules](#)

community

- [News](#)
- [Mailing lists](#)
- [Supporting BioPerl](#)
- [BioPerl Media](#)

Deobfuscator

Contents [\[hide\]](#)

- [1 What is the Deobfuscator?](#)
- [2 Where can I find the Deobfuscator?](#)
- [3 Have a suggestion?](#)
- [4 Feature requests](#)
- [5 Bugs](#)

What is the Deobfuscator?

The Deobfuscator was written to make it easier to determine the methods that are available from a given [BioPerl](#) module (a [common BioPerl FAQ](#)).

BioPerl is a highly [object-oriented](#) software package, with often multiple levels of [inheritance](#). Although each individual module is usually well-documented for the methods specific to it, identifying the inherited methods is less straightforward.

The Deobfuscator indexes all of the [BioPerl](#) POD documentation, taking account of the inheritance tree (thanks to [Class::Inspector](#)), and then presents all of the methods available to each module through a searchable web interface.

Where can I find the Deobfuscator?

The Deobfuscator is currently available [here](#), indexing *bioperl-live*.

Welcome to the BioPerl Deobfuscator

[[bioperl-live](#)]

[what is it?](#)

Search **class names** by string or Perl regex (examples: Bio::SeqIO, seq, fasta\$)

OR select a class from the list:

Bio::SearchIO::blast	Event generator for event based parsing of blast reports
Bio::SearchIO::blast_pull	A parser for BLAST output
Bio::SearchIO::blasttable	Driver module for SearchIO for parsing NCBI -m 8/9 format
Bio::SearchIO::blastxml	A SearchIO implementation of NCBI Blast XML parsing.
Bio::SearchIO::megablast	a driver module for Bio::SearchIO to parse megablast reports (format 0)
Bio::Tools::Run::RemoteBlast	Object for remote execution of the NCBI Blast via HTTP
Bio::Tools::Run::StandAloneBlast	Object for the local execution of the NCBI BLAST program suite (blastall, blastpgp, bl2seq). There is experimental support for WU-Blast and NCBI rpsblast.
Bio::Tools::Run::StandAloneNCBIblast	Object for the local execution of the NCBI BLAST program suite (blastall, blastpgp, bl2seq). With experimental support for NCBI rpsblast.

Deobfuscator

Bio::SearchIO::XML::BlastHandler	XML Handler for NCBI Blast XML parsing.
Bio::SearchIO::XML::PsiBlastHandler	XML Handler for NCBI Blast PSIBLAST XML parsing.

sort by method ▾

methods for **Bio::Tools::Run::StandAloneBlast**

executable	Bio::Tools::Run::StandAloneBlast	string representing the full path to the exe	my \$exe = \$blastfactory->executable('blasta
finally	Bio::Root::Root	not documented	not documented
io	Bio::Tools::Run::WrapperBase	Bio::Root::IO object	\$obj->io(\$newval)
new	Bio::Tools::Run::StandAloneBlast	Bio::Tools::Run::StandAloneNCBIBlast or StandAloneWUBlast	my \$obj = Bio::Tools::Run::StandAloneBlast
no_param_checks	Bio::Tools::Run::WrapperBase	value of no_param_checks	\$obj->no_param_checks(\$newva
otherwise	Bio::Root::Root	not documented	not documented
outfile_name	Bio::Tools::Run::WrapperBase	string	my \$outfile = \$wrapper->outfile_
program	Bio::Tools::Run::StandAloneBlast	not documented	not documented



BioPerl

main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#) ←
- [Scrapbook](#)
- [BioPerl Tutorial](#)
- [Tutorials](#)
- [Deobfuscator](#)

[page](#)

[discussion](#)

[view source](#)

[history](#)

API Docs

Contents [\[hide\]](#)

- 1 [Online POD Documentation](#)
- 2 [Documentation from the Deobfuscator](#)
- 3 [Documentation from the CPAN](#)
- 4 [Browsing Subversion repositories](#)

Online POD Documentation

POD Documentation is available for bioperl-live and past releases [at doc.bioperl.org](http://doc.bioperl.org).

Alternatively you can enter the module name in the search box and see any contributed Wiki documentation for the module.

Documentation from the Deobfuscator

The [Deobfuscator](#) indexes all of the BioPerl POD documentation, taking account of the inheritance tree, and then presents all of the methods available to each module through a [searchable web interface](#).

Documentation from the CPAN



Perldoc ([Pdoc rendered](#)) documentation for BioPerl Modules

Released Code

Official documentation for released code is available here:

- [BioPerl 1.6.0](#), download the entire doc set [here](#).
- [BioPerl 1.5.2](#), download the entire doc set [here](#).
- [BioPerl 1.5.1](#), download the entire doc set [here](#).
- [BioPerl 1.4](#), download the entire doc set [here](#).
- [BioPerl 1.2.3](#), download the entire doc set [here](#).
- [BioPerl 1.2.2](#), download the entire doc set [here](#).
- [BioPerl 1.2](#), download the entire doc set [here](#).
- [BioPerl 1.0.2](#), download the entire doc set [here](#).
- [BioPerl 1.0.1](#), download the entire doc set [here](#).
- [BioPerl 1.0](#), download the entire doc set [here](#).

Active Code

This documentation represents the active development code and is autogenerated daily from the SVN repository:

Module	Description
■ bioperl-live	BioPerl Core Code
■ bioperl-corba-server	BioPerl BioCORBA Server Toolkit (wraps bioperl objects as BioCORBA objects and runs them in an ORBit ORB)
■ bioperl-corba-client	BioPerl BioCORBA Client Toolkit (wraps BioCORBA objects as bioperl objects)

All Modules TOC All

bioperl-live
bioperl-live::Bio
bioperl-live::Bio::Align
bioperl-live::Bio::AlignIO
bioperl-

PhyloNetwork
PrimarySeq
PrimarySeqI
PullParserI
Range
RangeI
SearchDist
SearchIO
Seq
SeqAnalysisParserI
SeqFeatureI
SeqI
SeqIO
SeqUtils
SimpleAlign
SimpleAnalysisI

Bio SeqIO

[Summary](#) [Included libraries](#) [Package variables](#) [Synopsis](#) [Description](#) [General documentation](#) [Methods](#)

Toolbar

[WebCvs](#)

Summary

Bio::SeqIO - Handler for SeqIO Formats

Package variables

Privates (from "my" definitions)

```
%valid_alphabet_cache;  
$entry = 0
```

Included modules

[Bio::Factory::FTLocationFactory](#)
[Bio::Seq::SeqBuilder](#)
[Bio::Tools::GuessSeqFormat](#)
[Symbol](#)

Inherit

[Bio::Factory::SequenceStreamI](#) [Bio::Root::IO](#) [Bio::Root::Root](#)

Synopsis

Bio::SeqIO module synopsis

doc.bioperl.org

Synopsis

```
use Bio::SeqIO;

$in  = Bio::SeqIO->new(-file => "inputfilename" ,
                      -format => 'Fasta');
$out = Bio::SeqIO->new(-file => ">outputfilename" ,
                      -format => 'EMBL');

while ( my $seq = $in->next_seq() ) {
    $out->write_seq($seq);
}

# Now, to actually get at the sequence object, use the standard Bio::Seq
# methods (look at Bio::Seq if you don't know what they are)

use Bio::SeqIO;

$in  = Bio::SeqIO->new(-file => "inputfilename" ,
                      -format => 'genbank');

while ( my $seq = $in->next_seq() ) {
    print "Sequence ", $seq->id, " first 10 bases ",
          $seq->subseq(1,10), "\n";
}

# The SeqIO system does have a filehandle binding. Most people find this
```

Bio::SeqIO module description

doc.bioperl.org

Description

Bio::SeqIO is a handler module for the formats in the SeqIO set (eg, Bio::SeqIO::fasta). It is the officially sanctioned way of getting at the format objects, which most people should use.

The **Bio::SeqIO** system can be thought of like biological file handles. They are attached to filehandles with smart formatting rules (eg, genbank format, or EMBL format, or binary trace file format) and can either read or write sequence objects (Bio::Seq objects, or more correctly, Bio::SeqI implementing objects, of which Bio::Seq is one such object). If you want to know what to do with a Bio::Seq object, read **Bio::Seq**.

The idea is that you request a stream object for a particular format. All the stream objects have a notion of an internal file that is read from or written to. A particular SeqIO object instance is configured for either input or output. A specific example of a stream object is the Bio::SeqIO::fasta object.

Each stream object has functions

```
$stream->next_seq();
```

and

```
$stream->write_seq($seq);
```

Bio::SeqIO method list

doc.bioperl.org

Methods		
new	Description	Code
newFh	Description	Code
fh	Description	Code
_initialize	No description	Code
next_seq	Description	Code
write_seq	Description	Code
alphabet	Description	Code
_load_format_module	Description	Code
_concatenate_lines	Description	Code
_filehandle	Description	Code
_guess_format	Description	Code
DESTROY	No description	Code
TIEHANDLE	Description	Code
READLINE	No description	Code

Bio::SeqIO new method description

doc.bioperl.org

Methods description

[new](#)

[code](#)

[next](#)

[Top](#)

```
Title      : new
Usage      : $stream = Bio::SeqIO->new(-file => $filename,
                                       -format => 'Format')
Function: Returns a new sequence stream
Returns   : A Bio::SeqIO stream initialised with the appropriate format
Args      : Named parameters:
            -file => $filename
            -fh => filehandle to attach to
            -format => format

Additional arguments may be used to set factories and
builders involved in the sequence object creation. None of
these must be provided, they all have reasonable defaults.
    -seqfactory   the Bio::Factory::SequenceFactoryI object
    -locfactory   the Bio::Factory::LocationFactoryI object
    -objbuilder   the Bio::Factory::ObjectBuilderI object
```

See [Bio::SeqIO::Handler](#)

Manipulation of sequences from a file

Problem:

You have a sequence file and you want to do something to each sequence.

What do you do first?

HowTo:

<http://www.bioperl.org/wiki/HOWTOs>

HOWTOs

HOWTOs are [narrative](#)-based descriptions of [BioPerl](#) modules focusing more on a concept or a task than one specific module.

BioPerl HOWTOs

Beginners HOWTO



An introduction to [BioPerl](#), including reading and writing sequence files, running and parsing [BLAST](#), retrieving from databases, and more.

SeqIO HOWTO

Sequence file I/O, with many script examples.

SearchIO HOWTO

Parsing reports from sequence comparison programs like [BLAST](#) and writing custom reports.

Tiling HOWTO

Using search reports parsed by SearchIO to obtain robust overall alignment statistics

Feature-Annotation HOWTO

Reading and writing detailed data associated with sequences.

SimpleWebAnalysis HOWTO

Submitting sequence data to Web forms and retrieving results.

Flat Databases HOWTO

Indexing local sequence files for fast retrieval.

PAML HOWTO

Using the [PAML](#) package using [BioPerl](#).

OBDA Access HOWTO



main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)





BioPerl

[howto](#)

[discussion](#)

[view source](#)

[history](#)

HOWTO:Beginners

main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)
- [Tutorials](#)
- [Deobfuscator](#)
- [Browse Modules](#)

community

- [News](#)
- [Mailing lists](#)
- [Supporting BioPerl](#)

Contents [\[hide\]](#)

- 1 [Authors](#)
- 2 [Copyright](#)
- 3 [Abstract](#)
- 4 [Introduction](#)
- 5 [Installing Bioperl](#)
- 6 [Getting Assistance](#)
- 7 [Perl Itself](#)
- 8 [Writing a script in Unix](#)
- 9 [Creating a sequence, and an Object](#)
- 10 [Writing a sequence to a file](#)
- 11 [Retrieving a sequence from a file](#)
- 12 [Retrieving a sequence from a database](#)
- 13 [Retrieving multiple sequences from a database](#)
- 14 [The Sequence Object](#)
- 15 [Example Sequence Objects](#)
- 16 [BLAST](#)
- 17 [Indexing for Fast Retrieval](#)
- 18 [More on Bioperl](#)
- 19 [Perl's Documentation System](#)
- 20 [The Basics of Perl Objects](#)
- 21 [A Simple Procedural Example](#)
- 22 [A Simple Object-Oriented Example](#)



Retrieving a sequence from a file

One beginner's mistake is to not use `Bio::SeqIO` when working with sequence files. This is understandable in some respects. You may have read about Perl's `open` function, and Bioperl's way of retrieving sequences may look odd and overly complicated, at first. But don't use `open`! Using `open` immediately forces you to do the parsing of the sequence file and this can get complicated very quickly. Trust the `SeqIO` object, it's built to open and parse all the common [sequence formats](#), it can read and write to files, and it's built to operate with all the other Bioperl modules that you will want to use.

Let's read the file we created previously, "sequence.fasta", using `SeqIO`. The syntax will look familiar:

```
#!/bin/perl -w

use Bio::SeqIO;

$seqio_obj = Bio::SeqIO->new(-file => "sequence.fasta", -format => "fasta" );
```

One difference is immediately apparent: there is no `>` character. Just as with with the `open()` function this means we'll be reading from the "sequence.fasta" file. Let's add the key line, where we actually retrieve the Sequence object from the file using the `next_seq` method:

```
#!/bin/perl -w

use Bio::SeqIO;

$seqio_obj = Bio::SeqIO->new(-file => "sequence.fasta", -format => "fasta" );

$seq_obj = $seqio_obj->next_seq;
```



main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)

HOWTOs

HOWTOs are [narrative](#)-based descriptions of [BioPerl](#) modules focusing more on a concept or a task than one specific module.

BioPerl HOWTOs

Beginners HOWTO

An introduction to [BioPerl](#), including reading and writing sequence files, running and parsing [BLAST](#), retrieving from databases, and more.

SeqIO HOWTO

Sequence file I/O, with many script examples.

SearchIO HOWTO

Parsing reports from sequence comparison programs like [BLAST](#) and writing custom reports.

Tiling HOWTO

Using search reports parsed by SearchIO to obtain robust overall alignment statistics

Feature-Annotation HOWTO

Reading and writing detailed data associated with sequences.

SimpleWebAnalysis HOWTO

Submitting sequence data to Web forms and retrieving results.

Flat Databases HOWTO

Indexing local sequence files for fast retrieval.

PAML HOWTO

Using the [PAML](#) package using [BioPerl](#).

OBDA Access HOWTO





BioPerl

[howto](#)

[discussion](#)

[view source](#)

[history](#)

HOWTO:SeqIO

This HOWTO will teach you about the [Bio::SeqIO](#) system for reading and writing sequences of various formats

Contents [\[hide\]](#)

- [1 The basics](#)
- [2 10 second overview](#)
- [3 Background Information](#)
- [4 Formats](#)
- [5 Working Examples](#)
- [6 To and From a String](#)
- [7 And more examples...](#)
- [8 Caveats](#)
- [9 Error Handling](#)
- [10 Speed, Bio::Seq::SeqBuilder](#)

The basics

This section assumes you've never seen [BioPerl](#) before, perhaps you're a biologist trying to get some information about this hot topic, "[bioinformatics](#)". Your first script may want to get some information from a file con

A piece of advice: always use the module [Bio::SeqIO](#)! Here's what the first lines of your script might look like:

```
#!/bin/perl

use strict;
use Bio::SeqIO;

my $file = shift; # get the file name, somehow
my $seqio_object = Bio::SeqIO->new(-file => $file);
my $seq_object = $seqio_object->next_seq;
```

main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)
- [Tutorials](#)
- [Deobfuscator](#)
- [Browse Modules](#)

community

- [News](#)
- [Mailing lists](#)
- [Supporting BioPerl](#)
- [BioPerl Media](#)
- [Hot Topics](#)
- [About this site](#)


```
#!/usr/bin/perl -w
#file: inFasta_loop.pl
use strict;
use Bio::SeqIO;

my $file = shift;

my $seqIO_object = Bio::SeqIO->new(
    -file => $file,
    -format => 'fasta',
);

while (my $seq_object = $seqIO_object->next_seq) {
    #do stuff to each sequence in the fasta
}
```

What is a SeqIO object?
What is a Seq object?

Objects

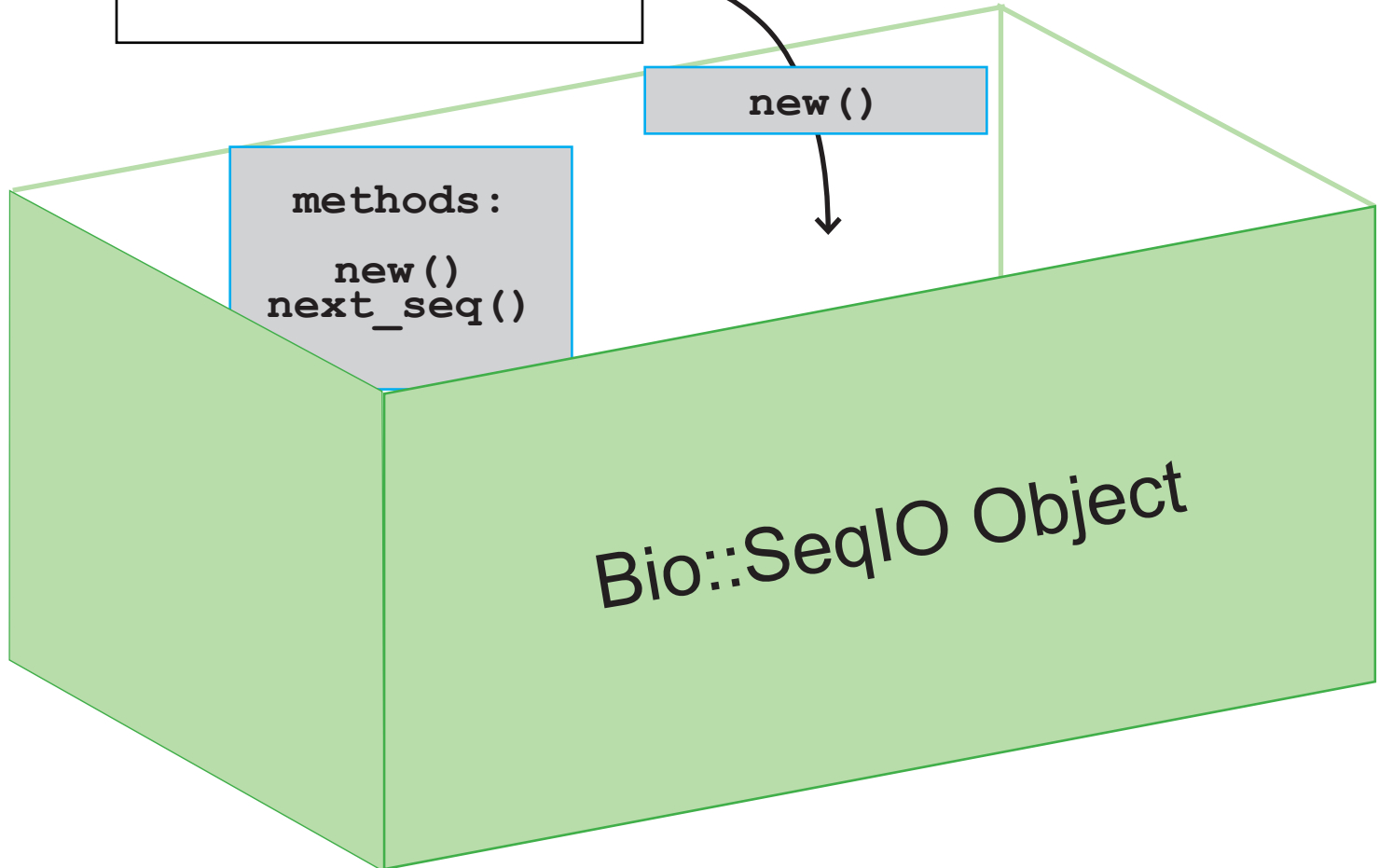
Objects are like boxes that hold
your data and
tools (methods) for your data

data:

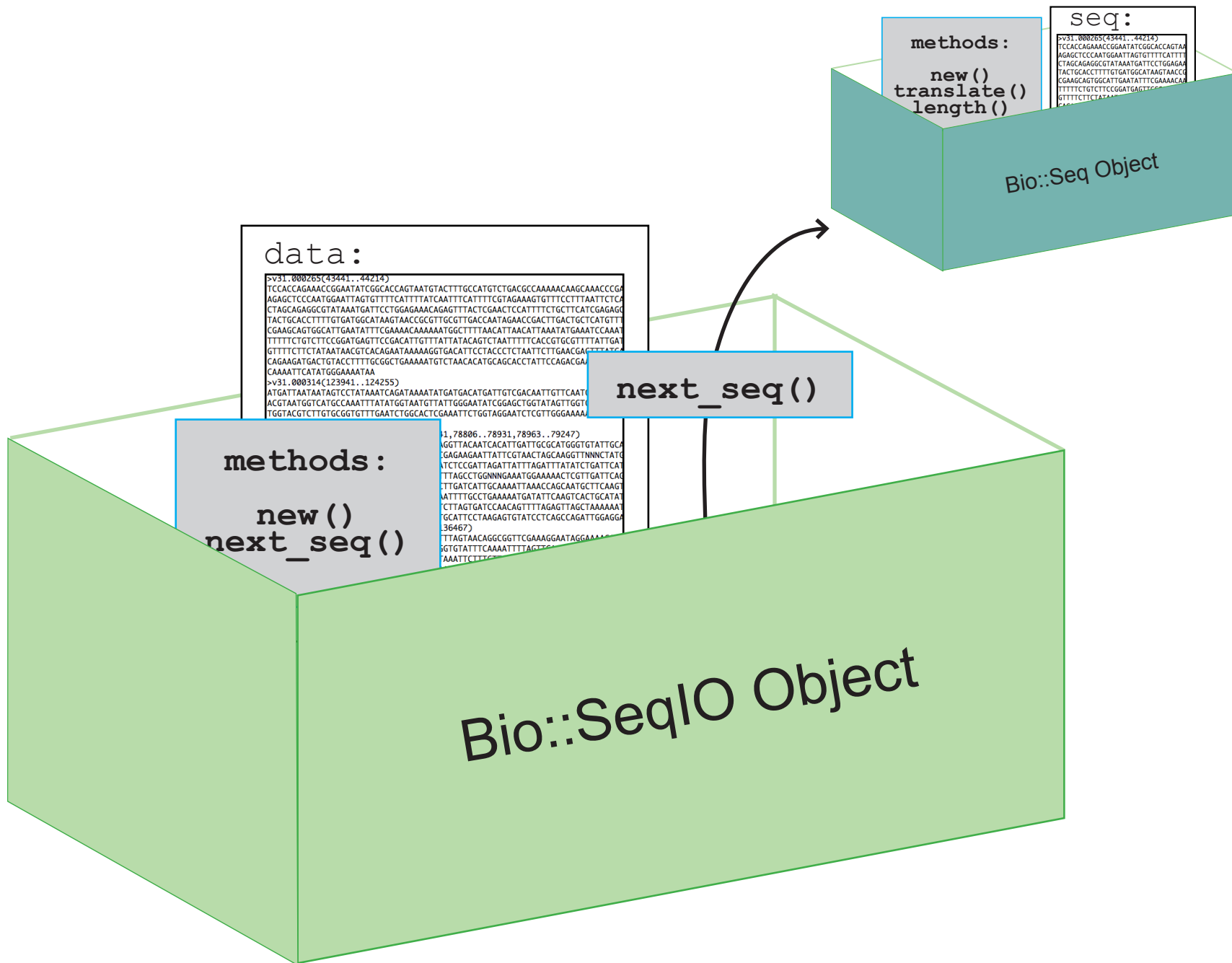
```
>v31.000265(43441..44214)
TCCACCAGAAACGGGAATATCGGCACAGTAATGTACTTTGCCATGTCTGACGCCAAAAACAAGCAAACCCG
AGAGCTCCCAATGGAAATAGTGTTCATTTTATCAATTTTCATTTCTGAGAAAGTGTTCCTTTAATTC
CTAGCAGAGGCGTATAAATGATTCCTGGAGAAACAGAGTTACTCGAAGTCCATTTCTGCTTCATCGAGAG
TACTGCACCTTTGTGATGGCATAAGTAACCGGTTGCGTTGACCAATAGAACCAGCTTGACTGCTCATGTT
CGAAGCAGTGGCATTGAATTTTGGAAACAAAAATGGCTTTAACATTAACATTAATATGAAATCAAAT
TTTTCTGTCTCCGGATGAGTTCCGACATGTTTATATACAGCTCAATTTTCCAGCGTGTGTTATGAT
GTTTTCTCTATAAAGCTCAGAAATAAAAAGGTGACATCTACCTCTAATCTTGAACGAGTTTATGA
CAGAAGATGACTGTACTTTGGGCTGAAAAATGTCAACACATGCGACCTATTCCAGACGAACCTCCA
CAAATTCATATGGAAAAATA
>v31.000314(123941..124255)
ATGATTAATAAGTCTATAAATCAGATAAAAATGATGACATGATTGTCGACAAATGTTCAATGCAAATGG
ACGTAATGGTCATGCCAAATTTATATGGTAATGTTATGGGAATATCGGAGCTGGTATAGTTGGTGGCTGG
TGGTACGCTTGTGCGGTTTGAATCTGGCCTCGAAATCTGGTAGGAATCTGTTGGAAAAACATAGCG
ACGCTCTGTGACATGTTAGACTATTTAGGTGAT
>v31.000349(76077..76235,76277..76441,78806..78931,78963..79247)
GAAATTAAGCTCTTTAGAAAGTCTCTTAATTTAGGTTACAATCACATGATTGCGCATGGGTGATTTGCG
TTTTAGAGACTATATTTCAAATGGTAAATTAACGAGAAGAAATTCGTAAGTACGCAAGGTTNNCTATG
ACTAGTGAAGTCGGATTTGAAGAAACACTTGAATCTCCGATTAGATTATTTAGATTATATCTGATTCAT
TTTTAATTTATATCTCATTTTCATTTTATTTTCATTTAGCTGGNNGAAATGAAACCTCGTTGATTCAG
TGTCTAATTTCAATTTAGACAAATTCAAAATATCTTGATCATTGCAAAATTAACACAGCAATGCTCAAGT
GGTCAATTTTATTTCCGAATAAAAACTAGTGGAAATTTGCTGAAAAATGATATTCAAGTCACTGCATAT
GATCGACTGGAGCAGACAGGGGAAAAACGCTCTTAGTGATCCAACAGTTTTAGAGTAGCTAAAAAAT
TATTTGATAAGCTATATTTATCCTGGGTTTGTCTGATCCTAAGAGTGTATCTCAGCCAGATTGGAGG
>v31.000482(133155..133514,136192..136467)
ATGACGTTAAATACAAATTTCTGGAAAAACAGTTTGTAGTAAACAGCGGTTGAAAGGAATAGGAAACAGT
CTGGAGCTAGTGTTATTTCAATTAAGCAGATCTACTGGTATTTCAAAATTTAGTTCAAAGGAAATTAATTT
TGATATTAGTAATGGGATGAATTTAATCCATAATAAATCTTTGTGCTGTGGATTCTTGTATAAAT
AAATTTGGTGACATCAATGAAAGAAATGACGAAATGATCAATCAAAATGCAATCAGTTATCAACATTT
AGCGGGCCTGGATGCTATTACAGAAATATGGCTTGTAGCTTGGCCGCATACACATACGGGTGAATCTGT
AGATATGGGTGCTTTTATTTGAAATGACGAGTCTAAGAGGATAAGTTGATTTGAGAAATCTTTTGGCGAGA
GTGATGTTATTATGTTTGTGACAGATCATTGCACGCTTGTGACGGGGGGCGCAATCCCATTTGATGGT
```

new ()

methods :
new ()
next_seq ()



Bio::SeqIO Object



```
#!/usr/bin/perl -w
#file: inFasta_loop.pl
use strict;
use Bio::SeqIO;

# get fasta filename from user input
my $file = shift;

# create a SeqIO obj with $file as filename
# $SeqIO_object contains all the individual sequence
# that are in file named $file
my $seqIO_object = Bio::SeqIO->new(
    -file => $file,
    -format => 'fasta',
);

# using while loop and next_seq method to "get to"
# and create a Seq obj for each individual sequence
# in the SeqIO obj of many sequences
while (my $seq_object = $seqIO_object->next_seq) {
    #do stuff to each sequence in the fasta
}
```

1. Get a file name from user input (@ARGV) and stores in \$file

```
#!/usr/bin/perl -w  
use strict;  
use Bio::SeqIO;
```

2. Create a new seqIO object in \$seqIO_object, using filename \$file and format 'fasta'

```
my $file = shift;  
my $seqIO_object = Bio::SeqIO->new(  
    -file => $file,  
    -format => 'fasta',  
);
```

3. Create a second seqIO object in \$out using format 'fasta'

```
my $out = Bio::SeqIO->new(-format => 'fasta');
```

4. Loop thru each seq object in \$seqIO_object storing information from the object in variables.

```
while (my $seq_object = $seqIO_object->next_seq){  
    my $id = $seq_object->id;  
    my $desc = $seq_object->desc;  
    my $seqString = $seq_object->seq;  
    my $revComp = $seq_object->revcom;  
    my $alphabet = $seq_object-> alphabet;  
    my $translation_seq_obj = $seq_object-> translate;  
    my $translation = $translation_seq_obj -> seq;  
    my $seqLen = $seq_object->length;
```

5. Print out the stored information

```
print "translation: $translation\n";  
print "alphabet: $alphabet\n";  
print "seqLen: $seqLen\n";
```

6. Print out \$seq_object using the method or tool 'write_seq()' and the seqIO object \$out.

```
#prints to STDOUT  
$out->write_seq($seq_object);
```

```
}
```

fasta input:

```
>seqName seq description is blah blah blah
AGGCTCAATTTAGTTTTCTTGTCCTTATTTTAAAAGGTGTCCAGTG
TGATGTGCAGCTGGTGGAGTCTGGGGGAGGCTTAGTGCAGCCTGGAG
GGTCCCGGAAACTCTCCTGTGCAGCCTCTGGATTCACTTTCAGTAGC
TTTGG AATGCACTGGGTTCGTCAGGCTCCAGAGAAGGGGCTGGAGTG
GGTCGCATACATTAGTAGTGGCAGTAGTACCCTCCACTATGCAGACA
CAGTGAAGGGCCGATTCACCATCTCAAGAGACAATCCCAAGAACACC
CTGTTCCCTGCAAATGACCAGTCTAAGGTCTGAGGACACGGCCATGTA
T TACTGTGCAAGATGGGGTAACTACCCTTACTATGCTATGGACTACT
GGGGTCAA
```

```
translation: RLNLVFLVLILKGVQCDVQLVESGGGLVQPGGSRKLSAASGFTFSSF
GMHWVRQAPEKGLEWVAYISSGSSTLHYADTVKGRFTISRDNPKNTLFLQMTSLRSEDAM
YYCARWGNYPYYAMDYWGQGTSVTVSS
```

```
alphapet: dna
```

```
seqLen: 408
```

```
>seqName seq description is blah blah blah
```

```
AGGCTCAATTTAGTTTTCTTGTCCTTATTTTAAAAGGTGTCCAGTGTGATGTGCAGCTG
GTGGAGTCTGGGGGAGGCTTAGTGCAGCCTGGAGGGTCCCGGAAACTCTCCTGTGCAGCC
TCTGGATTCACTTTCAGTAGCTTTGGAATGCACTGGGTTCGTCAGGCTCCAGAGAAGGGG
CTGGAGTGGGTTCGCATACATTAGTAGTGGCAGTAGTACCCTCCACTATGCAGACACAGTG
AAGGGCCGATTCACCATCTCAAGAGACAATCCCAAGAACACCCTGTTCCCTGCAAATGACC
AGTCTAAGGTCTGAGGACACGGCCATGTATTACTGTGCAAGATGGGGTAACTACCCTTAC
TATGCTATGGACTACTGGGGTCAAGGAACCTCAGTCACCGTCTCCTCA
```

output:

Table from <http://www.bioperl.org/wiki/HOWTO:Beginners>

List of seq object methods

Table 1: Sequence Object Methods

Name	Returns	Example	Note
new	Sequence object	<code>\$so = Bio::Seq->new(-seq => "MPQRAS")</code>	create a new one, see Bio::Seq for more
seq	sequence string	<code>\$seq = \$so->seq</code>	get or set the sequence
display_id	identifier	<code>\$so->display_id("NP_123456")</code>	get or set an identifier
primary_id	identifier	<code>\$so->primary_id(12345)</code>	get or set an identifier
desc	description	<code>\$so->desc("Example 1")</code>	get or set a description
accession	identifier	<code>\$acc = \$so->accession</code>	get or set an identifier
length	length, a number	<code>\$len = \$so->length</code>	get the length
alphabet	alphabet	<code>\$so->alphabet('dna')</code>	get or set the alphabet ('dna', 'rna', 'protein')
subseq	sequence string	<code>\$string = \$seq_obj->subseq(10,40)</code>	Arguments are start and end
trunc	Sequence object	<code>\$so2 = \$so1->trunc(10,40)</code>	Arguments are start and end
revcom	Sequence object	<code>\$so2 = \$so1->revcom</code>	Reverse complement
translate	protein Sequence object	<code>\$prot_obj = \$dna_obj->translate</code>	See the Bioperl Tutorial ↗ for more
species	Species object	<code>\$species_obj = \$so->species</code>	See Bio::Species for more


```
#!/usr/bin/perl -w
use strict;
use Bio::SeqIO;
```

```
my $file = shift;
my $seqIO_object = Bio::SeqIO->new(
    -file => $file,
    -format => 'fasta',
);
```

```
my $out = Bio::SeqIO->new(-format => 'genbank');
```

```
while (my $seq_object = $seqIO_object->next_seq){
    $out->write_seq($seq_object); #prints to STDOUT
}
```



Change 'format' in the new() method from 'fasta' to 'genbank' to change the way the SeqIO object \$out is displayed in STDOUT.

```
LOCUS          seqName                408 bp    dna    linear    UNK
DEFINITION    seq description is blah blah blah
ACCESSION    unknown
FEATURES             Location/Qualifiers
BASE COUNT          95 a     98 c     111 g     104 t
ORIGIN
     1  aggctcaatt tagttttcct tgtccttatt ttaaaagggtg tccagtgtga tgtgcagctg
    61  gtggagtctg ggggaggcct agtgcagcct ggagggtccc ggaaactctc ctgtgcagcc
   121  tctggattca ctttcagtag ctttggaatg cactgggttc gtcaggctcc agagaagggg
   181  ctggagtggg tcgcatacat tagtagtggc agtagtacc tccactatgc agacacagtg
   241  aagggccgat tcaccatctc aagagacaat cccaagaaca cctgttcct gcaaatgacc
   301  agtctaaggt ctgaggacac ggccatgtat tactgtgcaa gatggggtaa ctacccttac
   361  tatgctatgg actactgggg tcaaggaacc tcagtcaccg tctcctca
```

Query a local fasta file

Query a local fasta file

You have a fasta file that contains many records.

You want to retrieve a specific record.

You do not want to loop through all records until you find the correct record.

Use `Bio::DB::Fasta`.



BioPerl

main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)
- [Tutorials](#)
- [Deobfuscator](#)
- [Browse Modules](#)

community

- [News](#)
- [Mailing lists](#)
- [Supporting BioPerl](#)
- [BioPerl Media](#)

Deobfuscator

Contents [\[hide\]](#)

- [1 What is the Deobfuscator?](#)
- [2 Where can I find the Deobfuscator?](#)
- [3 Have a suggestion?](#)
- [4 Feature requests](#)
- [5 Bugs](#)

What is the Deobfuscator?

The Deobfuscator was written to make it easier to determine the methods that are available from a given [BioPerl](#) module (a [common BioPerl FAQ](#)).

BioPerl is a highly [object-oriented](#) software package, with often multiple levels of [inheritance](#). Although each individual module is usually well-documented for the methods specific to it, identifying the inherited methods is less straightforward.

The Deobfuscator indexes all of the [BioPerl](#) POD documentation, taking account of the inheritance tree (thanks to [Class::Inspector](#)), and then presents all of the methods available to each module through a searchable web interface.

Where can I find the Deobfuscator?

The Deobfuscator is currently available [here](#), indexing *bioperl-live*.

Welcome to the BioPerl Deobfuscator

[[bioperl-live](#)]


Search **class names** by string or Perl regex (examples: Bio::SeqIO, seq, fasta\$)




OR select a class from the list:

Bio::AlignIO::fasta	fasta MSA Sequence input/output stream
Bio::AlignIO::largemultifasta	Largemultifasta MSA Sequence input/output stream
Bio::AlignIO::metafasta	Metafasta MSA Sequence input/output stream
Bio::DB::Fasta	Fast indexed access to a directory of fasta files
Bio::DB::Flat::BDB::fasta	fasta adaptor for Open-bio standard BDB-indexed flat file
Bio::Index::Fasta	Interface for indexing (multiple) fasta files
Bio::Search::HSP::FastaHSP	HSP object for FASTA specific data
Bio::Search::Hit::Fasta	Hit object specific for Fasta-generated hits
Bio::SearchIO::fasta	A SearchIO parser for FASTA results
Bio::Seq::SeqFastaSpeedFactory	Instantiates a new Bio::PrimarySeqI (or derived class) through a factory

Bio::AlignIO::metafasta	Metafasta MSA Sequence input/output stream
Bio::DB::Fasta	Fast indexed access to a directory of fasta files
Bio::DB::Flat::BDB::fasta	fasta adaptor for Open-bio standard BDB-indexed flat file
Bio::Index::Fasta	Interface for indexing (multiple) fasta files
Bio::Search::HSP::FastaHSP	HSP object for FASTA specific data
Bio::Search::Hit::Fasta	Hit object specific for Fasta-generated hits
Bio::SearchIO::fasta	A SearchIO parser for FASTA results
Bio::Seq::SeqFastaSpeedFactory	Instantiates a new Bio::PrimarySeqI (or derived class) through a factory

sort by method 

methods for Bio::DB::Fasta			
Method	Class	Returns	Usage
alphabet	Bio::DB::Fasta 	not documented	not documented
basename	Bio::DB::Fasta	not documented	not documented
calculate_offsets	Bio::DB::Fasta	not documented	not documented
caloffset	Bio::DB::Fasta	not documented	not documented
carp	Bio::Root::RootI	not documented	not documented
CLEAR	Bio::DB::Fasta	not documented	not documented
confess	Bio::Root::RootI	not documented	not documented
dbmargs	Bio::DB::Fasta	not documented	not documented
debug	Bio::Root::Root	none	\$obj->debug("This is debugging output"):

Bio::DB::Fasta

Other packages in the module: [Bio::DB::Fasta](#) [Bio::PrimarySeq::Fasta](#)

[Summary](#)[Included libraries](#)[Package variables](#)[Synopsis](#)[Description](#)

Toolbar

[WebCvs](#)

Summary

Bio::DB::Fasta -- Fast indexed access to a directory of fasta files

Package variables

No package variables defined.

Included modules

[AnyDBM_File](#)[Fcntl](#)[File::Basename](#) qw ([basename](#) [dirname](#))[IO::File](#)

Inherit

[Bio::DB::SeqI](#) [Bio::Root::Root](#)

Synopsis

```
use Bio::DB::Fasta;

# create database from directory of fasta files
my $fdb = Bio::DB::Fasta->new('/path/to/fasta/files');
```

Can also find these pages at <http://doc.bioperl.org/bioperl-live/>

Bio::DB::fasta module synopsis

doc.bioperl.org

Synopsis

```
use Bio::DB::Fasta;

# create database from directory of fasta files
my $db      = Bio::DB::Fasta->new('/path/to/fasta/files');

# simple access (for those without Bioperl)
my $seq     = $db->seq('CHROMOSOME_I',4_000_000 => 4_100_000);
my $revseq  = $db->seq('CHROMOSOME_I',4_100_000 => 4_000_000);
my @ids     = $db->ids;
my $length  = $db->length('CHROMOSOME_I');
my $alphabet = $db->alphabet('CHROMOSOME_I');
my $header  = $db->header('CHROMOSOME_I');

# Bioperl-style access
my $db      = Bio::DB::Fasta->new('/path/to/fasta/files');

my $obj     = $db->get_Seq_by_id('CHROMOSOME_I');
my $seq     = $obj->seq; # sequence string
my $subseq  = $obj->subseq(4_000_000 => 4_100_000); # string
my $trunc   = $obj->trunc(4_000_000 => 4_100_000); # seq object
my $length  = $obj->length;
# (etc)

# Bio::SeqIO-style access
my $stream  = Bio::DB::Fasta->new('/path/to/files')->get_PrimarySeq_stream;
while (my $seq = $stream->next_seq) {
    # Bio::PrimarySeqI stuff
}

my $fh = Bio::DB::Fasta->newFh('/path/to/fasta/files');
```


Bio::DB::fasta module description

doc.bioperl.org

Description

Bio::DB::Fasta provides indexed access to one or more Fasta files. It provides random access to each sequence entry, and to subsequences within each entry, allowing you to retrieve portions of very large sequences without bringing the entire sequence into memory. When you initialize the module, you point it at a single fasta file or a directory of multiple such files. The first time it is run, the module generates an index of the contents of the file or directory using the AnyDBM module (Berkeley DB* preferred, followed by GDBM_File, NDBM_File, and SDBM_File). Thereafter it uses the index file to find the file and offset for any requested sequence. If one of the source fasta files is updated, the module reindexes just that one file. (You can also force reindexing manually). For improved performance, the module keeps a cache of open filehandles, closing less-recently used ones when the cache is full. The fasta files may contain any combination of nucleotide and protein sequences; during indexing the module guesses the molecular type. Entries may have any line length up to 65,536 characters, and different line lengths are allowed in the same file. However, within a sequence entry, all lines must be the same length except for the last.

Bio::DB::fasta method description

doc.bioperl.org

get_Seq_by_id

code

prev

```
Title      : get_Seq_by_id
Usage      : my $seq = $db->get_Seq_by_id($id)
Function:  Bio::DB::RandomAccessI method implemented
Returns   : Bio::PrimarySeqI object
Args      : id
```

Query a local fasta file

```
#!/usr/bin/perl -w
use strict;
use Bio::DB::Fasta;

my $dbfile = 'uniprot_sprot.fasta';
my $db = Bio::DB::Fasta->new($dbfile);

# retrieve a sequence
my $id = 'sp|Q13547|HDAC1_HUMAN';
my $seq_obj = $db->get_Seq_by_id($id);

if ( $seq_obj ) {
    print "seq: ", $seq_obj->seq, "\n";
} else {
    warn("Cannot find $id\n");
}
```

output

```
seq: MAQTQGTRRKVCYYYDGDVGNYYYYGQGHMCKPHRIRMTHNLL LNYGLYRKMEIYRPHKANAE
EMTKYHSDDYIKFLRSIRPDNMSEYSKQMQRFNVGEDCPVFDGLFEFCQLSTGGSVASAVKLNKQQT
DIAVNWAGGLHHAKKSEASGFCYVNDIVLAIL ELLKYHQRVLYIDIDIHHGDGVEEAFYTTDRVMTV
SFHKYGEYFPGTGDLRDIGAGKGKYYAVNYPLRDGIDDES YEAI FKPVMSKVMEMFQPSAVVLQCGS
DSL S GDRLGCFNLTIKGHAKCVEFVKSFNLPMLMLGGGGYTIRNVARCWTYETAVALDTEIPNELPY
NDYFEYFGPDFKLHISPSNMTNQNTNEYLEKIKQRLFENLRMLPHAPGVQMQAIPEDAIP EESGDED
EDDPDKRISICSSDKRIACEEEFSDSEEEGEGGRKNSSNFKKAKRVKTEDEKEKDPEEKKEVTEEEK
TKEEKPEAKGVKEEVKLA
```

Creating a sequence record

Creating a sequence record

You have a sequence and want to create a Seq object on the fly.

Use `Bio::Seq`.

Create a sequence record on the fly.

```
#!/usr/bin/perl -w  
use strict;  
use Bio::Seq;  
use Bio::SeqIO;
```

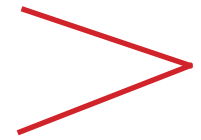
```
#file:createSeqOnFly.pl
```

```
my $seqObj = Bio::Seq->new(-seq => 'ATGAATGATGAA',  
                          -display_id => 'seq_example',  
                          -description=> 'this seq is awesome');
```



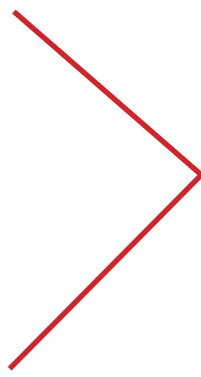
1. Create a new seq object

```
my $out = Bio::SeqIO->new(-format => 'fasta');  
$out->write_seq($seqObj);
```



2. Create and print a new seqIO object in fasta format using \$seqObj

```
print "Id: ", $seqObj->display_id, "\n";  
print "Length: ", $seqObj->length, "\n";  
print "Seq: ", $seqObj->seq, "\n";  
print "Subseq (3..6): ", $seqObj->subseq(3,6), "\n";  
print "Translation: ", $seqObj->translate->seq, "\n";
```



3. Get features of \$seqObj by using seqObj methods



Notice the coupling of methods.

Output

```
>seq_example this seq is awesome  
ATGAATGATGAA  
Id: seq_example  
Length: 12  
Seq: ATGAATGATGAA  
Subseq (3..6): GAAT  
Translation: MNDE
```

File format conversions

File format conversions



You have GenBank files and want to extract only the sequence in fasta format.

Use `Bio::SeqIO`.

Formats

BioPerl's SeqIO system understands lot of formats and can interconvert all of them. Here is a current listing of formats, as of version 1.5.

Table 1: [Bio::SeqIO](#) modules and formats supported

Name	Description	File extension	Module
abi	ABI tracefile	ab[i1]	Bio::SeqIO::abi
ace	Ace database	ace	Bio::SeqIO::ace
agave	AGAVE XML		Bio::SeqIO::agave
alf	ALF tracefile	alf	Bio::SeqIO::alf
asciitree	write-only, to visualize features		Bio::SeqIO::asciitree
bsml	BSML , using XML::DOM 	bsml	Bio::SeqIO::bsml
bsml_sax	BSML , using XML::SAX 		Bio::SeqIO::bsml_sax
chadoxml	CHADO sequence format		Bio::SeqIO::chadoxml
chaos	CHAOS sequence format		Bio::SeqIO::chaos
chaosxml	Chaos XML		Bio::SeqIO::chaosxml
ctf	CTF tracefile	ctf	Bio::SeqIO::ctf

<http://www.bioperl.org/wiki/HOWTO:SeqIO>

```

LOCUS       MUSIGHBA1               408 bp    mRNA    linear   ROD 27-APR-1993
DEFINITION  Mouse Ig active H-chain V-region from MOPC21, subgroup VH-II,
            mRNA.
ACCESSION   J00522
VERSION     J00522.1  GI:195052
KEYWORDS    constant region; immunoglobulin heavy chain; processed gene; variable re-
            gion; variable region subgroup VH-II.
SOURCE      Mus musculus (house mouse).
            ORGANISM  Mus musculus
                Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
                Mammalia; Eutheria; Euarchontoglires; Glires; Rodentia;
                Sciurognathi; Muroidea; Muridae; Murinae; Mus.
REFERENCE   1  (bases 1 to 408)
AUTHORS     Bothwell,A.L., Paskind,M., Reth,M., Imanishi-Kari,T., Rajewsky,K.
            and Baltimore,D.
TITLE       Heavy chain variable region contribution to the NPb family of
            antibodies: somatic mutation evident in a gamma 2a variable region
JOURNAL     Cell 24 (3), 625-637 (1981)
PUBMED     6788376
COMMENT     Original source text: Mouse C57Bl/6 myeloma MOPC21, cDNA to mRNA,
            clone pAB-gamma-1-4. [1] studies the response in C57Bl/6 mice to
            NP proteins. It is called the b-NP response because this mouse
            strain carries the b-IgH haplotype. See other entries for b-NP
            response for more comments.
FEATURES             Location/Qualifiers
     source             1..408
                        /db_xref="taxon:10090"
                        /mol_type="mRNA"
                        /organism="Mus musculus"
     CDS                 <1..>408
                        /db_xref="GI:195055"
                        /codon_start=1
                        /protein_id="AAD15290.1"
                        /translation="RLNLVFLVLILKGVQCDVQLVESGGGLVQPGGSRKLSAASGFT
                        FSSFGMHWVRQAPEKGLEWVAYISSGSSTLHYADTVKGRFTISRDNPKNTLFLQMTSL
                        RSEDTAMYICARWGNYPYAMDYWGQTSVTVSS"
                        /note="Ig H-chain V-region from MOPC21"
     sig_peptide         <1..48
     mat_peptide         49..>408
                        /product="Ig H-chain V-region from MOPC21 mature peptide"
     misc_recomb        343..344
                        /note="V-region end/D-region start (+/- 1bp)"
     misc_recomb        356..357
                        /note="D-region end/J-region start"
BASE COUNT      95 a      98 c      111 g      104 t
ORIGIN          57 bp upstream of PvuII site, chromosome 12.
     1  aggctcaatt tagtttctct tgtccttatt ttaaaaggtg tccagtgtga tgtgcagctg
     61  gtggagtctg ggggaggctt agtgcagcct ggaggggtccc ggaaactctc ctgtgcagcc
    121  tctggattca ctttcagtag ctttggaatg cactgggttc gtcaggctcc agagaagggg
    181  ctggagtggg tcgcatacat tagtagtggc agtagtacc tccactatgc agacacagtg
    241  aagggccgat tcaccatctc aagagacaat cccaagaaca ccctgttctc gcaaatgacc
    301  agtctaaggt ctaggacac ggccatgtat tactgtgcaa gatggggtaa ctacccttac
    361  tatgctatgg actactgggg tcaaggaacc tcagtcccg tctcctca
//

```

= GenBank Format



Fasta Format

```

>MUSIGHBA1 Mouse Ig active H-chain V-region from MOPC21,
subgroup VH-II, mRNA.
AGGTC AATTTAGTTTTTCCTTGTCCCTTATTTTTAAAAGGTGTCCAGTGTGATGTGCAGCTG
GTGGAGTCTGGGGGAGGCTTAGTGCAGCCTGGAGGGTCCCGGAAACTCCTGTGCAGCC
TCTGGATTCACTTTCAGTAGCTTTGGAATGCACTGGGTTCGTCAGGCTCCAGAGAAGGGG
CTGGAGTGGGTGCGATACATTAGTAGTGGCAGTAGTACCCTCCACTATGCAGACACAGTG
AAGGGCCGATTCACTATCTCAAGAGACAATCCCAAGAACACCCCTGTTCTGCAAAATGACC
AGTCTAAGGTCTGAGGACACGGCCATGTATTACTGTGCAAGATGGGGTAACTACCCTTAC
TATGCTATGGACTACTGGGGTCAAGGAACCTCAGTCACCGTCTCCTCA

```

Convert from GenBank to fasta.

```
#!/usr/bin/perl -w  
use strict;  
use Bio::SeqIO;
```

```
#file:convert_genbank2fasta.pl
```

```
my ($informat,$outformat) = ('genbank','fasta');  
my ($infile,$outfile) = @ARGV;
```

```
my $in = Bio::SeqIO->new(  
    -format => $informat,  
    -file => $infile,  
);
```

```
my $out = Bio::SeqIO->new(  
    -format => $outformat,  
    -file => ">$outfile"  
);
```

```
while ( my $seqObj = $in->next_seq ) {  
    $out->write_seq($seqObj);  
}
```

Retrieving annotations

Retrieving annotations

You have GenBank files and want to retrieve annotations.

Use `Bio::SeqIO`.

Sample GenBank file with Features/Annotations

LOCUS MUSIGHBA1 408 bp mRNA linear ROD 27-APR-1993
DEFINITION Mouse Ig active H-chain V-region from MOPC21, subgroup VH-II,
mRNA.
ACCESSION J00522
VERSION J00522.1 GI:195052
KEYWORDS constant region; immunoglobulin heavy chain; processed gene; variable re-
gion; variable region subgroup VH-II.
SOURCE Mus musculus (house mouse).
ORGANISM Mus musculus
Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
Mammalia; Eutheria; Euarchontoglires; Glires; Rodentia;
Sciurognathi; Muroidea; Muridae; Murinae; Mus.
REFERENCE 1 (bases 1 to 408)
AUTHORS Bothwell,A.L., Paskind,M., Reth,M., Imanishi-Kari,T., Rajewsky,K.
and Baltimore,D.
TITLE Heavy chain variable region contribution to the NPb family of
antibodies: somatic mutation evident in a gamma 2a variable region
JOURNAL Cell 24 (3), 625-637 (1981)
PUBMED 6788376
COMMENT Original source text: Mouse C57Bl/6 myeloma MOPC21, cDNA to mRNA,
clone pAB-gamma-1-4. [1] studies the response in C57Bl/6 mice to
NP proteins. It is called the b-NP response because this mouse
strain carries the b-IgH haplotype. See other entries for b-NP
response for more comments

FEATURES Location/Qualifiers
source 1..408
/db_xref="taxon:10090"
/mol_type="mRNA"
/organism="Mus musculus"
CDS <1..>408
/db_xref="GI:195055"
/codon_start=1
/protein_id="AAD15290.1"
/translation="RLNLVFLVLILKGVQCDVQLVESGGGLVQPGGSRKLSCAASGFT
FSSFGMHWVRQAPEKGLEWVAYISSGSSTLHYADTVKGRFTISRDNPKNTLFLQMTSL
RSED TAMYYCARWGNYPYAMDYWGQTSVTVSS"
/note="Ig H-chain V-region from MOPC21"
sig_peptide <1..48
mat_peptide 49..>408
/product="Ig H-chain V-region from MOPC21 mature peptide"
misc_recomb 343..344
/note="V-region end/D-region start (+/- 1bp)"
misc_recomb 356..357
/note="D-region end/J-region start"

BASE COUNT 95 a 98 c 111 g 104 t
ORIGIN 57 bp upstream of PvuII site, chromosome 12.
1 aggctcaatt tagttttcct tgtccttatt ttaaaagggtg tccagtgatg tgtgcagctg
61 gtggagctcg ggggaggctt agtgcagcct ggagggtccc gaaactctc ctgtgcagcc
121 tctggattca ctttcagtag ctttggaatg cactgggttc gtcaggctcc agagaagggg
181 ctggagtggt tgcatacat tagtagtggc agtagtacc tccactatgc agacacagtg
241 aagggccgat tcacatctc aagagacaat cccaagaaca ccctgttctc gcaaatgacc
301 agtctaaggt ctgaggacac ggccatgtat tactgtgcaa gatggggtaa ctacccttac
361 tatgctatgg actactggg tcaaggaacc tcagtcaccg tctcctca

//

FEATURES

Location/Qualifiers

source

1..408
/db_xref="taxon:10090"
/mol_type="mRNA"
/organism="Mus musculus"

CDS

<1..>408
/db_xref="GI:195055"
/codon_start=1
/protein_id="AAD15290.1"
/translation="RLNLVFLVLILKGVQCDVQLVESGGGLVQPGGSRKLSCAASGFT
FSSFGMHWVRQAPEKGLEWVAYISSGSSTLHYADTVKGRFTISRDNPKNTLFLQMTSL
RSEDAMYYCARWGNYPYYAMDYWGQGTSVTVSS"
/note="Ig H-chain V-region from MOPC21"

sig_peptide

<1..48

mat_peptide

49..>408

/product="Ig H-chain V-region from MOPC21 mature peptide"

misc_recomb

343..344

/note="V-region end/D-region start (+/- 1bp)"

misc_recomb

356..357

/note="D-region end/J-region start"



primary_tag



tag=value


Get annotations from a GenBank file

#file: get_annot_from_genbank.pl

```
#!/usr/bin/perl -w
use strict;
use Bio::SeqIO;

my $infile = shift;
my $seqIO = Bio::SeqIO->new(
    -file => $infile,
    -format => 'genbank',
);
while (my $seqObj = $seqIO -> next_seq){
    my $name = $seqObj -> id;
    foreach my $feature ($seqObj->get_SeqFeatures){
        my $primary_tag = $feature->primary_tag;
        my ($start, $end) = ($feature->start , $feature->end);
        my $range = $start . ".." . $end;
        foreach my $tag ( sort $feature->get_all_tags ) {
            my @values = $feature->get_tag_values($tag);
            my $value_str = join ",", @values;
            print "$name($range)\t$primary_tag\t$tag:$value_str\n";
        }
    }
}
```

get_SeqFeature
produces an array of
Bio::SeqFeatureI objects



Output

```
MUSIGHBA1 (1..408)      source      db_xref:taxon:10090
MUSIGHBA1 (1..408)      source      mol_type:mRNA
MUSIGHBA1 (1..408)      source      organism:Mus musculus
MUSIGHBA1 (1..408)      CDS         codon_start:1
MUSIGHBA1 (1..408)      CDS         db_xref:GI:195055
MUSIGHBA1 (1..408)      CDS         note:Ig H-chain V-region from MOPC21
MUSIGHBA1 (1..408)      CDS         protein_id:AAD15290.1
MUSIGHBA1 (1..408)      CDS         translation:RLNLVFLVLILKGVQCDVQLVESGGGLVQPGGSRKLSCAASGFTFSSF
GMHWVRQAPEKGLEWVAYISSGSSTLHYADTVKGRFTISRDNPKNTLFLQMTSLRSEDAMYYCARWGNYPYYAMDYWGQGTSVTVSS
MUSIGHBA1 (49..408)    mat_peptide product:Ig H-chain V-region from MOPC21 mature pep-
tide
MUSIGHBA1 (343..344)    misc_recomb note:V-region end/D-region start (+/- 1bp)
MUSIGHBA1 (356..357)    misc_recomb note:D-region end/J-region start
```

Manipulating Multiple Alignments

Use Bio::AlignIO

for parsing and writing multiple alignment file formats
including:

fasta, phylip, nexus, clustalw, msf, mega,
meme, pfam, psi, selex, stockholm.

Convert from fasta_aln to nexus

#file: multi_align_convert.pl

```
#!/usr/bin/perl -w  
use strict;  
use Bio::AlignIO;
```

```
my $align_fasta = shift;  
my $in_alignIO_obj = Bio::AlignIO->new(  
    -format => 'fasta',  
    -file => $align_fasta  
);
```

```
my $out_alignIO_obj = Bio::AlignIO->new(  
    -format => 'nexus',  
    -file => ">$align_fasta.nex"  
);
```

```
while( my $align_obj = $in_alignIO_obj->next_aln ){  
    $out_alignIO_obj->write_aln($align_obj);  
}
```

next_aln produces a
Bio::SimpleAlign object



Bio::SimpleAlign Object

Remove some sequences and rewrite the result

Extract or remove columns


Calculate consensus string and percent identity

Parsing BLAST Output

Parsing BLAST reports

Use `Bio::SearchIO`

Where do you start?



[howto](#) [discussion](#) [view source](#) [history](#)

HOWTO:Beginners

Contents [\[hide\]](#)

- [1 Authors](#)
- [2 Copyright](#)
- [3 Abstract](#)
- [4 Introduction](#)
- [5 Installing Bioperl](#)
- [6 Getting Assistance](#)
- [7 Perl Itself](#)
- [8 Writing a script in Unix](#)
- [9 Creating a sequence, and an Object](#)
- [10 Writing a sequence to a file](#)
- [11 Retrieving a sequence from a file](#)
- [12 Retrieving a sequence from a database](#)
- [13 Retrieving multiple sequences from a database](#)
- [14 The Sequence Object](#)
- [15 Example Sequence Objects](#)
- [16 BLAST](#)

main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#) ←
- [API Docs](#)
- [Scrapbook](#)
- [BioPerl Tutorial](#)
- [Tutorials](#)
- [Deobfuscator](#)
- [Browse Modules](#)

Here's an example of how one would use SearchIO to extract data from a BLAST report:

```
use Bio::SearchIO;
my $report_obj = new Bio::SearchIO(-format => 'blast',
                                   -file   => 'report.bls');
while( $result = $report_obj->next_result ) {
    while( $hit = $result->next_hit ) {
        while( $hsp = $hit->next_hsp ) {
            if ( $hsp->percent_identity > 75 ) {
                print "Hit\t", $hit->name, "\n", "Length\t", $hsp->length('total'),
                    "\n", "Percent_id\t", $hsp->percent_identity, "\n";
            }
        }
    }
}
```



BioPerl

main links

- [Main Page](#)
- [Getting Started](#)
- [Downloads](#)
- [Installation](#)
- [Recent changes](#)
- [Random page](#)

documentation

- [Quick Start](#)
- [FAQ](#)
- [HOWTOs](#)
- [API Docs](#)

[howto](#)

[discussion](#)

[view source](#)

[history](#)

HOWTO:SearchIO

Abstract

This is a HOWTO about the [Bio::SearchIO](#) system, how to use it, and how one goes about writing new adaptors to different output formats. We will also describe how the [Bio::SearchIO::Writer](#) modules work for outputting various formats from [Bio::Search](#) objects.

Contents [\[hide\]](#)

- 1 Abstract
 - 1.1 Authors
- 2 Background
- 3 Design
- 4 Parsing with [Bio::SearchIO](#)
 - 4.1 Avoiding possible confusion
 - 4.2 Using [SearchIO](#)

NCBI BLAST Report

Result

BLASTX 2.2.12 [Aug-07-2005]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", Nucleic Acids Res. 25:3389-3402.

Query= smed-HDAC1-1
(1213 letters)

Database: swissprot.aa
427,028 sequences; 157,875,145 total letters

Searching.....done

Hit

Sequences producing significant alignments:	Score (bits)	E Value
sp P56517 HDAC1_CHICK RecName: Full=Histone deacetylase 1; Short...	535	e-151

Result

HSP

```
>sp|P56517|HDAC1_CHICK RecName: Full=Histone deacetylase 1; Short=HD1
Length = 480

Score = 535 bits (1379), Expect = e-151
Identities = 255/343 (74%), Positives = 292/343 (85%), Gaps = 1/343 (0%)
Frame = +3

Query: 3 CPVFDGLFEFCQLSAGGSVASAVKLNKNKADIAINWSGGLHHAKKSEASGFCYVNDIVMG 182
          CPVFDGLFEFCQLSAGGSVASAVKLNK + DIA+NW+GGLHHAKKSEASGFCYVNDIV+
Sbjct: 100 CPVFDGLFEFCQLSAGGSVASAVKLNKQQTDI AVNWAGGLHHAKKSEASGFCYVNDIVLA 159

Query: 183 ILELLKYHERVLYVDIDIHHGDGVVEEAFYTTDRVMTVSFHKYGEYFPXXXXXXXXXXXXX 362
           ILELLKYH+RVLY+DIDIHHGDGVVEEAFYTTDRVMTVSFHKYGEYFP
Sbjct: 160 ILELLKYHQRVLYIDIDIHHGDGVVEEAFYTTDRVMTVSFHKYGEYFPGTGLDRDIGAGKG 219

Query: 363 XNYAVNFPLRDGIDDESYESIFKPVVEKVIESFKPNAIVLQCGADSLSGDRLGCFNLSLK 542
           YAVN+PLRDGIDDESYE+IFKPV+ KV+E+F+P+A+VLQCG+DSLSDRLGCFNL++K
Sbjct: 220 KYAVNYPLRDGIDDESYEAFKPVISKVMTFQPSAVVLQCGSDSLSGDRLGCFNLTIK 279

Query: 543 GHGKCVEMRQQPIPLLMLGGGGYTIRNVARCWTYETALALGTTIPNELPYNDYIEYFTP 722
           GH KCVE+++ +P+LMLGGGGYTIRNVARCWTYETA+AL T IPNELPYNDY+EYF P
Sbjct: 280 GHAKCVEFVKSFNLPMLMLGGGGYTIRNVARCWTYETAVALDTEIPNELPYNDYFEYFGP 339

Query: 723 DFKLHISPSNMANQNTPEYLERMKQKLFENLRSIPHAPSVQMDDIPEDAMDIDDGEQMDN 902
           DFKLHISPSNM NQNT EYLE++KQ+LFENLR +PHAP VQMQ IPEDA+ D G++ +
Sbjct: 340 DFKLHISPSNMTNQNNTNEYLEKIKQRLFENLRMLPHAPGVQMDDIPEDAVQEDSGDE-EE 398

Query: 903 ADPDKRISILASDKYREHEADLSDSEDEGD-NRKNVDCFKSKR 1028
           DP+KRISI SDK ++SDSEDEG+ RKNV FK +
Sbjct: 399 EDPEKRISIRNSDKRISCDEEFSDSEDEGEGGRKNVANFKKAK 441
```

Database: /common/data/swissprot.aa
Posted date: Oct 4, 2009 2:02 AM
Number of letters in database: 157,875,145
Number of sequences in database: 427,028

Lambda	K	H
0.318	0.134	0.401

Gapped Lambda	K	H
0.267	0.0410	0.140

Matrix: BLOSUM62
Gap Penalties: Existence: 11, Extension: 1
Number of Hits to DB: 281,587,467
Number of Sequences: 427028
Number of extensions: 5577736
Number of successful extensions: 16223
Number of sequences better than 1.0e-10: 1
Number of HSP's better than 0.0 without gapping: 15290
Number of HSP's successfully gapped in prelim test: 0
Number of HSP's that attempted gapping in prelim test: 0
Number of HSP's gapped (non-prelim): 16078
length of database: 157,875,145
effective HSP length: 119
effective length of database: 107,058,813
effective search space used: 30404702892
frameshift window, decay const: 40, 0.1
T: 12
A: 40
X1: 16 (7.3 bits)
X2: 38 (14.6 bits)
X3: 64 (24.7 bits)
S1: 41 (21.7 bits)

Bookmark it!!

See

<http://www.bioperl.org/wiki/HOWTO:SearchIO>

for a GREAT example of a blast report,

code to parse it,

a table of methods,

and the values the methods return.

Bio::SearchIO object for BLAST reports

```
#!/usr/bin/perl -w
use strict;
use Bio::SearchIO;
#file: blast_parser_intro.pl

my $blast_report = shift;

my $searchIO_obj = Bio::SearchIO->new(
    -file => $blast_report,
    -format => 'blast'
);
```

Result object and methods

#file: sample_Blast_parser_1.pl

```
#!/usr/bin/perl -w
```

```
use strict;
```

```
use Bio::SearchIO;
```

```
my $blast_report = shift;
```

```
my $searchIO_obj = Bio::SearchIO->new(  
    -file => $blast_report,  
    -format => 'blast'  
);
```

```
while (my $result_obj = $searchIO_obj ->next_result ) {  
    my $program = $result_obj ->algorithm;  
    my $queryName = $result_obj ->query_name;  
    my $queryDesc = $result_obj ->query_description;  
    my $queryLen = $result_obj ->query_length;  
    print "program=$program\tqueryName=$queryName\t";  
    print "queryDesc=$queryDesc\tqueryLen=$queryLen\n";  
}
```

Output:

```
program=BLASTX queryName=smed-HDAC1-1 queryDesc=histone deacetylase 1 queryLen=1213
```

<http://www.bioperl.org/wiki/HOWTO:SearchIO>

Object	Method	Example	Description
Result	algorithm	BLASTX	algorithm string
Result	algorithm_version	2.2.4 [Aug-26-2002]	algorithm version
Result	query_name	20521485 dbj AP004641.2	query name
Result	query_accession	AP004641.2	query accession
Result	query_length	3059	query length
Result	query_description	Oryza sativa ... 977CE9AF checksum.	query description
Result	database_name	test.fa	database name
Result	database_letters	1291	number of residues in database
Result	database_entries	5	number of database entries
Result	available_statistics	effectivespaceused ... dbletters	statistics used
Result	available_parameters	gapext matrix allowgaps gapopen	parameters used
Result	num_hits	1	number of hits
Result	hits		List of all Bio::Search::Hit::GenericHit object(s) for this Result
Result	rewind		Reset the internal iterator that dictates where <code>next_hit()</code> is pointing, useful for re-iterating through the list of hits.

Hit object and methods

```
#!/usr/bin/perl -w  
use strict;  
use Bio::SearchIO;
```

```
#file: sample_Blast_parser_2.pl
```

```
my $blast_report = shift;
```

```
my $searchIO_obj = Bio::SearchIO->new(  
    -file => $blast_report,  
    -format => 'blast'  
);
```

```
while (my $result_obj = $searchIO_obj->next_result ) {  
    while (my $hit_obj = $result_obj->next_hit){  
        my $hitName = $hit_obj->name;  
        my $hitAcc = $hit_obj->accession;  
        my $hitLen = $hit_obj->length;  
        my $hitSig = $hit_obj->significance;  
        my $hitScore = $hit_obj->raw_score;  
  
        print "hitName=$hitName\thitAcc=$hitAcc\thitLen=$hitLen\t";  
        print "hitSig=$hitSig\thitScore=$hitScore\n";  
    }  
}
```

must get hit objects
from a result object



Output:

```
hitName=sp|P56517|HDAC1_CHICK hitAcc=P56517 hitLen=480 hitSig=1e-151 hitScore=535
```

<http://www.bioperl.org/wiki/HOWTO:SearchIO>

Hit	name	443893 124775	hit name
Hit	length	331	Length of the Hit sequence
Hit	accession	443893	accession (usually when this is a genbank formatted id this will be an accession number- the part after the <i>gb</i> or <i>emb</i>)
Hit	description	LaForas sequence	hit description
Hit	algorithm	BLASTX	algorithm
Hit	raw_score	92	hit raw score
Hit	significance	2e-022	hit significance
Hit	bits	92.0	hit bits
Hit	hsps		List of all Bio::Search::HSP::GenericHSP object(s) for this Hit
Hit	num_hsps	1	number of HSPs in hit
Hit	locus	124775	locus name
Hit	accession_number	443893	accession number
Hit	rewind		Resets the internal counter for <code>next_hsp()</code> so that the iterator will begin at the beginning of the list


```
#!/usr/bin/perl -w
use strict;
use Bio::SearchIO;
```


HSP object and methods

```
#file: sample_Blast_parser.pl
```

```
my $blast_report = shift;
```

```
my $searchIO_obj = Bio::SearchIO->new(
    -file => $blast_report,
    -format => 'blast'
);
```

must get hsp objects
from a hit object



```
while (my $result_obj = $searchIO_obj->next_result ) {
    while (my $hit_obj = $result_obj->next_hit){
        while (my $hsp_obj = $hit_obj->next_hsp){
            my $evalue = $hsp_obj->evalue;
            my $hitString = $hsp_obj->hit_string;
            my $queryString = $hsp_obj->query_string;
            my $homologyString = $hsp_obj->homology_string;

            print "hsp evalue: $evalue\n";
            print "HIT      : ",substr($hitString,0,50)," \n";
            print "HOMOLOGY: ",substr($homologyString,0,50)," \n";
            print "QUERY   : ",substr($queryString,0,50)," \n";
        }
    }
}
```

Output:

```
hsp evalue: 1e-151
HIT      : CPVFDGLFEFCQLSAGGSVASAVKLNKQQTDIAVNWAGGLHHAKKSEASG
HOMOLOGY: CPVFDGLFEFCQLSAGGSVASAVKLNK + DIA+NW+GGLHHAKKSEASG
QUERY   : CPVFDGLFEFCQLSAGGSVASAVKLNKKNKADIAINWSGGLHHAKKSEASG
```

http://www.bioperl.org/wiki/HOWTO:SearchIO

HSP	algorithm	BLASTX	algorithm
HSP	evaluate	2e-022	e-value
HSP	expect	2e-022	alias for evaluate()
HSP	frac_identical	0.884615384615385	Fraction identical
HSP	frac_conserved	0.923076923076923	fraction conserved (conservative and identical replacements aka "fraction similar") (only valid for Protein alignments will be same as frac_identical)
HSP	gaps	2	number of gaps
HSP	query_string	DMGRCSSG ..	query string from alignment
HSP	hit_string	DIVQNSS ...	hit string from alignment
HSP	homology_string	D+...SSGN	string from alignment
HSP	length('total')	52	HSP seq_inds('query','conserved') (966,967,969,971,973,974,975, ...)
HSP	length('hit')	50	HSP seq_inds('hit','identical') (197,202,203,204,205, ...)
HSP	length('query')	154	HSP seq_inds('hit','conserved-not-identical') (198,200)
HSP	hsp_length	52	HSP seq_inds('hit','conserved',1) (197,202-246)
HSP	frame	0	HSP score 227
HSP	num_conserved	48	HSP bits 92.0
HSP	num_identical	46	HSP range('query') (2896,3051)
HSP	rank	1	HSP range('hit') (197,246)
HSP	seq_inds('query','identical')	(966,967,969,971,973,974,975, ...)	HSP percent_identity 88.4615384615385
HSP	seq_inds('query','conserved-not-identical')	(966,967,969,971,973,974,975, ...)	HSP strand('hit') 1
			HSP strand('query') 1
			HSP start('query') 2896
			HSP end('query') 3051
			HSP start('hit') 197
			HSP end('hit') 246
			HSP matches('hit') (46,48)
			HSP matches('query') (46,48)
			HSP get_aln <i>sequence alignment</i>
			HSP hsp_group <i>Not available in this report</i>
			HSP links <i>Not available in this report</i>

[Bio::SimpleAlign](#) object

Group field from WU-BLAST reports run with -topcombon

Links field from WU-BLAST reports run with -links showing

Other Cool Things

Whole set of wrappers for running Bioinformatics tools
in bioperl-run

Run BLAST locally or submit remote jobs (through NCBI)

Run PAML - handles setup and take down of temporary
files and directories

Run alignment progs through similar interfaces: TCoffee, MUSCLE,
Clustalw

Relational Databases for sequence and features

Repository of scripts to do really cool things. (<http://www.bioperl.org/wiki/Scripts>)

HTML

HTML

- HyperText Markup Language
- Not a programming language
- Stored in text files (just like Perl)

A basic page

```
<html>  
  
  <head>  
    <title>My web page title</title>  
  </head>  
  
  <body>  
  
    Your HTML content here  
  
  </body>  
</html>
```



A kosher page

```
<?xml version="1.0" encoding="utf-8"?>  
  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">  
  
<head>  
    <title>An XHTML 1.0 Strict standard template</title>  
</head>  
  
<body>  
    <p>... Your HTML content here ...</p>  
  
</body>  
</html>
```

Why use web standards?

- Accessibility
 - To robots
 - To people
- Stability

<Tags />

- Most tags open and close
- Tags must be nested properly

Right

```
<strong>
  <em>
    Strong and emphasis
  </em>
</strong>
```

Wrong

```
<strong>
  <em>
    Strong and emphasis
  </strong>
</em>
```

- Some tags stand alone

```
<br /> <hr />
```

- Some tags take attributes

```

```

```
<a href="theonion.com">The Onion</a>
```

- Elements consist of start and end tags flanking content

XHTML tags

<!-->	<!DOCTYPE>	<a>	<abbr>	<acronym>	<address>	<area />		<base />	<bdo>
<big>	<blockquote>	<body>	 	<button>	<caption>	<cite>	<code>	<col />	<colgroup>
<dd>		<dfn>	<div>	<dl>	<dt>		<fieldset>	<form>	<frame />
<frameset>	<head>	<h1> - <h6>	<hr />	<html>	<i>	<iframe>		<input />	<ins>
<kbd>	<label>	<legend>		<link />	<map>	<meta />	<noframes>	<noscript>	<object>
	<optgroup>	<option>	<p>	<param />	<pre>	<q>	<samp>	<script>	<select>
<small>			<style>	<sub>	<sup>	<table>	<tbody>	<td>	<textarea>
<tfoot>	<th>	<thead>	<title>	<tr>	<tt>		<var>		

<http://www.w3schools.com/tags/>

Text tags

- Heading tag

```
<h1>This is a top level heading</h1>  
<h6>This is the bottom level heading</h6>
```

- Paragraph tag

```
<p>This is definitely a paragraph</p>
```

- Line break

```
This is just two lines<br />  
With a hard break
```

- Emphasis and Strong

```
That's <em>exactly</em> what I mean - I am <strong>sick</strong> of  
this slide
```

- Comment Tag

```
<!-- This is a comment. You won't see this on the web-->
```

Tables

```
<table border="1">
  <tr>
    <th>Column 1 heading</th>
    <th>Column 2 heading</th>
    <th>Column 3 heading</th>
  </tr>
  <tr>
    <td>Row 2, cell 1</td>
    <td colspan="2">Row 2, cell 2, also spanning Row 2, cell 3</td>
  </tr>
  <tr>
    <td rowspan="2">Row 3, cell 1, also spanning Row 4, cell 1</td>
    <td>Row 3, cell 2</td>
    <td>Row 3, cell 3</td>
  </tr>
  <tr>
    <td>Row 4, cell 2</td>
    <td>Row 4, cell 3</td>
  </tr>
</table>
```

output:

Column 1 heading	Column 2 heading	Column 3 heading
Row 2, cell 1	Row 2, cell 2, also spanning Row 2, cell 3	
Row 3, cell 1, also spanning Row 4, cell 1	Row 3, cell 2	Row 3, cell 3
	Row 4, cell 2	Row 4, cell 3

Lists

```
<ol>
  <li>First things first</li>
  <ul>
    <li>Who you know</li>
  </ul>
  <li>Not</li>
  <ul>
    <li>What you know</li>
    <li>What you can do with it</li>
  </ul>
</ol>
```

output:

1. First things first
 - Who you know
2. Not
 - What you know
 - What you can do with it

Links

- Relative

```
<a href="myDirectory/index.html">Go down a directory</a>
```

```
<a href="../index.html">Go up a directory</a>
```

- Absolute

```
<a href="/">Go to the root</a>
```

```
<a href="http://nytimes.com">Go to the NY Times</a>
```

- Anchors

```
<a href="#theEnd">Go to the end</a>
```

```
<h1 id="theEnd">This is the end</h1>
```

Images

```

```



Forms

```
<form name="input" action="html_form_submit.pl" method="post">
```

- **POST vs GET**

Text fields

```
<form name="input" action="handleMyForm.pl" method="get">  
  First name:  
  <input type="text" name="firstname" />  
  <br />  
  Last name:  
  <input type="text" name="lastname" />  
  <input type="submit" value="Submit" />  
</form>
```

output:

First name:

Last name:

Radio buttons

```
<form name="input" action="handleMyForm.pl" method="get">  
  <input type="radio" name="sex" value="male" /> Male  
  <br />  
  <input type="radio" name="sex" value="female" /> Female  
  <br />  
  <input type="submit" value="Submit" />  
</form>
```

output:

Male
 Female

xHTML + CSS = Web

```
<body>
<div id="wrap">
  <div id="header"><h1>Simple 2 column CSS layout, final layout</h1></div>
  <div id="nav">
    <ul>
      <li><a href="#">Option 1</a></li>
      <li><a href="#">Option 2</a></li>
      <li><a href="#">Option 3</a></li>

      <li><a href="#">Option 4</a></li>
      <li><a href="#">Option 5</a></li>
    </ul>
  </div>
  <div id="main">
    <h2>Column 1</h2>
    <p><a href="#">456 Berea Street Home</a></p>

    <p><a href="/lab/developing_with_web_standards/csslayout/2-col/">Simple 2 column CSS layout</a>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris vel magna. Mauris risus
    <p>Nulla a lacus. Nulla facilisi. Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    <p>Aenean tempor. Mauris tortor quam, elementum eu, convallis a, semper quis, purus. Cras at
    <h3>Consectetur adipiscing elit</h3>
    <p>Nulla dictum. Praesent turpis libero, pretium in, pretium ac, malesuada sed, ligula. Sed

    <p>Maecenas eu velit nec magna venenatis consequat. In neque. Vivamus pellentesque, lacus eu
  </div>
  <div id="sidebar">
    <h2>Column 2</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris vel magna.</p>
    <ul>
      <li><a href="#">Link 1</a></li>

      <li><a href="#">Link 2</a></li>
      <li><a href="#">Link 3</a></li>
      <li><a href="#">Link 4</a></li>
      <li><a href="#">Link 5</a></li>
      <li><a href="#">Link 6</a></li>
      <li><a href="#">Link 7</a></li>

      <li><a href="#">Link 8</a></li>
      <li><a href="#">Link 9</a></li>
      <li><a href="#">Link 10</a></li>
      <li><a href="#">Link 11</a></li>
      <li><a href="#">Link 12</a></li>
      <li><a href="#">Link 13</a></li>

      <li><a href="#">Link 14</a></li>
      <li><a href="#">Link 15</a></li>
    </ul>
  </div>
  <div id="footer">
    <p>Footer</p>
  </div>
</div>
</body>
```

+

```
<style type="text/css">
body,html {
  margin:0;
  padding:0;
  color:#000;
  background:#a7a09a;
}
#wrap {
  width:750px;
  margin:0 auto;
  background:#99c;
}
#header {
  padding:5px 10px;
  background:#ddd;
}
h1 {
  margin:0;
}
#nav {
  padding:5px 10px;
  background:#c99;
}
#nav ul {
  margin:0;
  padding:0;
  list-style:none;
}
#nav li {
  display:inline;
  margin:0;
  padding:0;
}
#main {
  float:left;
  width:480px;
  padding:10px;
  background:#9c9;
}
h2 {
  margin:0 0 1em;
}
#sidebar {
  float:right;
  width:230px;
  padding:10px;
  background:#99c;
}
#footer {
  clear:both;
  padding:5px 10px;
  background:#cc9;
}
#footer p {
  margin:0;
}
* html #footer {
  height:1px;
}
</style>
```

=

Simple 2 column CSS layout, final layout

[Option 1](#) [Option 2](#) [Option 3](#) [Option 4](#) [Option 5](#)

Column 1

[456 Berea Street Home](#)

[Simple 2 column CSS layout](#)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris vel magna. Mauris risus nunc, tristique varius, gravida in, lacinia vel, elit. Nam ornare, felis non faucibus molestie, nulla augue adipiscing mauris, a nonummy diam ligula ut risus. Praesent varius. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

Nulla a lacus. Nulla facilisi. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce pulvinar lobortis purus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec semper ipsum et urna. Ut consequat neque vitae felis. Suspendisse dapibus, magna quis pulvinar laoreet, dolor neque lacinia arcu, et luctus mi erat vestibulum sem. Mauris faucibus iaculis lacus. Aliquam nec ante in quam sollicitudin congue. Quisque congue egestas elit. Quisque viverra. Donec feugiat elementum est. Etiam vel lorem.

Aenean tempor. Mauris tortor quam, elementum eu, convallis a, semper quis, purus. Cras at tortor in purus tincidunt tristique. In hac habitasse platea dictumst. Ut eu lectus eu metus molestie iaculis. In ornare. Donec at enim vel erat tempor congue. Nullam varius. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla feugiat hendrerit risus. Integer enim velit, gravida id, sollicitudin at, consequat sit amet, leo. Fusce imperdiet condimentum velit. Phasellus nonummy interdum est. Pellentesque quam.

Consectetur adipiscing elit

Nulla dictum. Praesent turpis libero, pretium in, pretium ac, malesuada sed, ligula. Sed a urna eu ipsum luctus faucibus. Curabitur fringilla. Suspendisse sit amet quam. Sed vel pede id libero luctus fermentum. Vestibulum volutpat tempor lectus. Vivamus convallis tempus ante. Nullam adipiscing dui blandit ipsum. Nullam convallis lacus ut quam. Mauris semper elit at ante. Vivamus tristique. Pellentesque pharetra ante at pede. In ultrices arcu vitae purus. In rutrum, erat at mollis consequat, leo massa consequat libero, ac vestibulum libero tellus sed risus. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Maecenas eu velit nec magna venenatis consequat. In neque. Vivamus pellentesque, lacus eu aliquet semper, lorem metus rhoncus metus, a ornare orci ante a diam. Nunc sem nisl, aliquet quis, elementum nec, imperdiet in, wisi. Proin in lorem. Etiam molestie diam eget ante. Morbi quis tortor id lacus mollis venenatis. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam vel orci sit amet tellus mollis eleifend. Donec urna diam, viverra eget, ultricies gravida, ultrices eu, erat. Proin vel arcu. Sed diam. Cras hendrerit arcu sed augue. Sed justo felis, luctus a, accumsan in, tincidunt facilisis, libero. Phasellus eu eros.

Footer

Column 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris vel magna.

- [Link 1](#)
- [Link 2](#)
- [Link 3](#)
- [Link 4](#)
- [Link 5](#)
- [Link 6](#)
- [Link 7](#)
- [Link 8](#)
- [Link 9](#)
- [Link 10](#)
- [Link 11](#)
- [Link 12](#)
- [Link 13](#)
- [Link 14](#)
- [Link 15](#)

Cascading Style Sheets

- Help separate **content** from **appearance**
- One style sheet can be applied to hundreds of web pages
- Change styles in just one location

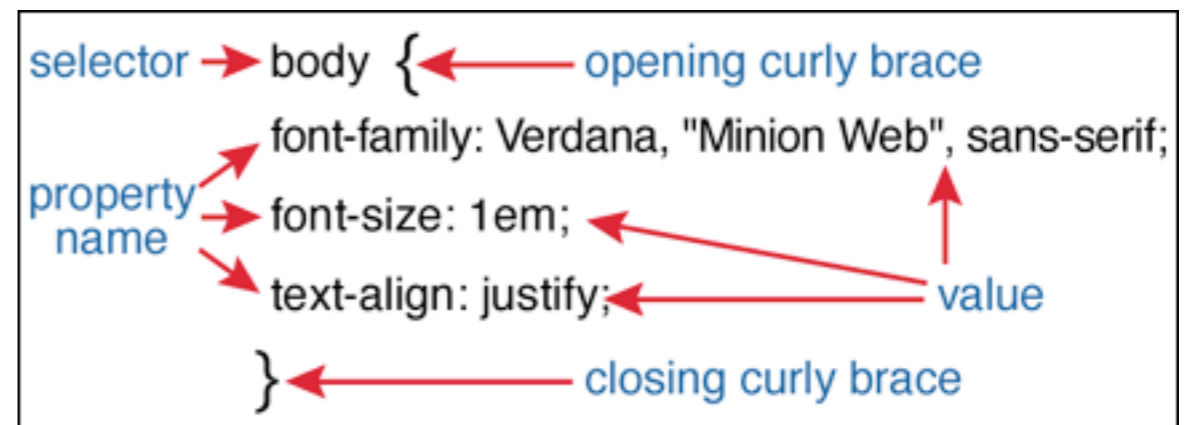
How CSS works

- Statements consist of

- Selectors

- Declarations

- Properties: Values (units)



CSS: Where do I put it?

- Embedded in the `<head>` of each page

```
<head><style type="text/css"> </style></head>
```

- Linked in the `<head>`

Advantages: templating, speed

```
<link rel="stylesheet" type="text/css"
href="/styles/style.css" />
```

- Inline (avoid this)

```
<p style="color: red">text</p>
```

CSS Selectors

- HTML selectors - raw tags in the style sheet)
- Class selectors
 - use `.className` in style sheet
 - use `class="className"` in HTML
- ID selectors
 - use `#idName` in style sheet
 - use `id="idName"` in HTML

Divs and Spans

- Divs
 - Use `<div id="myDiv"> </div>` to define block elements. Useful for both formatting and positioning.
 - The id is unique. It refers to one element
- Spans
 - Use when you want to apply a class to some text inline
 - This is my sequence
`ACTGATCTAGCT`

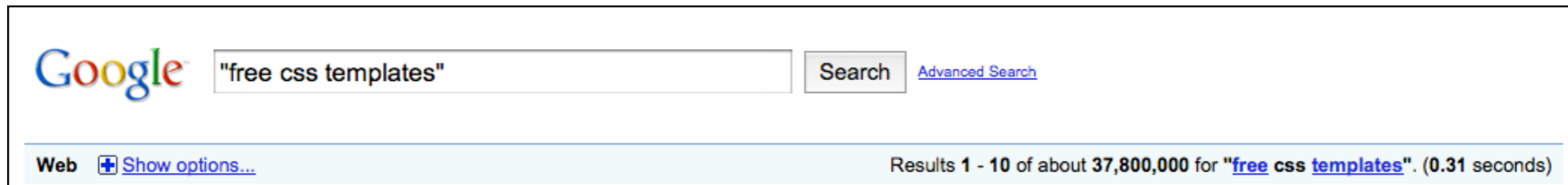
BlueprintCSS

- CSS framework
- grid
- “sensible typography”
- stylesheet for printing

Blueprint Tests: grid.css

The screenshot displays several grid layouts generated by the Blueprint CSS framework. The top section shows three columns of text. Below that, there are two rows of three columns each. The middle section features two columns of text with a longer paragraph in the left column. The bottom section shows a grid with 24 columns and 6 rows, with numbers 1 through 24 placed in various cells to illustrate the grid structure.

Do Not Reinvent the Wheel



A screenshot of a Google search interface. The search bar contains the text "free css templates". To the right of the search bar is a "Search" button and a link for "Advanced Search". Below the search bar, on the left, is the word "Web" followed by a plus sign and a link "Show options...". On the right, it displays "Results 1 - 10 of about 37,800,000 for 'free css templates'. (0.31 seconds)".

Results 1 - 10 of about 317,000 for "two column css". (0.40 seconds)

- <http://www.freecsstemplates.org>

Where does my website go?

- On Mac OS X
 - Personal web: `~/Sites`
 - Main web: `/Library/Webserver/Documents`
- Linux: `/var/www/html` or `/var/apache2/htdocs`
- XP Home: `C:\Program Files\ApacheGroup\Apache\htdocs`
- Could be elsewhere. Don't give up!

Naming your html files

- .html .htm
- Why index.html is special

Resource: HTML

- HTML Dog
<http://htmldog.com>



- W3C tags
<http://www.w3schools.com/tags>



Resources: CSS

Selectors	Box Model	Boxes
<ul style="list-style-type: none"> * All elements div <div> div * All elements within <div> div span within <div> div, span <div> and div > span with parent <div> div + span preceded by <div> .class Elements of class "class" div.class <div> of class "class" #itemid Element with id "itemid" div#itemid <div> with id "itemid" a[attr] <a> with attribute "attr" a[attr='x'] <a> when "attr" is "x" a[class~='x'] <a> when class is a list containing 'x' a[lang]='en' <a> when lang begins "en" 	<p>The diagram illustrates the CSS Box Model. It shows a central 'Visible Area' (content box) surrounded by 'Padding', then a 'Border', and finally an outer 'Margin'. Labels indicate 'Height' and 'Width' for the content box, and 'Border', 'Padding', and 'Margin' for the surrounding layers.</p>	<ul style="list-style-type: none"> margin x border-color x margin-top border-top-color margin-right border-right-color margin-bottom border-bottom-color margin-left border-left-color padding x border-style x padding-top border-top-style padding-right border-right-style padding-bottom border-bottom-style padding-left border-left-style border x border-width x border-top x border-top-width border-bottom x border-right-width border-right x border-bottom-width border-left x border-left-width
Pseudo-Selectors and Pseudo-Classes	Positioning	Tables
<ul style="list-style-type: none"> :first-child First child element :first-line First line of element :first-letter First letter of element :hover Element with mouse over :active Active element :focus Element with focus :link Unvisited links :visited Visited links :lang(var) Element with language "var" :before Before element :after After element 	<ul style="list-style-type: none"> display clear position z-index top direction + right unicode-bidi bottom overflow left clip float visibility 	<ul style="list-style-type: none"> caption-side + border-spacing + table-layout empty-cells + border-collapse + speak-header +
Sizes and Colours	Dimensions	Paging
<ul style="list-style-type: none"> 0 0 requires no unit Relative Sizes em 1em equal to font size of parent (same as 100%) ex Height of lower case "x" % Percentage Absolute Sizes px Pixels cm Centimeters mm Millimeters in Inches pt 1pt = 1/72in pc 1pc = 12pt Colours #789abc RGB Hex Notation #acf Equates to "#aacfff" rgb(0,25,50) Value of each of red, green, and blue. 0 to 255, may be swapped for percentages. 	<ul style="list-style-type: none"> width min-height min-width max-height max-width vertical-align height 	<ul style="list-style-type: none"> size page-break-inside + marks page + page-break-before orphans + page-break-after widows +
Color / Background	Text	Interface
<ul style="list-style-type: none"> color + background-repeat background x background-image background-color background-position background-attachment 	<ul style="list-style-type: none"> text-indent + word-spacing + text-align + text-transform + text-decoration white-space + text-shadow line-height + letter-spacing + 	<ul style="list-style-type: none"> cursor + outline-style outline x outline-color outline-width
Fonts	Aural	Miscellaneous
<ul style="list-style-type: none"> font + x font-weight + font-family + font-stretch + font-style + font-size + font-variant + font-size-adjust + 	<ul style="list-style-type: none"> volume + elevation speak + speech-rate pause x voice-family pause-before pitch pause-after pitch-range cue x stress cue-before richness cue-after speak-punctuation play-during speak-numeral azimuth + 	<ul style="list-style-type: none"> content list-style-type + quotes + list-style-image + counter-reset list-style-position + counter-increment marker-offset list-style + x
Note Shorthand properties are marked x Properties that inherit are marked +	Available free from www.AddedBytes.com	

Cheat sheet:

<http://www.addedbytes.com/download/css-cheat-sheet-v2/pdf/>

CSS tutorial

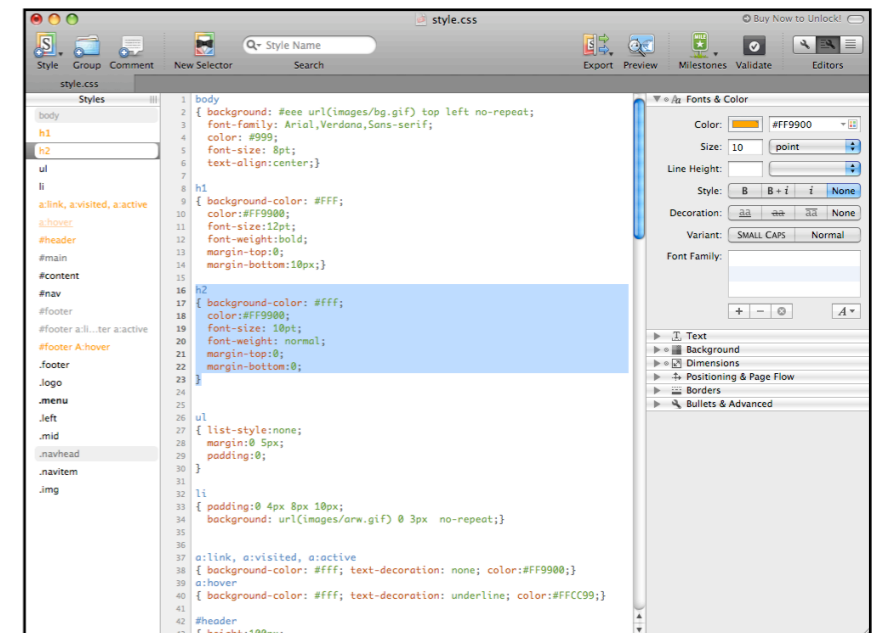
http://westciv.com/wiki/Main_Page

Two column style sheet and tutorial

http://www.456bereastreet.com/lab/developing_with_web_standards/csslayout/2-col/

Tools of the Trade

- Web Developer Plugin for Firefox
- CSS editors
 - MacRabbit CSSEdit
 - SimpleCSS
 - TopStyle (Windows)



Web programming with CGI.pm

Sunday, October 23, 2011

1

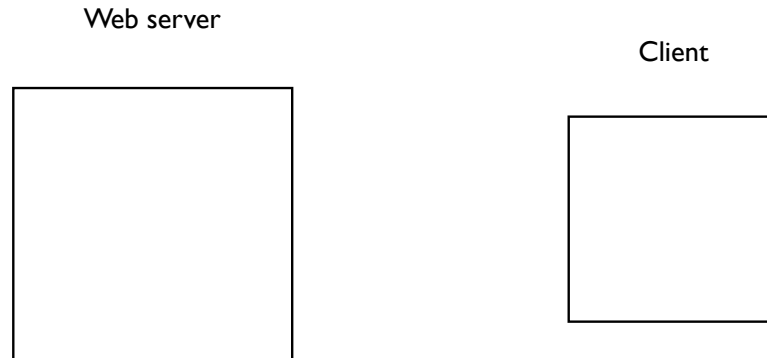
Server-Client Architecture and CGI

- Wikipedia summary: The Common Gateway Interface (CGI) is a standard method for web servers to delegate the generation of web pages to executable files. Such files are known as CGI scripts; they are programs, often stand-alone applications, usually written in a scripting language.
- Until now, you have run scripts from the command line.
 - Your scripts are somewhere like your home directory or `~/perl/` etc and the output is printed on the screen or in a file
- In web programming, scripts go somewhere like `Public/username/cgi-bin/` on a web server
 - The output of scripts is HTML and is sent to a browser running on a client machine where it is rendered as a web page.
 - This set up allows you to create dynamic web pages that are generated in response to user input e.g. if the user enters a search query on a form on a web page, the search terms are sent to a CGI script which runs on the web server and returns the results of the search as HTML which is then displayed in the web browser exactly as if it was a web page.

Sunday, October 23, 2011

2

This is better explained as a diagram, which we will draw together



Sunday, October 23, 2011

3

Setting up and executing CGI scripts

- Here's how you set up CGI scripts on our computers
 - In Finder, use Connect to Server... (command - k)
 - Select Public
 - Navigate to the directory with `your_username/cgi-bin/`
 - Save your CGI scripts in this directory.
 - This directory has to be executable by 'other'. You can use `chmod +755 <dirname>` to do this.
 - Your web scripts also have to be executable by 'other'. You can do this with `chmod +755 myscript.pl`

Sunday, October 23, 2011

4

A CGI Script that Creates Plain Text

```
#!/usr/bin/perl
# file: plaintext.pl

print "Content-type: text/plain\n\n";

print "When that Aprill with his shoures soote\n";
print "The droghte of March hath perced to the roote,\n";
print "And bathed every veyne in swich licour\n";
print "Of which vertu engendered is the flour...\n";
```

<http://mckay.cshl.edu/cgi-bin/course/plaintext.pl>

A CGI Script that Creates HTML

```
#!/usr/bin/perl
# file: chaucer.pl

print "Content-type: text/html\n\n";

print "<html><head><title>Chaucer</title></head><body>\n";
print "<h1>Chaucer Sez</h1>\n";

print "When that Aprill with his shoures soote<br>\n";
print "The droghte of March hath perced to the roote,<br>\n";
print "And bathed every veyne in swich licour<br>\n";
print "Of which vertu engendered is the flour...<p>\n";

print "<cite>-Geoffrey Chaucer</cite>\n";
print "<hr>\n";
print "</body></html>\n";
```

<http://mckay.cshl.edu/cgi-bin/course/chaucer.pl>

A CGI Script that Does Something Useful

A CGI script can do anything a Perl script can do, such as opening files and processing them. Just print your results to STDOUT.

```
#!/usr/bin/perl -w
# file: process_cosmids.pl
use strict;

my @GENES = qw/act-1 dpy-5 unc-13 let-653 skn-1 C02D5.1/;
my $URL = 'http://www.wormbase.org/db/gene/gene?name=';

print "Content-type: text/html\n\n";
print "<html><head><title>Genes</title></head><body>\n";
print "<h1>Genes</h1>\n";
print "<ol>\n";

for my $gene (@GENES) {
    print qq(<li><a href="$URL$gene">$gene</a>\n);
}

print "</ol>\n";
print "</body></html>\n";
```

http://mckay.cshl.edu/cgi-bin/course/process_genes.pl

Creating Fill-Out Forms

HTML includes about a half-dozen elements for creating fill-out form elements. A form must begin with <FORM> and end with </FORM>:

Code:

```
<form action="http://stein.cshl.org/cgi-bin/test-cgi.pl" method="POST">
  Choose a Motif:
  <input type="text" name="motif" value="TATTAT"> <br>
  <input type="submit" name="search" value="Search!"> <br>
</form>
```

Result:

Choose a Motif:

Creating Fill-Out Forms II

The <FORM> Tag

Attributes:

action (required)

CGI script to submit contents of form to.

method (required)

Submission method. Depends on CGI script. One of:

- POST
- GET

encoding

Required by certain scripts that accept file uploads. One of:

- application/x-www-form-urlencoded
 - multipart/form-data
-

Creating Fill-Out Forms III

<INPUT> Elements

Used for text fields, buttons, checkboxes, radiobuttons. Attributes:

type

Type of the field. Options:

- submit
- radio
- checkbox
- text
- password
- hidden
- file

name

Name of the field.

value

Starting value of the field. Also used as label for buttons.

size

Length of text fields.

checked

Whether checkbox/radio button is checked.

Creating Fill-Out Forms IV

Examples:

<code><input type="text" name="motif1" value="TATTAT"></code>	<input type="text" value="TATTAT"/>
<code><input type="checkbox" name="motif2" value="TATTAT"></code>	<input type="checkbox"/>
<code><input type="radio" name="motif3" value="TATTAT" checked></code> <code><input type="radio" name="motif3" value="GGGGGG"></code>	<input checked="" type="radio"/> <input type="radio"/>
<code><input type="hidden" name="settings" value="PRIVACY MODE ON"></code>	
<code><input type="submit" name="search" value="SEARCH!"></code>	<input type="submit" value="SEARCH!"/>

Creating Fill-Out Forms V

<SELECT> Element

Used to create selection lists.

Attributes:

name
Name the field.

size
Number of options to show simultaneously.

multiple
Allow multiple options to be shown simultaneously.

<OPTION> Element

Contained within a >SELECT< element. Defines an option:

```
>option>I am an option</option>
```

Attributes:

selected
Whether option is selected by default.

value
Give the option a value different from the one displayed.

Creating Fill-Out Forms VI

```

<select name="motif1">
  <option>GATTAA</option>
  <option selected>GGGTTTTC</option>
  <option>TTTTAAAA</option>
  <option>TATATATAT</option>
  <option value="Tricky!">GGCCGGTTA</option>
</select>

```

GGGTTTTC

```

<select name="motif2" size=6>
  <option>GATTAA</option>
  <option selected>GGGTTTTC</option>
  <option>TTTTAAAA</option>
  <option>TATATATAT</option>
  <option>GGCCGGTTA</option>
</select>

```

GATTAA
GGGTTTTC
TTTTAAAA
TATATATAT
GGCCGGTTA

```

<select name="motif3" size=6 multiple>
  <option>GATTAA</option>
  <option selected>GGGTTTTC</option>
  <option>TTTTAAAA</option>
  <option>TATATATAT</option>
  <option>GGCCGGTTA</option>
</select>

```

GATTAA
GGGTTTTC
TTTTAAAA
TATATATAT
GGCCGGTTA

```

<input type="submit" name="search" value="SEARCH!">

```

SEARCH!

Creating Fill-Out Forms VII

<TEXTAREA> Elements

Used to create big text elements.

Attributes:

name
name of field

rows
rows of text

cols
columns of text

wrap
type of word wrapping

```

<textarea name="sequence" rows=10 cols=30>
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
</textarea>

```

NN
NN
NN
NN

```

<input type="submit" name="search" value="SEARCH!">

```

SEARCH!

What is CGI.pm?

1. Standard module in Perl distribution (≥ 5.004)
2. Emits correct HTTP headers
3. HTML shortcuts
4. Parses CGI parameters
5. "Sticky" form fields
6. Creates & processes cookies
7. File uploads

Make HTML Beautiful

CGI.pm defines functions that emit HTML. The page is easier to read and write than raw HTML*

```
<h1>
  Eat Your Vegetables
</h1>
<ol>
  <li>peas</li>
  <li>broccoli</li>
  <li>cabbage</li>
  <li>
    peppers
    <ul>
      <li>red</li>
      <li>yellow</li>
      <li>green</li>
    </ul>
  </li>
</ol>
<hr>
```

```
#!/usr/bin/perl
# Script: vegetables1.pl

use CGI ':standard';

print header,
  start_html('Vegetables'),
  h1('Eat Your Vegetables'),
  ol(
    li('peas'),
    li('broccoli'),
    li('cabbage'),
    li('peppers',
      ul(
        li('red'),
        li('yellow'),
        li('green')
      )
    ),
  ),
  hr,
  end_html;
```

* if you speak Perl!

<http://mckay.cshl.edu/cgi-bin/course/vegetables.pl>

Make HTML Concise

Tag Functions are Distributive

```
print li('hi','how','are','you')

<LI>hi how are you</LI>

@items=('hi','how','are','you'); print li(\@items)

<LI>hi</LI>
<LI>how</LI>
<LI>are</LI>
<LI>you</LI>

print li(['hi','how','are','you'])

<LI>hi</LI>
<LI>how</LI>
<LI>are</LI>
<LI>you</LI>
```

Add HTML Attributes Using Hash References

```
%opts=(-type=>'square'); @items=('hi','how','are','you'); print li(\%opts,\@items)

<LI TYPE="square">hi</LI>
<LI TYPE="square">how</LI>
<LI TYPE="square">are</LI>
<LI TYPE="square">you</LI>

print li({-type=>'square'},['hi','how','are','you'])

<LI TYPE="square">hi</LI>
<LI TYPE="square">how</LI>
<LI TYPE="square">are</LI>
<LI TYPE="square">you</LI>
```



```
#!/usr/bin/perl
# Script: vegetables2.pl

use CGI ':standard';

print header,
  start_html("Vegetables"),
  h1("Eat Your Vegetables"),
  ol(
    li(['peas',
        'broccoli',
        'cabbage',
        'peppers' .
        ul(['red','yellow','green']),
        'kolrabi',
        'radishes'])
    ),
  hr,
  end_html;
```

<http://mckay.cshl.edu/cgi-bin/course/vegetables2.pl>

Using CGI.pm for the Genes Script

```
#!/usr/bin/perl -w
# file: process_genes2.pl

use strict;
use CGI ':standard';

my @GENES = qw/act-1 dpy-5 unc-13 let-653 skn-1 C02D5.1/;
my $URL = 'http://www.wormbase.org/db/gene/gene?name=';

my @list_items;
for my $gene (@GENES) {
  push @list_items, a({-href=>"$URL$gene"}, $gene);
}

print header(),
  start_html('Genes'),
  h1('Genes'),
  ol(
    li(\@list_items)
  ),
  end_html;
```

http://mckay.cshl.edu/cgi-bin/course/process_genes2.pl

Setting & Retrieving CGI Parameters

You can set and retrieve CGI parameters easily. In these examples, the CGI field string is:

```
banana=yellow&squash=green&tomato=red&tomato=green
```

Retrieve a Single-Valued Field Named "Tomato" :

Call `param()` with the name of the field and assign it to a scalar.

```
my $banana_color = param('banana');
# yields "yellow"
```

This works for both GET and POST types, including *multipart/form-data*.

Retrieve a Multi-Valued Field Named "Tomatoes" :

Call `param()` with the name of the field and assign it to an array:

```
my @tomatoes = param('tomato');
# yields ('red','green')
```

Finding out What Parameters are Available

Call `param()` without any arguments. Will return the names of each of the parameters:

```
my @fields = param;
# yields ('banana','squash','tomato')
```

Setting Single- and Multivalued Fields:

```
param(-name=>'tomato', -value=>'red');
param(-name=>'tomatoes',-value=>['red','green','blue']);
```

Parameters set in this way will be used as default values for fill-out form fields and hidden fields.

```
#!/usr/bin/perl
# file: final_exam.pl

use CGI 'standard';

print header;
print start_html("Your Final Exam"),
      h1("Your Final Exam"),
      start_form,
      "What's your name? ",textfield(-name=>'first_name'),
      p,
      "What's the combination?",
      p,
      checkbox_group(-name => 'words',
                    -values => ['eenie','meenie','minie','moe'],
                    -defaults => ['eenie','minie']),
      p,
      "What's your favorite color? ",
      popup_menu(-name => 'color',
                -values => ['red','green','blue','chartreuse']),
      p,
      submit,
      end_form,
      hr;

if (param()) {
    print
      "Your name is: ".param('first_name'),
      p,
      "The keywords are: ".join(" ", .param('words')),
      p,
      "Your favorite color is: ".param('color'),
      hr;
}

print end_html;
```

A Simple Form

Your Final Exam

What's your name?

What's the combination?

eenie meenie minie moe

What's your favorite color?

Your name is: Sheldon

The keywords are: eenie, minie

Your favorite color is: red

Form Generating Functions I

Run *perldoc CGI* for details:

start_form

Start the form (returns the <form> tag with default attributes).

end_form

End the form by returning the </form> tag.

textfield(-name=>\$name,-value=>\$starting_value)

Create a text field.

password_field(-name=>\$name,-value=>\$starting_value)

Create a password field.

textarea(-name=>\$name,-value=>\$starting_value,-rows=>\$rows,-cols=>\$cols)

Create a multiline text input area.

checkbox(-name=>\$name,-value=>\$value,-checked=>1)

Create a single checkbox.

checkbox_group(-name=>\$name,-value=>\@values,-default=>\@on)

Create a group of checkboxes sharing the same name. @values gives the list of checkbox values, and @on gives the list of those that are initially on.

Form Generating Functions II

radio_group(-name=>\$name,-value=>\@values,-default=>\$on)
 Create a group of radio buttons sharing the same name. @values gives the list of radio values, and \$on indicates which one is on to start with.

popup_menu(-name=>\$name,-value=>\@values,-default=>\$on)
 Create a popup menu. @values gives the list of items, and \$on indicates which one is initially selected.

scrolling_list(-name=>\$name,-value=>\@values,-default=>\$on)
 Create a scrolling list. @values gives the list of items, and \$on indicates which one (if any) is initially selected.

submit(-name=>\$name,-value=>\$value)
 Creates a submit button. \$value optionally sets the button label.

```
#!/usr/bin/perl
# file: reversec.pl

use CGI 'standard';

print header;
print start_html('Reverse Complementation'),
  h1('Reverse Complementator'),
  start_form,
  "Enter your sequence here:",br,
  textarea(-name=>'sequence',-rows=>5,-cols=>60),
  submit('Reverse Complement'),
  end_form,
  hr;

if ( param ) {
  my $sequence = param('sequence');

  my $reversec = do_reverse($sequence);
  $reversec =~ s/(.{60})/$1\n/g; # do word wrap

  print h2('Reverse complement');
  print pre($reversec);
}

print end_html;

sub do_reverse {
  my $seq = shift;
  $seq =~ s/\s//g;      # strip whitespace
  $seq =~ tr/gatcGATC/ctagCTAG/; # complement
  $seq = reverse $seq;  # and reverse
  return $seq;
}
```

A reverse complementation script

Reverse Complementator

Enter your sequence here:

AAAAAAGAGATTGTGCTTCTCATCTCTCTC

Reverse Complement

Reverse complement

GAGAGAGATGAGAAGCACAAATCTTTTTT

File Uploading

HTML: `<INPUT TYPE="FILE">` CGI.pm: `filefield()`

Annoying complication:

You have to start the form with `start_multipart_form()` rather than `start_form()`.

Let's modify `reversec.pl` to support file uploads:

- First part (script too big for one page), print the form

```
#!/usr/bin/perl
# file: sequpload.pl

use CGI ':standard';

print header;
print start_html('Reverse Complementation'),
      h1('Reverse Complementator'),
      start_multipart_form,
      "Enter your sequence here:".br,
      textarea(-name=>'sequence' ,-rows=>5,-cols=>60),br,
      'Or upload a sequence here: 'filefield(-name=>'uploaded_sequence'),
      submit('Reverse Complement'),
      end_form,
      hr;
```

sequpload.pl continued...

```
if ( param ) {
    my $sequence;

    # look for the uploaded sequence first...
    if ( my $upload = param('uploaded_sequence') ) {
        print h2("Reverse complement of $upload");

        while (my $line = <$upload>) {
            chomp $line;
            next unless $line =~ /^[gatcnGATCN]/;
            $sequence .= $line;
        }

    } else { # ... not found, so read it from the text field
        print h2('Reverse complement');
        $sequence = param('sequence');
    }

    $reversec = do_reverse($sequence);
    $reversec =~ s/(.{60})/$1\n/g; # do word wrap
    print pre($reversec);
}

print end_html;

sub do_reverse {
    my $seq = shift;
    $seq =~ s/\s//g;          # strip whitespace
    $seq =~ tr/gatcGATC/ctagCTAG/; # complement
    $seq = reverse $seq;      # and reverse
    return $seq;
}
```

If `param()` returns true, that means that we have some user input

Reverse Complementator

Enter your sequence here:

Or upload a sequence here: `smckay/Desktop/myseq.txt`

Reverse complement of myseq.txt

GAGAGAGATGAGAAGCACAACTCTTTTTT

<http://mckay.cshl.edu/cgi-bin/course/sequpload.pl>

Adding Cascading Stylesheets

```
#!/usr/bin/perl -w
# Script: veggies_with_style.pl
use CGI ':standard';

my $css = <<END;
<style type="text/css">
li.yellow { color: yellow }
li.green { color: green }
li.red { color: red }
ol {
background-color: gainsboro;
padding: 5px;
margin-left: 200px;
width: 150px;
}
ul { background-color: black }
</style>
END

print header,
start_html( -title => 'Vegetables',
            -head => $css );

print
hl('Eat Your Vegetables'),
ol(
li({'broccoli', 'peas', 'cabbage'}),
li('peppers',
ul(
li({'-class => 'red'}, 'red'),
li({'-class => 'yellow'}, 'yellow'),
li({'-class => 'green'}, 'green')
)
)
),
hr,
end_html;
```

Eat Your Vegetables

1. broccoli
2. peas
3. cabbage
4. peppers

- o red
- o yellow
- o green

http://mckay.cshl.edu/cgi-bin/course/veggies_with_style.pl

External stylesheet

```
#!/usr/bin/perl -w
# Script: veggies_with_style.pl
use CGI ':standard';

my $css = '/css/veggies.css';

print header,
start_html( -title => 'Vegetables',
            -style => $css );

print
hl('Eat Your Vegetables'),
ol(
li({'broccoli', 'peas', 'cabbage'}),
li('peppers',
ul(
li({'-class => 'red'}, 'red'),
li({'-class => 'yellow'}, 'yellow'),
li({'-class => 'green'}, 'green')
)
)
),
hr,
end_html;
```

http://mckay.cshl.edu/cgi-bin/course/veggies_with_style2.pl

CGI Exercises

Problem #1

Write a CGI script that prompts the user for his or her name and age. When the user presses the submit button, convert the age into "dog years" (divide by 7) and print the result.

Problem #2

Accept a DNA sequence and break it into codons.

Extra credit: Translate the codons into protein.

Databases and Database Access

A table of genes

Gene ID	Chromosome	Start	End	Strand	Class
GRMZM2G306328	chr2	175194049	175196453	-1	est
GRMZM2G027393	chr2	175212542	175213269	-1	cdna
GRMZM2G002915	chr2	175243929	175246053	1	est
GRMZM2G419606	chr2	175320426	175321226	-1	cdna
GRMZM2G119906	chr2	175323967	175325504	-1	cdna
GRMZM2G119950	chr2	175325765	175331607	-1	cdna
GRMZM2G125775	chr2	175462240	175463416	-1	cdna
GRMZM2G425965	chr2	175482597	175484512	-1	est
AC195825.3_FG001	chr2	176152209	176155132	-1	fgenesh

- Each row is a *record* of a gene
- Each column is a set of values constrained by a *type*
- A simple query: What is the location of gene 'GRMZM2G42775'?

A more complex query

Table 1: GO Terms of genes

Gene ID	Go Term
GRMZM2G002903	nucleic acid binding
GRMZM2G002903	intracellular
GRMZM2G002903	transport
GRMZM2G002915	DNA binding
GRMZM2G002915	transcription factor activity
GRMZM2G002915	nucleus
GRMZM2G002915	transcription
GRMZM2G002915	transcription regulator activity
GRMZM2G002948	multicellular organismal development
GRMZM2G002948	cellular process
GRMZM2G002950	nucleotide binding
GRMZM2G002950	protein binding

Table 2: Expression values

Gene ID	Exp1	Exp2	Exp3	Exp4
GRMZM2G003109	127.24	86.973	214.73	109.8
GRMZM2G003138	124.73	119.41	125.77	107.08
GRMZM2G003165	77.78	163.4	69.063	51.56
GRMZM2G003167	231.41	420.47	82.018	88.929
GRMZM2G003179	239.6	399.86	483.38	361.11
GRMZM2G003234	107.14	99.023	125.07	84.288
GRMZM2G003246	151.39	94.289	69.389	54.414
GRMZM2G003252	374.61	966.41	560.12	464.19
GRMZM2G003354	4170.1	3378.6	1876.9	2153.5
GRMZM2G003368	13835	5958.7	77.495	100.6

Table 3: Gene table

Gene ID	Chromosome	Start	End	Strand	Class
GRMZM2G306328	chr2	175194049	175196453	-1	est
GRMZM2G027393	chr2	175212542	175213269	-1	cdna
GRMZM2G002915	chr2	175243929	175246053	1	est
GRMZM2G419606	chr2	175320426	175321226	-1	cdna
GRMZM2G119906	chr2	175323967	175325504	-1	cdna
GRMZM2G119950	chr2	175325765	175331607	-1	cdna
GRMZM2G125775	chr2	175462240	175463416	-1	cdna
GRMZM2G425965	chr2	175482597	175484512	-1	est
AC195825.3_FG001	chr2	176152209	176155132	-1	fgenesh

“What are the classes of highly expressed genes in region 50Mb-55Mb of chromosome 5?”

What is a database

- A collection of data
 - Text file with a list of genes
 - GFF text file
 - BAM file
 - Excel spreadsheet
 - Set of tables in MySQL

DBMS: Software for managing databases

- Database Management Systems (DBMS)
 - General term for software for managing data
 - Creating tables
 - Loading data
 - Querying data
 - E.g: **MySQL, SQLite**, Oracle, Microsoft Access, Berkley DB, MongoDB



RDBMS:

Relational Database Management Systems

- Software for managing related data that is stored across multiple tables

Using a DBMS

- Through a user-interface
 - E.g: MySQL workbench, HeidiSQL, SequelPro, SQLite Manager, SQLite Spy
- Programmatically through SQL
 - Structured Query Language
 - E.g: “select gene_id from gene_table where chromosome = ‘chr2’;”
- Programmatically through an API in another programming language
 - **Perl DBI**
 - Java JDBC, C ODBC

What we will do today

- Creating databases, tables in MySQL
- Querying and manipulating data in SQL
- Querying and manipulating data using Perl DBI

MySQL

- A robust RDBMS is very popular for large bioinformatics databases. Great for:
 - Very large, persistent datasets
 - Multi users with different permission levels
 - High volume transactions
- To access the mysql client from the command line, you need 4 pieces of information:
 1. Host (*Defaults to localhost*)
 2. Port (*Defaults to 3306*)
 3. Username
 4. Password
- When MySQL is first installed, a 'root' account for administration is initialized, without a password

Using MySQL shell

- Start the MySQL client from the command line, and this will bring up a MySQL shell, connected to the MySQL server on your local machine
\$ `mysql -u root`
- For example, to connect to the public Ensembl MySQL:
\$ `mysql -h ensembl.org -P 5306 -u anonymous`
- Basic commands in the MySQL shell, line of commands must end with a ‘;’
`mysql> show databases;`
`mysql> create database progbio2011; # progbio2011 is the database name`
`mysql> use progbio2011; # Use another database`
`mysql> help;`
- To quit:
`mysql> \q;`
- To cancel a command:
`mysql> \c;`

Creating a table

Things to consider:

- Table name
- Name of each column
- Data type of each column
- Range of values of data in each column

Basic Datatypes

- Numeric
 - INT : for integers
 - Double : numerical data with decimals
- Strings
 - CHAR : for strings up to 255 in length
 - TEXT : large strings
- Lots more on the MySQL website
<http://dev.mysql.com/doc/refman/5.6/en/data-types.html>
- Also see cheat sheet on course webpage.

SQL: Creating a table

gene_id	chr	start	end
GRMZM2G306328	chr2	175194049	175196453
GRMZM2G027393	chr2	175212542	175213269
GRMZM2G002915	chr2	175243929	175246053
GRMZM2G419606	chr2	175320426	175321226

```
CREATE TABLE genes (  
    `gene_id` char(25) NOT NULL DEFAULT '',  
    `chr` char(5) NOT NULL DEFAULT '',  
    `start` int(9) NOT NULL DEFAULT '0',  
    `end` int(9) NOT NULL DEFAULT '0',  
    PRIMARY KEY (`gene_id`)  
);
```

```
CREATE TABLE tablename (  
    column_1_name datatype [optional constraint]  
    column_2_name datatype [optional constraint]  
    ....  
);
```

KEYS and Indexes

- INDEX
 - Synonymous with KEY
 - It is the lookup column, or a set of columns, for a table.
 - There can be more than one KEY in a table
- PRIMARY KEY
 - The primary key for a table represents the column, or set of columns, that is mostly frequently used as an index to the table
 - Columns used as the primary keys must be contain values that are unique to each row
 - There can only be one primary key in a table

SQL: Simple query

SELECT column_name [,column_names] from table;

```
SELECT gene_id from genes;
```

Use limit when the list is too long

```
SELECT gene_id, chr, start, end from genes  
limit 10;
```

Wildcard character "" for all columns in table*

```
SELECT * from genes limit 10;
```

SQL: SELECT ... WHERE for filtering results

what are the genes on chromosome 5

```
SELECT gene_id FROM genes WHERE chr='chr5';
```

what are the genes that lie within 50Mb – 55 Mb of chromosome 5

```
SELECT gene_id  
FROM genes  
WHERE chr='chr5'  
and end >= 50000000  
and start <= 55000000;
```

SQL: SELECT ... WHERE with OR

what are the genes that lie within chr5 with evidence 'est' or 'cdna' ?

```
SELECT gene_id , class
FROM genes
WHERE chr='chr5'
and (evidence='cdna' OR evidence= 'est');
```

SQL: Sorting and Distinct

What are the last 20 genes on chr 10;

```
SELECT gene_id, chr, start, end FROM genes
WHERE chr='chr10'
ORDER BY end desc LIMIT 20;
```

What is the unique list of gene evidences in the genes table?

```
SELECT DISTINCT evidence from genes;
```


SQL: SELECT COUNT... GROUP BY

count the number of rows in the table

```
SELECT COUNT(*) FROM genes;
```

count the number of genes in chromosome 5

```
SELECT COUNT(*) FROM genes where chr='chr5';
```

what if we want to return the number of genes in each chromosome?

```
SELECT chr, COUNT(*) FROM genes GROUP BY chr;
```

- Other SQL functions besides COUNT
 - avg, min, max, concat

SQL: select - join

Gene ID	Go Term
GRMZM2G002903	nucleic acid binding
GRMZM2G002903	intracellular
GRMZM2G002903	transport
GRMZM2G002915	DNA binding
GRMZM2G002915	transcription factor activity
GRMZM2G002915	nucleus
GRMZM2G002915	transcription
GRMZM2G002915	transcription regulator activity
GRMZM2G002948	multicellular organismal development
GRMZM2G002948	cellular process
GRMZM2G002950	nucleotide binding
GRMZM2G002950	protein binding

Gene ID	Exp1	Exp2	Exp3	Exp4
GRMZM2G003109	127.24	86.973	214.73	109.8
GRMZM2G003138	124.73	119.41	125.77	107.08
GRMZM2G003165	77.78	163.4	69.063	51.56
GRMZM2G003167	231.41	420.47	82.018	88.929
GRMZM2G003179	239.6	399.86	483.38	361.11
GRMZM2G003234	107.14	99.023	125.07	84.288
GRMZM2G003246	151.39	94.289	69.389	54.414
GRMZM2G003252	374.61	966.41	560.12	464.19
GRMZM2G003354	4170.1	3378.6	1876.9	2153.5
GRMZM2G003368	13835	5958.7	77.495	100.6

Gene ID	Chromosome	Start	End	Strand	Class
GRMZM2G306328	chr2	175194049	175196453	-1	est
GRMZM2G027393	chr2	175212542	175213269	-1	cdna
GRMZM2G002915	chr2	175243929	175246053	1	est
GRMZM2G419606	chr2	175320426	175321226	-1	cdna
GRMZM2G119906	chr2	175323967	175325504	-1	cdna
GRMZM2G119950	chr2	175325765	175331607	-1	cdna
GRMZM2G125775	chr2	175462240	175463416	-1	cdna
GRMZM2G425965	chr2	175482597	175484512	-1	est
AC195825.3_FG001	chr2	176152209	176155132	-1	fgenesh

SQL: simple join

what are the expression values for all transcription factors in experiment 1?

```
SELECT genes_go.gene_id, go_term, exp1
FROM genes_go, expression
WHERE genes_go.gene_id = expression.gene_id
and go_term = 'transcription regulator
activity' ;
```

Let's do this now

- “What are the classes of highly expressed genes in region 50Mb-55Mb of chromosome 5?”

Perl DBI

- DBI is a module that provides access to DBMS in Perl
- It hides the nuts and bolts for connecting to each type of DBMS, leaving a consistent interface for connecting to a database
- The key object in DBI is the database handle (`$dbh`), which represents a connection to a DBMS.

Three easy steps for database transaction in DBI:

1. Create a database handle

```
$dbh = DBI->connect(...)
```

2. Execute a SQL statement using the database handle

```
$dbh->do something
```

3. Disconnect the handle

```
$dbh->disconnect
```

Perl DBI Step 1: Constructing the handle

for MySQL

```
my $dbname = 'prog2011';
```

```
my $driver = 'mysql';
```

```
my $user = 'root';
```

```
my $passwd = '';
```

```
my $host = 'localhost';
```

```
my $port = 3306;
```

```
my $dsn = "DBI:$driver:database=$dbname;host=$host;port=$port";
```

```
my $dbh = DBI->connect($dsn,$user,$passwd);
```

for SQL lite

```
my $dbname = 'prog2011';
```

```
my $driver='SQLite';
```

```
my $dsn = "DBI:$driver:$dbname";
```

```
my $dbh = DBI->connect($dsn);
```

Perl DBI Step 2: Executing the SQL

1. Construct the SQL query

```
my $sql = "SELECT count(*) From gene";
```

2. Execute the transaction using the database handle

– For querying and fetching data:

```
my $results_array_ref = $dbh->selectall_arrayref($sql);
```

DBI Example: creating a table

With `$dbh->do()`

```
#!/usr/bin/perl
use strict;
use warnings;
use DBI;

my $dbname = 'progbio2011';
my $user = 'root';
my $passwd = '';
my $host = 'localhost';
my $port = 3306;

my $dsn = "DBI:mysql:database=$dbname;host=$host;port=$port";
my $dbh = DBI->connect($dsn,$user,$passwd);

my $sql = "CREATE table foo (bar char(10)); ";

$dbh->do($sql);

$dbh->disconnect;
exit;
```

```
=pseudocode
    $dbh = DBI->connect($dsn)
    $dbh->do(SQL)
    $dbh->disconnect
=end
```


DBI : Fetch a list of genes using DBI

With `$dbh->selectall_arrayref`

```
#!/usr/bin/perl
use strict;
use warnings;
use DBI;

my $dbname = 'progbio2011';
my $user = 'root';
my $passwd = '';
my $host = 'localhost';
my $port = 3306;

my $dsn = "DBI:mysql:database=$dbname;host=$host;port=$port";
my $dbh = DBI->connect($dsn,$user,$passwd);

my $query = "SELECT gene_id, chr, start, end from genes";
my @results = @{$dbh->selectall_arrayref($query)};

foreach my $row_ref ( @results){
    my $str = join "\t",@{$row_ref};
    print $str,"\n";
}

$dbh->disconnect;
exit;
```

=pseudocode

`$dbh = DBI->connect($dsn)`

`$fetched_results1 = $dbh->fetch SQL query`

`do something with results`

`$dbh->disconnect`

=end

DBI : Count using selectrow_arrayref

For SQL queries which will only return a *single* row,

e,g: SELECT COUNT query

We can use `$dbh->selectrow_arrayref`

```
#!/usr/bin/perl
use strict;
use warnings;
use DBI;

my $dbname = 'progbio2011';
my $user = 'root';
my $passwd = '';
my $host = 'localhost';
my $port = 3306;

my $dsn = "DBI:mysql:database=$dbname;host=$host;port=$port";
my $dbh = DBI->connect($dsn,$user,$passwd);

my $query = "SELECT COUNT(*) from genes where chr = 'chr10'";
my @row = @{$dbh->selectrow_arrayref($query)};

print join ("\t",@row),"\n";

$dbh->disconnect;
exit;
```

=pseudocode

`$dbh = DBI->connect($dsn)`

`$fetch_row = $dbh->fetch SQL query`

do something with results

`$dbh->disconnect`

=end

DBI : Querying using placeholders

fetch expression level for a list of genes

.....

```
my $dbh = DBI->connect( $dsn, $user, $passwd );
```

```
my $query = "SELECT exp1, exp2, exp3, exp4
            FROM expression
            WHERE gene_id = ?";
```

```
my $sth = $dbh->prepare($query);
```

```
my $file =shift;
```

```
open IN,"<",$file || die ("Can't open file $file $!");
```

```
while (my $gene_id = <IN>){
```

```
    chomp $gene_id;
```

```
    $sth->execute($gene_id);
```

```
    my @results = @{$sth->fetchall_arrayref};
```

```
    foreach my $row_ref (@results) {
```

```
        my $str = join "\t", @{$row_ref};
```

```
        print $gene_id,"\t",$str, "\n";
```

```
    }
```

```
}
```

```
close IN;
```

```
$dbh->disconnect;
```

```
exit;
```

=pseudocode

```
$dbh = DBI->connect($dsn)
```

```
$sth = $dbh->prepare(SQL)
```

```
loop:
```

```
    $sth->execute
```

```
    $sth->fetchrow_array;
```

```
end loop
```

```
$dbh->disconnect
```

=end

DBI : Placeholders vs selectall

fetch expression level for a list of genes

Using placeholders

=pseudocode

```
@genelist = read from file

$dbh = DBI->connect($dsn)
$sql = "SELECT exp1
      FROM expression
      WHERE gene_id = ?"

$stmt = $dbh->prepare($sql)

loop through genelist:
    $sth->execute($gene)
    $expr = $sth->fetchrow_array
    print $expr
end loop
$dbh->disconnect
```

=end

- 1 database transaction for each gene queried
- Slow if you have many genes to query

Using selectall

=pseudocode

```
@genelist = read from file

$dbh = DBI->connect($dsn)
$sql = "SELECT gene, exp1 FROM expression";

%gene_expr hash
for each result in $dbh->selectall_arrayref($sql)
    $gene_expr hash{$gene} = $expr
end

loop through genelist:
    print $expr if exist in %gene_expr hash
end loop

$dbh->disconnect
```

=end

- A single database transaction
- Slow if you're fetching millions of rows and you end up only needing a small fraction

For other DBI functions, see CPAN

With `$dbh->selectall_hashref`

```
...  
  
my $dsn = "DBI:mysql:database=$dbname;host=$host;port=$port";  
my $dbh = DBI->connect($dsn,$user,$passwd);  
  
my $query = "SELECT gene_id, chr, start, end from genes";  
  
my %results = %{$dbh->selectall_hashref($query,'gene_id')};  
  
foreach my $gene (keys %results){  
    print $gene,"\t",  
          $results{$gene}->{'chr'},"\t",  
          $results{$gene}->{'start'},"\t",  
          $results{$gene}->{'end'},"\n";  
}  
  
$dbh->disconnect;  
  
exit;
```

Other useful MySQL commands

- For loading a text file into a table from the unix command line:

```
$ mysqlimport --local -u root databasename filename.txt
```

Filename must have the same name as the table you are trying to load data into

- For dumping data from a table:

```
$ mysqldump -u root databasename tablename > table.sql
```

– Or for dumping the entire MySQL database:

```
$ mysqldump -u root databasename > database.sql
```

Other useful SQL commands

- Inserting new rows into table

```
INSERT into genes (gene_id, chr, start, end,  
evidence) values ('foo', 'chrX', 1000, 1500, 'cdna');
```

- Updating data in table

```
UPDATE genes SET gene_id = 'bar' WHERE gene_id =  
'foo';
```

Alternate RDBMS - SQLite

- Simple RDBMS that is installed by default in most systems
 - `sqlite3`
- To create a new database in SQLite from the unix command line
 - `$ sqlite3 mydatabase`
- The command above also brings you into the `sqlite3` client
- `sqlite3` client allows
 - `sqlite> .help`
 - `sqlite> .tables`
 - `sqlite> .import FILE table`
 - `sqlite> .exit`

Scientific Programming

Jim Tisdall

Programming for Biology 2011

Lecture Notes

1. [The Problem](#)
2. [Time and Space and Algorithms](#)
3. [Using Less Time](#)
4. [Using Less Space](#)
5. [Profiling](#)
6. [Parallel Processing](#)

Suggested Reading

Mastering Algorithms with Perl

by Orwant, Hietaniemi, and Macdonald
(An excellent algorithms text with implementations in Perl)

Introduction to Algorithms

by Cormen et al.
(This is the standard modern text)

Writing Efficient Code

by Jon Bentley
(Hard to find. Great book.)

Introduction to Automata Theory, Languages, and Computation

by Hopcroft and Ullman
(The standard, mathematical textbook for theoretical computer science.)

Computers and Intractability: A Guide to the Theory of NP-Completeness

by Gary and Johnson
(Very well written.)

Network Programming with Perl

by Lincoln Stein
(Client-server network programming.)

An Introduction to Parallel Algorithms

by Joseph Jaja
(For the next generation of computers.)

Programming for Biology

Jim Tisdall, tisdall -- at -- jimtisdall.com

Last modified: Sun Oct 23 20:36:27 EDT 2011

Moral

Bioinformatics often requires a programming style that minimizes the use of *space and time*.

How to minimize space and time comes under the general rubric of *scientific programming*.

This lecture will introduce some standard scientific programming methods and ideas.

The Problem

Biological data will just barely fit on modern and affordable computers.

Biological computations are just barely possible on modern and affordable computers.

High-throughput sequencing. Multiple genomes. Genbank. Image analysis.

[Programming for Biology](#) Last modified: Sun Oct 23 20:36:27 EDT 2011

Time and Space and Algorithms

Minimizing time and space results in programs that run faster and in smaller computers; it can make the difference between having a workable program or none at all.

A program's use of time and space depends on the **algorithms** and associated **data structures** used to solve a problem.

An **algorithm** is the design or idea of a computation. It usually can be expressed in terms of a specific computer program, or more informally as in *pseudocode*.

Typically there are many possible *algorithms* for a given problem. Some ways use less time and/or less space than other ways.

A **data structure** is the form of the computation as it proceeds. A great deal of biological data is organized into **two-dimensional tables in relational databases**. Relational database tables are the standard workhorse for storing data in biology, and are useful in a surprising number of situations.

It's important to know, however, that **often the best algorithm will use some other data structure** such as a doubly-linked list or a tree, for example. Such data structures might better represent biological structures, gene networks, evolutionary relationships, and so on. And, such data structures may be used in sometimes surprising ways to speed up a computation.

The **space** of an algorithm is just the amount of computer memory it uses. This will reflect the size of the input to the algorithm, and the data structures that are employed in the computation.

The **time** of an algorithm is usually given as a function on the size of the input. So if the input is of size n , the algorithm might take time n^2 . So, for instance, if you gave such an algorithm a hundred genes, it would take about 10000 units of time to run; if you gave it ten thousand genes, it would take 100000000 units of time to run.

Time is roughly estimated according to the number of basic operations performed by your program as it runs. Basic operations are adding, concatenating two strings, printing, etc. The overall structure of the program is what is important, not an actual prediction of exactly how many seconds the program will take.

What can be computed?

We are primarily interested in building software to achieve easily computed, but useful, results. We're learning beginning programming techniques, not computer science theory. Therefore, we will not delve into the study of algorithms in any depth in this course.

HOWEVER: it can easily happen that you may want to compute something that is hard to compute in a week, or a year, or even at all. This is a very practical problem, and it may come up fairly quickly for you, depending on your research problem. It's important to know what you can do about it.

The idea is that **there are limits to what can be computed**. These limits take two main forms: **intractability**

and **undecidability**.

The main point:

MANY PROBLEMS CANNOT BE COMPUTED

but it's possible to get "pretty good" answers for many of them

How algorithms are measured

Algorithms are typically classified by how fast they perform on given inputs, by giving their speed as a function of the size of the input. The size of the input is usually represented by the variable **n**. (**n** might be the size of a genome, for instance.)

Say for example that an algorithm gets an input of size **n**, and then just to write the answer it must write an output in space of size 2^n , which we say will take about 2^n time to write. Then the algorithm's time complexity is "order of 2 to the n", written in a shorthand called **big Oh** notation as

$O(2^n)$.

This way of measuring an algorithm is called *time complexity*.

Examples:

$O(2^n)$ computations: *exponential, intractable, bad*

$O(n^2)$ computations: *polynomial, tractable, good*

$O(5n)$ computations: *linear, tractable, great*

$O(\log(n))$ computations: *logarithmic, tractable, amazing*

$O(1)$ computations: *constant time, tractable, unbelievably great*

If the size of the input **n** is 3, then all methods take a short amount of time -- 8 and 9 and 15 and about 1, respectively.

But if the size of the input **n = 100**, then **log(n)** is about 6, **5n** is 500, and **n²** is 10,000 which is still not bad. However, **2ⁿ** is bigger than the number of atoms in the universe. (And is the universe really finite? Oh well ... who's counting?)

Intractability

Intractability means that a problem cannot be computed in a reasonable amount of time. Many biological problems are intractable.

Example: in phylogeny we learn that there are many possible trees that can be built, and that the number of possible trees grows exponentially as you increase the number of taxa and as you increase the evolutionary time under discussion.

To find the best solution in an exponentially-growing space, such as the space of all possible evolutionary trees, often requires examining each possibility, and so may take an exponentially-growing time. Problems that have this property (very loosely defined here) are called

NP

(for non-deterministic polynomial time), and certain canonical such problems are called

NP-complete.

NP-complete problems are all essentially interchangeable; that is, they all come down to essentially the same problem. The prototypical NP-complete problem is the

TRAVELING SALESMAN PROBLEM:

given a set of cities and the distances between them, what is the shortest route a traveling salesman can take to visit each one?

By the time you get to about 30 cities, the number of possible routes cannot be computed in your lifetime; by the time you reach about 60 cities, there are more possible routes than there are atoms in the universe. And we don't know a better way to find the best route than to look at each one.

An aside: no one has proved that NP-complete problems *must* require looking at each individual possibility. If you could find a *polynomial-time* algorithm for any NP-complete problem, you would be the most famous computer scientist/mathematician around, and would surely win a Nobel prize. Few people believe it will be done, but it's been an open problem for many years, and no one yet can prove that it can't be done. This is called the **P =? NP** problem.

The practical implications:

If you have a lot of data for your problem, and the problem is in NP, then you have **no practical solution to find the best, optimal answer** except on very small data sets.

But the good news is: there are **approximation algorithms** that will give you a **very good answer** in a reasonable amount of time, even if it's not the optimal answer. Such approximation algorithms underlie many of the practical approaches to such problems as phylogeny, sequence assembly, and many other problems in bioinformatics.

Undecidable problems

Less likely to be a problem for the practical bioinformatics programmer, but something to be aware of, is that there are **problems for which no solution is possible**.

These problems are called *undecidable*, and they were first demonstrated by Alan Turing and others in the

1930s.

Here's the most famous undecidable problem: the

HALTING PROBLEM

Write a program that can scan any other program and decide if it will eventually halt, or if it will go on forever without coming to a stop.

In other words, write a virus checker for nonhalting programs.

As an example of such a nonhalting "virus", here's a perl program that goes on forever (until you stop it):

```
while(1) {}
```

That looks easy to recognize. But we can *prove* that no program can be written that would catch *all* such non-halting programs.

The fact that such an easily-described problem as the HALTING PROBLEM has no solution is, when you think about it, a very deep and profound statement about the limits of human knowledge.

[Programming for Biology](#) Last modified: Sun Oct 23 20:36:27 EDT 2011

Using Less Time

The Art and Science of Algorithm Design

Knowledge can be classified into two types: *procedural knowledge* and *declarative knowledge*.

Declarative knowledge is a collection of facts. (E.g., Watson's great textbook "The Molecular Biology of the Gene")

Procedural knowledge is knowledge of *how to do things*. This is the kind of knowledge captured by computer algorithms.

Procedural knowledge has been growing immensely since (programmable digital) computers put the requirement to specify how to do something -- that is, to formulate an *algorithm* -- into the very center of our economic, scientific, and cultural lives.

Algorithms are discovered by a combination of mathematics and art and science and luck and training and talent. Much of what we do on computers relies on the accumulated procedural knowledge -- algorithms -- of our culture.

A good algorithm is more important than a good computer

Finding a better algorithm can be much more important than getting a better, faster computer.

For the following examples I created a set of random DNA that I'll use as my "promoters". I include the code here. (We'll return to this code later in the lecture).

```
#
# Main program -- make promoters from random DNA
#

srand();

$dna = make_random_DNA(1000000);
open(DNA, ">genomic_data") or die;
print DNA $dna;

@promoters = make_random_DNA_set(10, 5000);
open(PROMOTERS, ">promoters") or die;
print PROMOTERS join("\n",@promoters),"\n";

exit 0;

#
# Subroutines
#

# Make a string of random DNA of specified length.
sub make_random_DNA {
```



```

    my($length) = @_;
    my $dna;

    for (my $i=0 ; $i < $length ; ++$i) {
        $dna .= randomnucleotide( );
    }

    return $dna;
}

# Make a set of random DNA
sub make_random_DNA_set {

    my($length, $size_of_set) = @_;
    my $dna;
    my @set;

    # Create set of random DNA
    for (my $i = 0; $i < $size_of_set ; ++$i) {

        $dna = make_random_DNA ( $length );
        push( @set, $dna );
    }

    return @set;
}

# Select at random one of the four nucleotides
sub randomnucleotide {

    my(@nucleotides) = ('A', 'C', 'G', 'T');

    return randomelement(@nucleotides);
}

sub randomelement {

    my(@array) = @_;

    return $array[rand @array];
}

```

Consider this fragment of perl code, written to find a set of short sequences in a genome ("findpromoters0"):

```

# Read the promoter data from a file
open(PROMOTERS, "promoters") or die "a horrible death: $!";
my @promoters = <PROMOTERS>;

# Look for each occurrence of each promoter in the genome
foreach my $promoter (@promoters) {
    chomp $promoter;

    # Read the genome data from a file
    open(GENOME, "genome_data") or die "a horrible death: $!";

```

```

my $genome = <GENOME>;

while($genome =~ /$promoter/g) {
    # $-[0] prints the location of the find
    #print "$promoter $-[0]\n"; exit;
    $db{$promoter} = $-[0];
}
}

```

Now this code is good perl. It is syntactically correct, and it will produce the correct output. It will run, and in the end you will print out all the locations of the sequence.

Let's see how long it takes to run:

```

-bash-3.00$ date; perl findpromoters0; date
Thu Oct 20 14:28:06 EDT 2005
Thu Oct 20 14:28:48 EDT 2005
-bash-3.00$

```

Okay, so 42 seconds isn't bad! But wait ... what if we had the entire human genome, and a million tags? I'll let you do the math, or the experiment, but it takes too long.

So we try to make it faster. How? Well, we notice that for each tag, we're reading in the entire genome from the disk. Let's rewrite the code so that it only reads the genome in once (findpromoters1):

```

# Read the genome data from a file
open(GENOME, "genome_data") or die "a horrible death: $!";
my $genome = <GENOME>;

# Read the promoter data from a file
open(PROMOTERS, "promoters") or die "a horrible death: $!";
my @promoters = <PROMOTERS>;

# Look for each occurrence of each promoter in the genome
foreach my $promoter (@promoters) {
    chomp $promoter;
    while($genome =~ /$promoter/g) {
        # $-[0] prints the location of the find
        #print "$promoter $-[0]\n"; exit;
        $db{$promoter} = $-[0];
    }
}
}

```

And the time for that is:

```

-bash-3.00$ date; perl findpromoters1; date
Thu Oct 20 14:30:46 EDT 2005
Thu Oct 20 14:31:05 EDT 2005
-bash-3.00$

```

>From 42 seconds to 19 seconds -- sweet!

But can we do better? Notice that for each promoter, we're scanning through the entire genome. So we're

scanning through the entire genome 5000 times.

Is there a way we can scan through the entire genome just once? Yes, and here is one solution:

```
# Read the genome data from a file
open(GENOME, "genome_data") or die "a horrible death: $!";
my $genome = <GENOME>;

# Read the promoter data from a file
open(PROMOTERS, "promoters") or die "a horrible death: $!";
foreach $promoter (<PROMOTERS>) {
    chomp $promoter;
    $promoters{$promoter} = 1;
}

# Look for each occurrence of each promoter in the genome
my $genomelength = length($genome);
for($i = 0; $i < $genomelength - 10 + 1; ++$i) {
    my $subsequence = substr($genome, $i, 10);

    # Now we just look in the hash to see if this subsequence is a promoter
    if($promoters{$subsequence}) {
        $db{$promoter} = $i;
    }
}
}
```

and we run a timing on it to get ("findpromoters2"):

```
-bash-3.00$ date ; perl findpromoters2 ; date
Thu Oct 20 15:42:15 EDT 2005
Thu Oct 20 15:42:16 EDT 2005
-bash-3.00$
```

That's one second, maybe less.

And so we've achieved a 43-fold speedup in our program. What was taking, say, two days to compute, now takes an hour. We couldn't have achieved that speedup going to a super expensive computer (well, maybe a cluster, which we'll discuss later.)

And so we see that finding a better algorithm is the best way to get good performance.

What, exactly, did we do? We eliminated unnecessary work. We eliminated the repetitive reading in of the genome data from the disk; and we eliminated multiple scanning through the genome data.

These are the kinds of things that you can often find in the first version of a working program. So don't neglect the important step of editing your code after you get a working draft.

[Programming for Biology](#) Last modified: Sun Oct 23 20:36:27 EDT 2011

Using Less Space

Here is the main problem of space in bioinformatics:

Very large strings will swamp the main memory on your computer.

(Main memory, or RAM, is where your computer holds a running program; it is much smaller than the memory on your disks.)

When a program on your computer starts to use up too much main memory, its performance starts to degrade. The program will first enlist a portion of disk space to hold the part of the running program that it can no longer fit. This is called **swapping**.

But when a program starts swapping, which involves a lot of writing and reading to and from hard disk, it can get increasingly slow. The program may even start **thrashing**, that is, repeatedly writing and reading large amounts of data between main memory and hard disk. A program that is thrashing is going really slow, and it's slowing down the whole computer and other programs, too.

Take this snippet of code that calls `get_chromosome`:

```
my $chromosome1 = getchromosome(1);
```

When `getchromosome(1)` returns the data from human chromosome 1 to be stored in `$chromosome1`, the program uses 100Mb of memory.

Operating on the chromosome may use additional memory. For instance, in perl, when you do a regular expression search, you often want to save the successful match by using parentheses that set the special variables `$1`, `$&`, and so on.

```
$chromosome =~ /AA(GAGTC*T)/;  
my $pattern = $1;
```

But once you use these special variables, the inner workings of perl require the use of considerable additional memory by your program. And you may make copies of all or part of the chromosome.

Your resulting code may be clear, straightforward to understand, and correct -- all good and proper things for code to be -- but the amount of memory usage may still seriously slow down your program.

Motto: copying large strings is slow and takes up large amounts of memory

Editing for Space

Often, a program that barely runs at all and takes many hours of clogging up the computer, can be rewritten to run more quickly by rewriting the algorithm so that it uses only a small fraction of the memory. It will fit into less memory, and also run a lot faster.

Use references to save space

There's one easy way to cut down on the number of big strings in a program.

Normally (without using references) a subroutine makes copies of the values passed into it, and it makes copies of the values returned from it.

References allow subroutines to avoid the string copying.

When we pass a reference to a string as an argument to a subroutine, we don't pass a copy of the string -- we pass a reference to the string, which takes almost no additional space.

And when the subroutine ends, whatever we've done with the string is immediately available to the calling program, without having to use the `return` function, which would also copy the string.

In our example:

```
load_chromosome( 1, \%chromosome1 );
```

This new subroutine has two arguments. The 1 indicates that we want the biggest human chromosome, chromosome 1.

The second argument is a reference to a scalar variable. Inside the subroutine, the reference is most likely used to initialize an argument `$chromref`, which is a reference to the genomic data. And then, in the subroutine, the DNA data is put into the dereferenced string:

```
sub load_chromosome {
    my($chromnumber, $chromref) = @_;

    ...(omitted)...

    $$chromref = <CHROMOSOME1>
}
```

It is not necessary to return the whole chromosome from the subroutine, which would make a copy of it. The value is passed by the reference *out* of the subroutine.

Using references is also a great way to pass a large amount of data *into* a subroutine without making copies of it. In this case, however, the fact that the subroutine can change the contents of the referenced data is something to watch out for.

The rule of thumb is: **if you don't need two copies of the data, you can use references.**

Managing Memory with Buffers

One of the most efficient ways to deal with very large strings is to deal with them a little at a time.

Here is an example of a program that searches an entire chromosome for a particular 12-base pattern, using

very little memory.

When searching for any regular expression in a chromosome, it's hard to see how you could avoid putting the whole chromosome in a string. But very often there's a limit to the size of what you're searching for. In this program, I'm looking for the 12-base pattern "ACGTACGTACGT."

I'm going to read the chromosome data into memory just a line or two at a time, search for the pattern, and then *reuse* the memory to read in the next line or two of data.

The extra programming work involves:

First, keeping track of how much of the data has been read in, so I can report the locations in the chromosome of successful searches.

Second, making sure I search across line breaks as well as within lines of data from the input file.

The following program reads in a FASTA file searches for my pattern in any amount of DNA--a whole chromosome, a whole genome, a year's worth of Solexa data, even all known genetic data, while using only a small amount of main memory.

```
$ perl find_fragment human.dna
```

For testing purposes I made a very short FASTA DNA file, `human.dna`, which contains:

```
>human dna: ACGTACGTACGT appears at positions 10, 40, and 98
AAAAAAAAACGTACGTACGTCCGCGCGCGCGCGCGCGCACGTACGTACG
TGGGGGGGGGGGGGGGGCCCCCCCCCGGGGGGGGGGGGAAAAAAAAACG
TACGTACGTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
```

Here's the code for the program `find_fragment`:

```
#!/usr/bin/perl

use warnings;
use strict;

# $fragment: the pattern to search for
# $fraglen:  the length of $fragment
# $buffer:   a buffer to hold the DNA from the input file
# $position: the position of the buffer in the total DNA

my($fragment, $fraglen, $buffer, $position) = ('ACGTACGTACGT', 12, '', 0);

# The first line of a FASTA file is a header and begins with '>'
my $header = <>;

# Get the first line of DNA data, to start the ball rolling
$buffer = <>;
chomp $buffer;

# The remaining lines are DNA data ending with newlines
while(my $newline = <>) {
```

```

# Add the new line to the buffer
chomp $newline;
$buffer .= $newline;

# Search for the DNA fragment, which has a length of 12
# (Report the character at string position 0 as being at position 1,
# as usual in biology)
while($buffer =~ /$fragment/gi) {
    print "Found $fragment at position ", $position + $-[0] + 1, "\n";
}

# Reset the position counter (will be true after you reset the buffer, next)
$position = $position + length($buffer) - $fraglen + 1;

# Discard the data in the buffer, except for a portion at the end
# so patterns that appear across line breaks are not missed
$buffer = substr($buffer, length($buffer) - $fraglen + 1, $fraglen - 1);
}

```

Here's the output of running the command

```
perl find_fragment human.dna:
```

```

Found ACGTACGTACGT at position 10
Found ACGTACGTACGT at position 40
Found ACGTACGTACGT at position 98

```

How the Code Works

I want to search for the fragment even if it is broken by new lines, so I'll have to look at least at two lines at a time. I get the first line, and in the while loop that follows I'll start by adding more lines to the buffer.

Then the while loop starts reading in the next lines of the FASTA file. The newline character is removed with `chomp` and the new line is added to the `$buffer`.

Then comes the short while loop that does the regular expression pattern match of the `$fragment` in the `$buffer`.

When the fragment is found the program simply prints out the fragment's position. The variable `$position` holds the position of the beginning of the buffer in the total DNA.

I also add 1, because biologists always say that the first base in a sequence of DNA is at position 1, whereas Perl says that the first character in a string is at position 0. So I add 1 to the Perl position to get the biologist's position.

The last two lines of code reset the buffer. First we eliminate the beginning (already searched) of the buffer, and then we adjust the `$position` counter accordingly. The buffer is shortened so that it just keeps the part at the very end that might be part of a pattern match that spans the newlines.

The program manages to search the entire genome for the fragment, while keeping at most two lines' worth of DNA in `$buffer`. It performs very quickly, compared to a program that reads in a whole genome and blows

out the memory in the process.

When You Should Bother

Programs may be developed on one computer, but run on very different computers.

A space-inefficient program might well work fine on your computer, but not work well at all when you run it on another computer with less main memory installed. Or, it might work fine on the fly genome, but start thrashing when you try it on the human genome.

If you know you'll be dealing with large data sets, like genomes, take the amount of space your program uses as an important constraint when designing and coding. Then you won't have to go back and redo the entire program when a large amount of DNA gets thrown at the program.

Data Compression

In Perl, as in any programming system, the size of the data that the program uses is an absolute lower bound on how fast the program can perform.

Each base is typically represented in a computer language as one ASCII character taking one 8-bit byte, so 3 gigabases equals 3 gigabytes. Of course, you could represent each of the four bases using only 2 bits, so considerable compression is possible; but such space efficiency is not commonly employed. When it is, you can pack 4 times as much data into a given space (for nucleotides, that is.)

```
A 00  
C 01  
G 10  
T 11
```

If you want an exercise, try using perl functions `pack` and `vec` to compress DNA sequence data to 4 bases per byte.

[Programming for Biology](#) Last modified: Sun Oct 23 20:36:27 EDT 2011

Profiling

You saw earlier an easy way on Unix to see how long a program takes:

```
date; perl findpromoters1; date
```

This prints the time, then immediately runs the program, and then immediately prints the time again.

Perl has several much more detailed ways to examine the performance of a program.

I'll just show you one of them, called **DProf**. DProf reports on various aspects of your program's performance.

The most valuable report is probably the summary by subroutine.

By seeing which subroutines are taking the most time, you can narrow your re-editing of the program to just those subroutines, and quickly make the improvements where they count the most.

For demonstration, I'm going to use a program with a few subroutines; namely, the `makerandom` program we used earlier to make random DNA genomic sequence and putative DNA binding sites.

First you have to load the `Devel::Prof` module in your program. You do this by adding the `-d:DProf` command-line argument. Then when your program runs, the module makes counts of many things in the program. Your program will take a bit longer to run, but you'll collect valuable statistics on its performance.

So one can simply run the program as usual, adding the command-line argument. When it's done, it will have created a file called `tmon.out` in my directory. I then run the `dprofpp tmon.out` program to see the results of the profile of my program:

```
$ perl -d:DProf makerandom
$ dprofpp tmon.out
Total Elapsed Time = 5.464274 Seconds
  User+System Time = 5.354274 Seconds
Exclusive Times
%Time ExclSec Cumuls #Calls sec/call Csec/c Name
 72.2   3.870   7.594 105000  0.0000 0.0000  main::randomnucleotide
 69.5   3.725   3.725 105000  0.0000 0.0000  main::randomelement
 33.7   1.807   9.402   5001   0.0004 0.0019  main::make_random_DNA
 0.22   0.012   0.525     1     0.0125 0.5250  main::make_random_DNA_set
$
```

If I wanted to speed this program up, I'd head straight for the `randomelement` and `randomnucleotide` subroutines to see what I might be able to tweak in them, since my analysis shows that they take almost all the time in the program.

DProf has many options, but this is how I almost always use it, as it's simple and tells me what I need to know.

Some older perls might not have DProf installed, in which case you have to do something like this: (you may

need root permission):

```
$ perl -MCPAN -e shell

cpan shell -- CPAN exploration and modules installation (v1.7601)
ReadLine support enabled

cpan> install Devel::DProf
CPAN: Storable loaded ok
Going to read /root/.cpan/Metadata
  Database was generated on Wed, 19 Oct 2005 22:01:03 GMT
Devel::DProf is up to date.

cpan> quit
Lockfile removed.
$
```

In this case perl reported that the `Devel::DProf` module was already installed with the latest version; if not, it would have installed it.

You know, I wonder if I can speed up my `makerandom` program. Let's look at it. Hmm. I did try a few things out: let's see how the new program `makerandom2` behaves:

```
$ perl -d:DProf makerandom2
$ dprofpp tmon.out
Total Elapsed Time = 1.27999 Seconds
  User+System Time = 1.27999 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c Name
 96.8  1.240  1.240   5001  0.0002 0.0002 main::make_random_DNA
  0.78  0.010  0.050     1  0.0100 0.0500 main::make_random_DNA_set
$
```

Cool! From over 5 seconds to a little over 1 second. A five-fold speedup!

How did I do it? Here's the new version:

```
srand();

my(@nucleotides) = ('A', 'C', 'G', 'T');

$dna = make_random_DNA(1000000);
open(DNA, ">genomic_data") or die;
print DNA $dna;

@promoters = make_random_DNA_set(10, 5000);
open(PROMOTERS, ">promoters") or die;
print PROMOTERS join("\n", @promoters), "\n";

# Make a string of random DNA of specified length.
sub make_random_DNA {

    my($length) = @_;
    my $dna;
```

```

    for (my $i=0 ; $i < $length ; ++$i) {
        $dna .= $nucleotides[rand @nucleotides];
    }

    return $dna;
}

# make_random_DNA_set
sub make_random_DNA_set {

    my($length, $size_of_set) = @_ ;
    my $dna;
    my @set;

    # Create set of random DNA
    for (my $i = 0; $i < $size_of_set ; ++$i) {

        # make a random DNA fragment
        $dna = make_random_DNA ( $length );

        # add $dna fragment to @set
        push( @set, $dna );
    }

    return @set;
}

```

First, I moved the line

```
my(@nucleotides) = ('A', 'C', 'G', 'T');
```

out of a subroutine and up to the top of the program. This way the array doesn't have to get reinitialized each time the program is called.

But much more importantly, I eliminated two subroutine calls entirely, and put their functionality directly into the lines of code that were calling them. First I axed `randomelement` by putting its functionality directly into the calling subroutine `randomnucleotide`: from

```

sub randomnucleotide {

    my(@nucleotides) = ('A', 'C', 'G', 'T');

    return randomelement(@nucleotides);
}

sub randomelement {

    my(@array) = @_ ;

    return $array[rand @array];
}

```

to

```
my(@nucleotides) = ('A', 'C', 'G', 'T');  
  
sub randomnucleotide {  
    return $nucleotides[rand @nucleotides];  
}
```

and finally I eliminated `randomnucleotide` by putting its code directly into the calling program: from

```
$dna .= randomnucleotide( );
```

to

```
$dna .= $nucleotides[rand @nucleotides];
```

In short, I eliminated two subroutine calls that were each being called 105000 times, and that made a significant speedup. Usually, you're more likely to try to improve a subroutine than to eliminate it, but as you see eliminating a subroutine can on occasion have big payoffs.

The book by Bentley "Writing Efficient Code" discusses such "tricks" in entertaining and useful detail.

So I hope you're convinced that `DProf` is worthwhile. There are other profiling methods available in Perl too, and you might want to explore them.

[Programming for Biology](#) Last modified: Sun Oct 23 20:36:27 EDT 2011

There are different ways to think of parallel processing.

Parallel Algorithms

One kind of parallel processing actually uses the specific topology of the interconnections between the CPUs to implement new kinds of algorithms. This kind of parallel processing is fascinating and gives you very fast programs, but is *way* beyond the scope of this lecture or this course. But I thought you'd like to know that it exists.

In this hard-core parallel algorithms work, you might work on special computers (e.g. "grids", "butterfly networks") or even on purely theoretical models of parallel computation, and you design algorithms to run on those types of parallel computers.

Parallel Processing on Networks and Clusters

More common is this scenario: say you are doing 40 tasks, one after the other, and each one takes an hour. It will take your working week to finish the tasks.

Now let's say you figure out a way to do all the tasks simultaneously, and each one still takes an hour. You'll now finish the tasks, all of them, in one hour instead of one week.

One kind of parallel processing is just like this example. That's the kind of parallelism I'll talk about here, in terms of networks and clusters and threads. You simply divide your program up into parts that can be performed simultaneously, and then you run each part on its own CPU. Not all problems can be divided up like this, but those that can (say running a million blast searches) can get big speedups fairly easily.

Network Programming

One of the most successful forms of multi-processor computing has been *network programming*.

Network programming involves connecting two or more computers by a communications line and implementing a protocol that enables them to exchange information.

The development of computer networks began in earnest in the 1950s, and the various networks were interconnected by the *internet* (from *interconnected networks*) beginning in the late 1970s.

The protocols supported by the internet gradually expanded, until the protocols known as the *web* (or "world wide web") became widely popular beginning around 1990.

It is quite possible to program several computers to interact, using the several programming interfaces to the protocols that are available from such languages as perl.

Perl has supported these protocol interfaces since the beginning. I can speak from personal experience that it's a lot of fun to build a useful network service in this way. (In 1992 I was searching all of Genbank with

regular expressions in about 35 seconds, by distributing the job with a network service written entirely in perl.)

I recommend the book "Network Programming with Perl" by Lincoln Stein if you're interested in these techniques.

Threads

Threads are different from, but related to, multiprocessing. Threads are multiple execution paths built into one process, that share resources like global variables, signals, and such. You can have a multithreading program that runs on a single processor; or, if you're running on a multiprocessor (it's common to have from 2 to around 24 processors on a given machine) the threads may be executed on different processors, giving you the advantage of parallelism.

Threads are a capability that is built into an operating system (or not, as the case may be.) If your operating system supports threads, and your programming language gives you access to them, then you can use them in your program.

If you're interested in threads, you want to use the "threads" (not "Threads") module:

```
use threads;
```

I'm going to skip the examples of threads programs: see me if you're interested.

Clusters

Clusters are multiple CPUs joined in a simple network. They are typically used to take a program that must compute the same way over many inputs, and run the program on all the CPUs, dividing the input up between them.

If you have access to a (usually) Linux cluster where you work, take the time to find out how to submit programs to it.

Once I had to do three computation-intensive calculations over several genomes. Each one took a week or two to finish when running on a single computer. On the Linux cluster, they all finished within a small number of hours, and using that precomputation I was able to carry my search for novel genes to a successful conclusion.

That Linux cluster had about 450 CPUs, and is a fairly big one. But it's quite straightforward -- you could do it yourself -- to buy 10 or 20 inexpensive Linux boxes and construct a Linux cluster that can speed up your large-scale, repetitive computations by 10 or 20 times.

Cloud computing

Cloud computing is a marketing term that has become very popular, so it's not a very exact way to describe a computation.

Typically, a "cloud" is a collection of computers owned by a company that can lease time to smaller firms or individuals, in order to accomplish large computations without the need for the customers to buy and maintain large computers and networks.

Increasingly, a "cloud" may incorporate some form of interface to services that enable cluster computing, for instance.

[Programming for Biology](#) Last modified: Sun Oct 23 20:36:27 EDT 2011

Programming for Biology Protein Evolution / Similarity Searching

What BLAST Does / Why BLAST works

Bill Pearson
wrp@virginia.edu

1

Sequence Similarity - Conclusions

- *Homologous* sequences share a common ancestor, but most sequences are *non-homologous*
- Always compare Protein Sequences
- Sequence Homology can be reliably inferred from statistically significant similarity (non-homology cannot from non-similarity)
- Homologous proteins share common structures, but not necessarily common functions
- Sequence statistical significance estimates are accurate (verify this yourself) $10^{-6} < E() < 10^{-3}$ is statistically significant
- Scoring matrices set evolutionary look back horizons - not every discovery is distant
- PSI-BLAST can be more sensitive, but with lower statistical accuracy

2

*Establishing homology from
statistically significant similarity*

Why BLAST works

- For most proteins, homologs are easily found over long evolutionary distances (500 My – 2 By) using standard approaches (BLAST, FASTA)
- Difficult for distant relationships or very short domains
- Most default search parameters are optimized for distant relationships and work well

3

This talk is not about:

- *Alignment*
 - Alignment quality may be more sensitive to parameter choice
 - Multiple sequences for biologically accurate alignments
- *Inferring Protein Function*
 - Homology (common ancestry) implies common structure (guaranteed), not necessarily common function
 - Homologs have different functions
 - Non-homologs have similar (or identical) functions
- *The best sequences for building evolutionary trees*
 - Protein sequences are clearly best for establishing homology, but DNA sequences may be better for resolving recent divergence

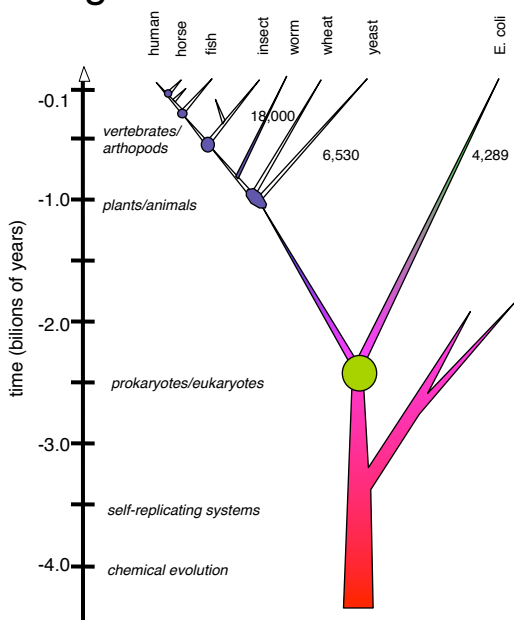
4

Protein Evolution and Sequence Similarity

- What is Homology and how do we recognize it?
- How do we measure sequence similarity – alignments and scoring matrices?
- DNA vs protein comparison
- Alignment Algorithms/Local sequence alignments
- Similarity scoring matrices
- When are we certain that an alignment is significant - similarity score statistics?
- When to trust similarity statistics?
- Improving sensitivity with PSI-BLAST

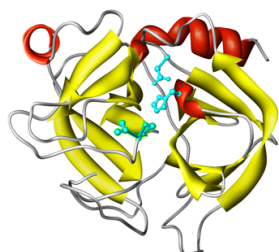
5

Homologues share a common ancestor



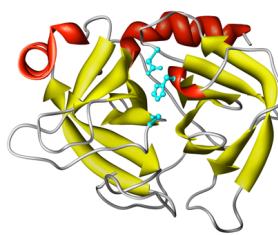
6

When do we infer homology?

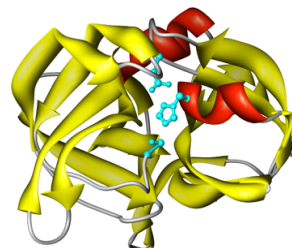


Bovine trypsin (5ptp)
 Structure: $E() < 10^{-23}$;
 RMSD 0.0 Å
 Sequence: $E() < 10^{-84}$
 100% 223/223

Homology \Leftrightarrow structural similarity
 ? sequence similarity



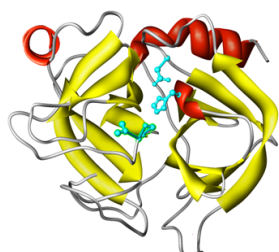
S. griseus trypsin (1sgt)
 $E() < 10^{-14}$ RMSD 1.6 Å
 $E() < 10^{-19}$ 36%; 226/223



S. griseus protease A (2sga)
 $E() < 10^{-4}$; RMSD 2.6 Å
 $E() < 2.6$ 25%; 199/181

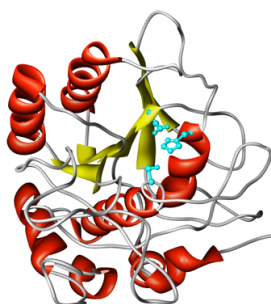
7

When can we infer non-homology?



Bovine trypsin (5ptp)
 Structure: $E() < 10^{-23}$
 RMSD 0.0 Å
 Sequence: $E() < 10^{-84}$
 100% 223/223

Non-homologous proteins have
 different structures



Subtilisin (1sbt)
 $E() > 100$
 $E() < 280$; 25% 159/275



Cytochrome c4 (1etp)
 $E() > 100$
 $E() < 5.5$; 23% 171/190

8

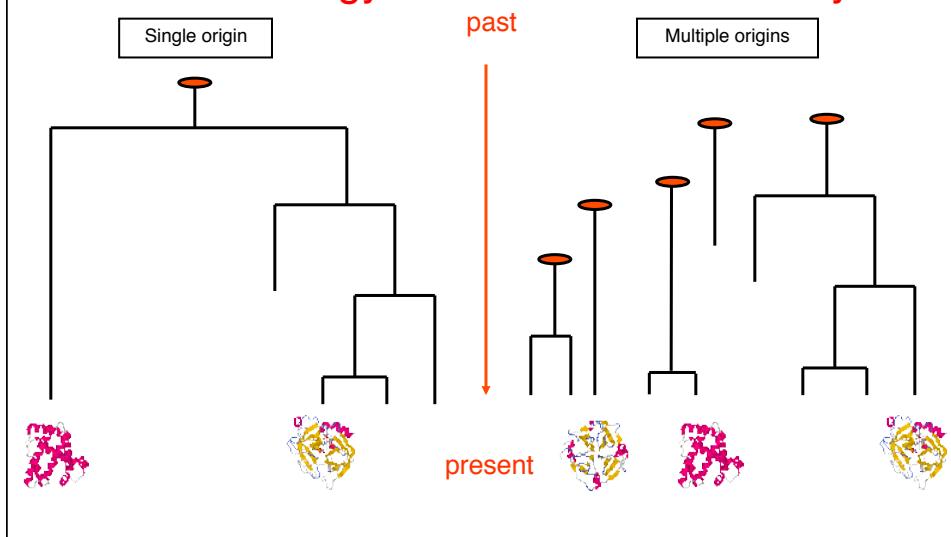
Homology is confusing I: Homology defined Three(?) Ways

- Proteins/genes/DNA that share a common ancestor
- Specific positions/columns in a multiple sequence alignment that have a 1:1 relationship over evolutionary history
 - sequences are *50% homologous* ???
- Specific (morphological/functional) characters that share a recent divergence (clade)
 - bird/bat/butterfly wings are/are not homologous

9

Homology is confusing II: Are All Sequences Homologous?

No Homology without excess similarity



Homology from (sequence/structure) similarity

- Sequences are inferred to share a common ancestor based on statistically significant *excess* similarity. Any evidence of *excess* similarity can be used to infer homology
- Lack of evidence *cannot* be used to infer non-homology.
 - Proteins with different structures are non-homologous
- There are always two alternative hypotheses: homology (common ancestry), or independence – one must weigh the evidence for each hypothesis (independence is the *null* hypothesis).

11

What BLAST does:

Similarity $\overset{?}{\rightleftharpoons}$ Homology

Why BLAST works:

Statistical $\overset{?}{\rightleftharpoons}$ Biological
Significance \rightleftharpoons Significance

Divergence $\overset{?}{\rightleftharpoons}$ Convergence

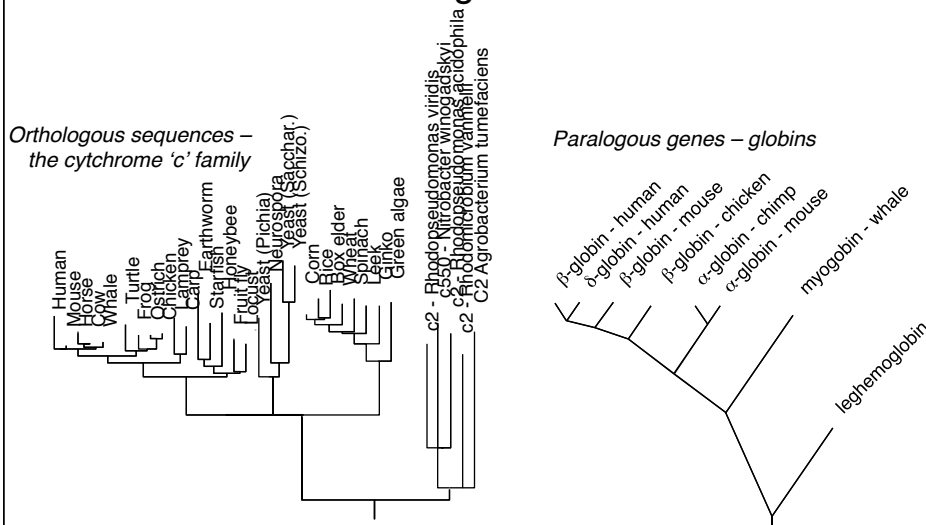
12

E. coli proteins vs Human – Ancient Protein Domains

expect	%_id	alen	E coli descr	Human descr	sp_name
2.7e-206	53.8	944	glycine decarboxylase, P	Glycine dehydrogenase [de	GCSP_HUMAN
1.2e-176	59.5	706	methylmalonyl-CoA mutase	Methylmalonyl-CoA mutase,	MUTA_HUMAN
3.8e-176	50.6	803	glycogen phosphorylase [E	Glycogen phosphorylase, 1	PHS1_HUMAN
9.9e-173	55.6	1222	B12-dependent homocystein	5-methyltetrahydrofolate-	METH_HUMAN
1.8e-165	41.8	1031	carbamoyl-phosphate synth	Carbamoyl-phosphate synth	CPSM_HUMAN
5.6e-159	65.7	542	glucosephosphate isomeras	Glucose-6-phosphate isome	G6PI_HUMAN
8.1e-143	53.7	855	aconitate hydratase 1 [Esch	Iron-responsive element b	IRE1_HUMAN
2.5e-134	73.0	459	membrane-bound ATP syntha	ATP synthase beta chain,	ATPB_HUMAN
3.3e-121	55.8	550	succinate dehydrogenase,	Succinate dehydrogenase [DHSA_HUMAN
1.5e-113	60.6	401	putative aminotransferase	Cysteine desulfurase, mit	NFS1_HUMAN
4.4e-111	60.9	460	fumarase C= fumarate hydr	Fumarate hydratase, mitoc	FUMH_HUMAN
1.5e-109	56.1	474	succinate-semialdehyde de	Succinate semialdehyde de	SSDH_HUMAN
3.6e-106	44.7	789	maltodextrin phosphorylas	Glycogen phosphorylase, m	PHS2_HUMAN
1.4e-102	53.1	484	NAD+-dependent betaine al	Aldehyde dehydrogenase, E	DHAG_HUMAN
3.8e-98	53.0	449	pyridine nucleotide trans	NAD(P) transhydrogenase,	NNTM_HUMAN
5.8e-96	49.9	489	glycerol kinase [Escheric	Glycerol kinase, testis s	GKP2_HUMAN
2.1e-95	66.8	328	glyceraldehyde-3-phospat	Glyceraldehyde 3-phospat	G3P2_HUMAN
5.0e-91	62.5	368	alcohol dehydrogenase cla	Alcohol dehydrogenase cla	ADHX_HUMAN
6.7e-91	56.5	393	protein chain elongation	Elongation factor Tu, mit	EFTU_HUMAN
9.5e-91	56.6	392	protein chain elongation	Elongation factor Tu, mit	EFTU_HUMAN
2.2e-89	59.1	369	methionine adenosyltransf	S-adenosylmethionine synt	METK_HUMAN
6.5e-88	53.3	422	enolase [Escherichia coli	Alpha enolase (2-phospho-	ENOA_HUMAN
9.2e-88	43.3	536	NAD-linked malate dehydro	NADP-dependent malic enzy	MAOX_HUMAN
7.3e-86	55.5	389	2-amino-3-ketobutyrate Co	2-amino-3-ketobutyrate co	KBL_HUMAN
5.2e-83	44.4	543	degrades sigma32, integra	AFG3-like protein 2 (Para	AF32_HUMAN

13

Orthologs and Paralogs – Inferring Function

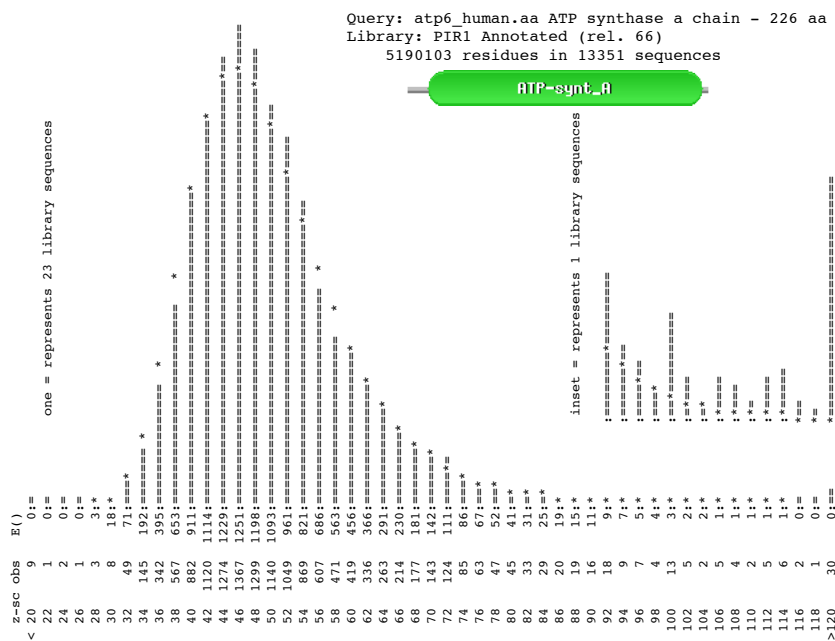


14

Protein Evolution and Sequence Similarity

- What is Homology and how do we recognize it?
- How do we measure sequence similarity – alignments and scoring matrices?
- DNA vs protein comparison
- Alignment Algorithms/Local sequence alignments
- Similarity scoring matrices
- When are we certain that an alignment is significant - similarity score statistics?
- When to trust similarity statistics?
- Improving sensitivity with PSI-BLAST

15



16

Inferring Homology from Statistical Significance

- Real **UNRELATED** sequences have similarity scores that are indistinguishable from **RANDOM** sequences
- If a similarity is NOT **RANDOM**, then it must be NOT **UNRELATED**
- Therefore, NOT **RANDOM** (statistically significant) similarity must reflect **RELATED** sequences

17

```

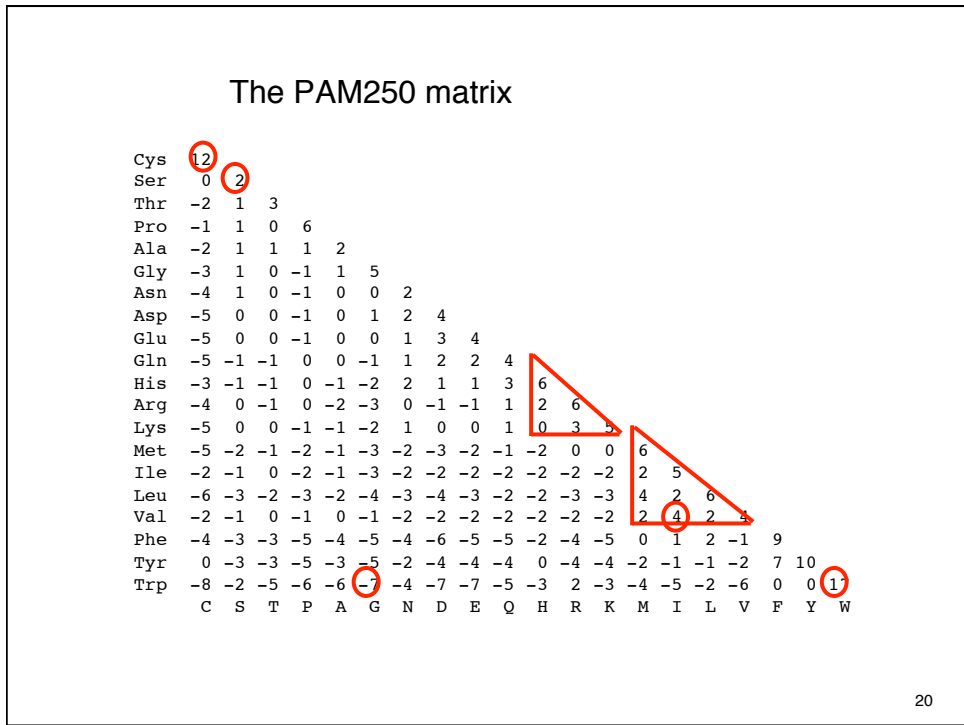
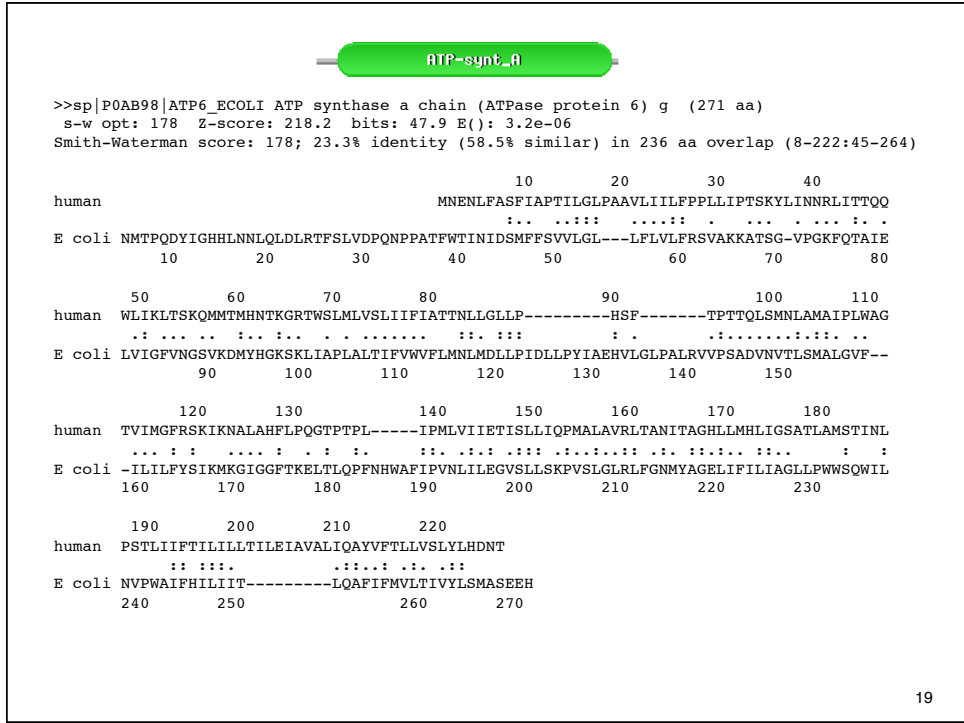
Query: atp6_human.aa ATP synthase a chain - 226 aa
Library: 5190103 residues in 13351 sequences
The best scores are:
      ( len)  s-w bits E(13351)  %_id  %_sim  alen
sp|P00846|ATP6_HUMAN ATP synthase a chain (AT ( 226) 1400 325.8 5.8e-90 1.000 1.000 226
sp|P00847|ATP6_BOVIN ATP synthase a chain (AT ( 226) 1157 270.5 2.5e-73 0.779 0.951 226
sp|P00848|ATP6_MOUSE ATP synthase a chain (AT ( 226) 1118 261.7 1.2e-70 0.757 0.916 226
sp|P00849|ATP6_XENLA ATP synthase a chain (AT ( 226) 745 176.8 4.0e-45 0.533 0.847 229
sp|P00851|ATP6_DROYA ATP synthase a chain (AT ( 224) 473 115.0 1.7e-26 0.378 0.721 222
sp|P00854|ATP6_YEAST ATP synthase a chain pre ( 259) 428 104.7 2.3e-23 0.353 0.694 232
sp|P00852|ATP6_EMENI ATP synthase a chain pre ( 256) 365 90.4 4.8e-19 0.304 0.691 230
sp|P14862|ATP6_COCHE ATP synthase a chain (AT ( 257) 353 87.7 3.2e-18 0.313 0.650 214
sp|P68526|ATP6_TRITI ATP synthase a chain (AT ( 386) 309 77.6 5.1e-15 0.289 0.651 235
sp|P05499|ATP6_TOBAC ATP synthase a chain (AT ( 395) 309 77.6 5.2e-15 0.283 0.635 233
sp|P07925|ATP6_MAIZE ATP synthase a chain (AT ( 291) 283 71.7 2.3e-13 0.311 0.667 180
sp|P0AB98|ATP6_ECOLI ATP synthase a chain (AT ( 271) 178 47.9 3.2e-06 0.233 0.585 236
sp|P0C2Y5|ATPI_ORYSA Chloroplast ATP synth (A ( 247) 144 40.1 0.00062 0.242 0.580 231
sp|P06452|ATPI_PEA Chloroplast ATP synthase a ( 247) 143 39.9 0.00072 0.250 0.586 232
sp|P27178|ATP6_SYNY3 ATP synthase a chain (AT ( 276) 142 39.7 0.00095 0.265 0.571 170
sp|P06451|ATPI_SPIOL Chloroplast ATP synthase ( 247) 138 38.8 0.0016 0.242 0.580 231
sp|P08444|ATP6_SYN6 ATP synthase a chain (AT ( 261) 127 36.3 0.0095 0.263 0.557 167
sp|P69371|ATPI_ATRBE Chloroplast ATP synthase ( 247) 126 36.0 0.01 0.221 0.571 231
sp|P06289|ATPI_MARPO Chloroplast ATP synthase ( 248) 126 36.0 0.011 0.240 0.575 167
sp|P30391|ATPI_EUGGR Chloroplast ATP synthase ( 251) 123 35.4 0.017 0.257 0.579 214

sp|P19568|TLCA_RICPR ADP,ATP carrier protein ( 498) 122 35.0 0.043 0.243 0.579 152

sp|P24966|CYB_TAYTA Cytochrome b ( 379) 113 33.0 0.13 0.234 0.532 158
sp|P03892|NU2M_BOVIN NADH-ubiquinone oxidored ( 347) 107 31.7 0.31 0.261 0.479 211
sp|P68092|CYB_STEAT Cytochrome b ( 379) 104 31.0 0.54 0.277 0.547 137
sp|P03891|NU2M_HUMAN NADH-ubiquinone oxidored ( 347) 103 30.8 0.58 0.201 0.537 149
sp|P00156|CYB_HUMAN Cytochrome b ( 380) 102 30.5 0.74 0.268 0.585 205
sp|P15993|AROP_ECOLI Aromatic amino acid tr ( 457) 103 30.7 0.78 0.234 0.622 111
sp|P24965|CYB_TRANA Cytochrome b ( 379) 101 30.3 0.87 0.234 0.563 158
sp|P29631|CYB_POMTE Cytochrome b ( 308) 99 29.9 0.95 0.274 0.584 113
sp|P24953|CYB_CAPHI Cytochrome b ( 379) 99 29.8 1.2 0.236 0.564 140

```

18



```

>>sp|P30391|ATPI_EUGGR Chloroplast ATP synthase a chain precursor (251 aa)
s-w opt: 123 Z-score: 151.3 bits: 35.4 E(): 0.017
Smith-Waterman score: 123; 25.7% identity (57.9% similar) in 214 aa overlap (21-222:50-243)

                10      20      30      40      50      60
human           M N E N L F A S F I A P T I L G L P A A V L I I L F P P L L I P T S K Y L I N N R L I T T Q Q W L I K L T S K Q M M T M
                .....: : : : : : . . . . . : : : : : : . . . . .
Euglena V M N F I S G I F Q I A N V E V G Q H F Y W S I L G F Q I H G Q V L I N S W I V I L I I G F -- L S I Y T T K N L -- T L V P A N K Q I F I E L V T E F I T D I
              10      20      30      40      50      60      70      80

                70      80      90      100     110     120
human    H N T K - G R T - - - - W S L M L V S L I I F I A T T N L L G - L L P H S F T - - P T T Q L - - - S M N L A M A I P L W A G T V I M G F R S K I - K N A L A H F
              . : . : . . . . . : . . . . . : : : . . : : : . . : : : . . : : : . . : : : . . : : : . . : : : . . : : : . .
Euglena  S K T Q I G E K E Y S K W V P Y I G T M F L F I F V S N W S G A L I P W K I I E L P N G E L G A P T N D I N T A G L A I L T S L A Y F Y A G L N K K G L T Y F
              90      100     110     120     130     140     150     160

                130     140     150     160     170     180     190     200
Human    L P Q G T P T P L I P M L V I I E T I S L L I Q P M A L A V R L T A N I T A G H L L M H L I G S A T L A M S T I N L P S T L I I F T I L I L L T I L E I A V A L
              . : : . . . : . . . . . : : . . . : . . . . . : : : . . . : : . . . : . . . . . : : . . . : . . . . . : . . . . .
Euglena  K K Y V Q P T P I L L P I N I L E D F T - - - K P L S L S F R L F G N I L A D E L V V A V L V S L - - - - - - V P - - L I V P V P L I F L G L F - - - T S G
              170     180     190     200     210     220

                210     220
human    I Q A Y V F T L L V S L Y L H D N T
              : : . . : . .
Euglena  I Q A L I F A T L S G S Y I G E A M E G H H
              230     240     250

```

21

```

Query: atp6_human.aa ATP synthase a chain - 226 aa
Library: 5190103 residues in 13351 sequences

The best scores are:

|                                                        | ( len ) | s-w   | bits    | E(13351) | %_id  | %_sim | alen |
|--------------------------------------------------------|---------|-------|---------|----------|-------|-------|------|
| sp P00846 ATP6_HUMAN ATP synthase a chain ( AT ( 226 ) | 1400    | 325.8 | 5.8e-90 | 1.000    | 1.000 | 226   |      |
| sp P00847 ATP6_BOVIN ATP synthase a chain ( AT ( 226 ) | 1157    | 270.5 | 2.5e-73 | 0.779    | 0.951 | 226   |      |
| sp P00848 ATP6_MOUSE ATP synthase a chain ( AT ( 226 ) | 1118    | 261.7 | 1.2e-70 | 0.757    | 0.916 | 226   |      |
| sp P00849 ATP6_XENLA ATP synthase a chain ( AT ( 226 ) | 745     | 176.8 | 4.0e-45 | 0.533    | 0.847 | 229   |      |
| sp P00851 ATP6_DROYA ATP synthase a chain ( AT ( 224 ) | 473     | 115.0 | 1.7e-26 | 0.378    | 0.721 | 222   |      |
| sp P00854 ATP6_YEAST ATP synthase a chain pre ( 259 )  | 428     | 104.7 | 2.3e-23 | 0.353    | 0.694 | 232   |      |
| sp P00852 ATP6_EMENI ATP synthase a chain pre ( 256 )  | 365     | 90.4  | 4.8e-19 | 0.304    | 0.691 | 230   |      |
| sp P14862 ATP6_COCHE ATP synthase a chain ( AT ( 257 ) | 353     | 87.7  | 3.2e-18 | 0.313    | 0.650 | 214   |      |
| sp P68526 ATP6_TRITI ATP synthase a chain ( AT ( 386 ) | 309     | 77.6  | 5.1e-15 | 0.289    | 0.651 | 235   |      |
| sp P05499 ATP6_TOBAC ATP synthase a chain ( AT ( 395 ) | 309     | 77.6  | 5.2e-15 | 0.283    | 0.635 | 233   |      |
| sp P07925 ATP6_MAIZE ATP synthase a chain ( AT ( 291 ) | 283     | 71.7  | 2.3e-13 | 0.311    | 0.667 | 180   |      |
| sp P0AB98 ATP6_ECOLI ATP synthase a chain ( AT ( 271 ) | 178     | 47.9  | 3.2e-06 | 0.233    | 0.585 | 236   |      |
| sp P0C2Y5 ATPI_ORYSA Chloroplast ATP synth ( A ( 247 ) | 144     | 40.1  | 0.00062 | 0.242    | 0.580 | 231   |      |
| sp P06452 ATPI_PEA Chloroplast ATP synthase a ( 247 )  | 143     | 39.9  | 0.00072 | 0.250    | 0.586 | 232   |      |
| sp P27178 ATP6_SYNY3 ATP synthase a chain ( AT ( 276 ) | 142     | 39.7  | 0.00095 | 0.265    | 0.571 | 170   |      |
| sp P06451 ATPI_SPIOL Chloroplast ATP synthase ( 247 )  | 138     | 38.8  | 0.0016  | 0.242    | 0.580 | 231   |      |
| sp P08444 ATP6_SYN6 ATP synthase a chain ( AT ( 261 )  | 127     | 36.3  | 0.0095  | 0.263    | 0.557 | 167   |      |
| sp P69371 ATPI_ATTRBE Chloroplast ATP synthase ( 247 ) | 126     | 36.0  | 0.01    | 0.221    | 0.571 | 231   |      |
| sp P06289 ATPI_MARPO Chloroplast ATP synthase ( 248 )  | 126     | 36.0  | 0.011   | 0.240    | 0.575 | 167   |      |
| sp P30391 ATPI_EUGGR Chloroplast ATP synthase ( 251 )  | 123     | 35.4  | 0.017   | 0.257    | 0.579 | 214   |      |
|                                                        |         |       |         |          |       |       |      |
| sp P19568 TLCA_RICPR ADP,ATP carrier protein ( 498 )   | 122     | 35.0  | 0.043   | 0.243    | 0.579 | 152   |      |
|                                                        |         |       |         |          |       |       |      |
| sp P24966 CYB_TAYTA Cytochrome b ( 379 )               | 113     | 33.0  | 0.13    | 0.234    | 0.532 | 158   |      |
| sp P03892 NU2M_BOVIN NADH-ubiquinone oxidored ( 347 )  | 107     | 31.7  | 0.31    | 0.261    | 0.479 | 211   |      |
| sp P68092 CYB_STEAT Cytochrome b ( 379 )               | 104     | 31.0  | 0.54    | 0.277    | 0.547 | 137   |      |
| sp P03891 NU2M_HUMAN NADH-ubiquinone oxidored ( 347 )  | 103     | 30.8  | 0.58    | 0.201    | 0.537 | 149   |      |
| sp P00156 CYB_HUMAN Cytochrome b ( 380 )               | 102     | 30.5  | 0.74    | 0.268    | 0.585 | 205   |      |
| sp P15993 AROP_ECOLI Aromatic amino acid tr ( 457 )    | 103     | 30.7  | 0.78    | 0.234    | 0.622 | 111   |      |
| sp P24965 CYB_TRANA Cytochrome b ( 379 )               | 101     | 30.3  | 0.87    | 0.234    | 0.563 | 158   |      |
| sp P29631 CYB_POMTE Cytochrome b ( 308 )               | 99      | 29.9  | 0.95    | 0.274    | 0.584 | 113   |      |
| sp P24953 CYB_CAPHI Cytochrome b ( 379 )               | 99      | 29.8  | 1.2     | 0.236    | 0.564 | 140   |      |


```

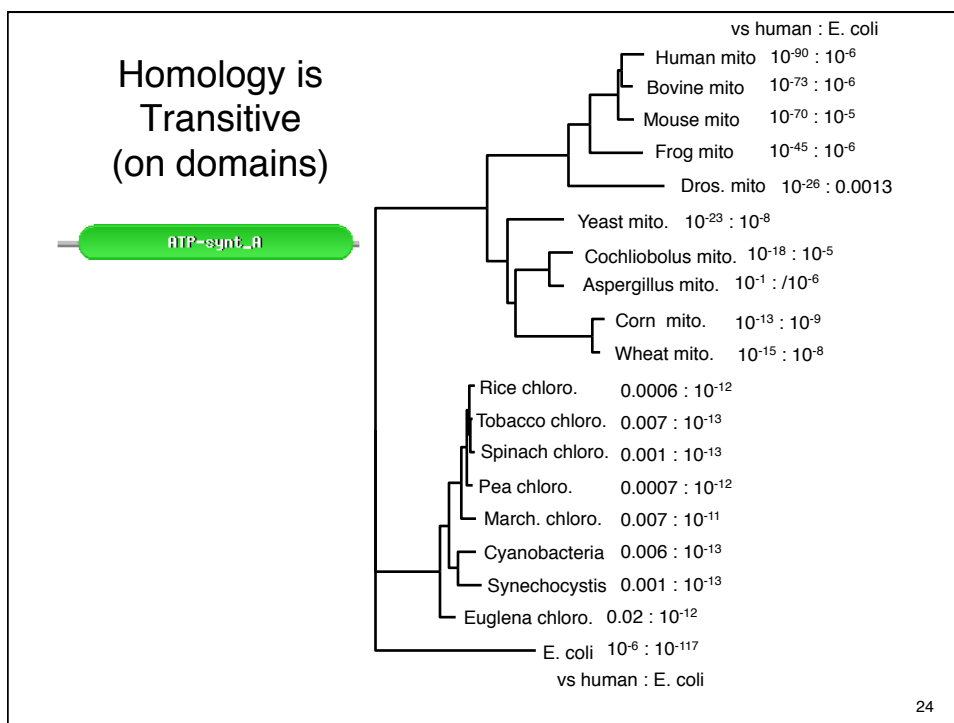
22

Query: atp6_ecoli.aa ATP synthase a - 271 aa
Library: 5190103 residues in 13351 sequences

The best scores are:

	(len)	s-w bits	E(13351)	%_id	%_sim	alen	
sp P0AB98 ATP6_ECOLI	ATP synthase a chain (AT (271)	1774	416.8	3.e-117	1.000	1.000	271
sp P06451 ATPI_SPIOL	Chloroplast ATP synthase (247)	274	70.4	5.8e-13	0.270	0.616	211
sp P69371 ATPI_ATRBE	Chloroplast ATP synthase (247)	271	69.7	9.3e-13	0.270	0.607	211
sp P08444 ATP6_SYNP6	ATP synthase a chain (AT (261)	271	69.7	9.9e-13	0.267	0.600	240
sp P06452 ATPI_PEA	Chloroplast ATP synthase a (247)	266	68.5	2.1e-12	0.274	0.614	223
sp P30391 ATPI_EUGGR	Chloroplast ATP synthase (251)	265	68.3	2.5e-12	0.298	0.596	225
sp P0C2Y5 ATPI_ORYSA	Chloroplast ATP synthase (247)	260	67.2	5.4e-12	0.259	0.603	239
sp P27178 ATP6_SYNY3	ATP synthase a chain (AT (276)	260	67.1	6.1e-12	0.264	0.578	258
sp P06289 ATPI_MARPO	Chloroplast ATP synthase (248)	250	64.8	2.7e-11	0.261	0.621	211
sp P07925 ATP6_MAIZE	ATP synthase a chain (AT (291)	215	56.7	8.7e-09	0.259	0.578	232
sp P68526 ATP6_TRITI	ATP synthase a chain (AT (386)	209	55.3	3.1e-08	0.259	0.603	239
sp P00854 ATP6_YEAST	ATP synthase a chain pre (259)	204	54.2	4.5e-08	0.235	0.578	277
sp P05499 ATP6_TOBAC	ATP synthase a chain (AT (395)	189	50.7	7.8e-07	0.220	0.582	268
sp P00846 ATP6_HUMAN	ATP synthase a chain (AT (226)	178	48.2	2.5e-06	0.237	0.589	236
sp P00852 ATP6_EMENI	ATP synthase a chain pre (256)	178	48.2	2.8e-06	0.209	0.590	244
sp P00849 ATP6_XENLA	ATP synthase a chain (AT (226)	173	47.1	5.5e-06	0.261	0.630	165
sp P00847 ATP6_BOVIN	ATP synthase a chain (AT (226)	172	46.8	6.5e-06	0.233	0.581	236
sp P14862 ATP6_COCHE	ATP synthase a chain (AT (257)	171	46.6	8.7e-06	0.204	0.608	265
sp P00848 ATP6_MOUSE	ATP synthase a chain (AT (226)	166	45.5	1.7e-05	0.259	0.617	193
sp P00851 ATP6_DROYA	ATP synthase a chain (AT (224)	139	39.2	0.0013	0.225	0.549	253
sp P24962 CYB_STELO	Cytochrome b (379)	125	35.9	0.021	0.223	0.575	193
sp P09716 US17_HCMVA	Hypothetical protein HVL (293)	109	32.3	0.21	0.260	0.565	131
sp P68092 CYB_STEAT	Cytochrome b (379)	109	32.2	0.27	0.211	0.562	194
sp P24960 CYB_ODOHE	Cytochrome b (379)	104	31.1	0.61	0.210	0.555	200
sp P03887 NULM_BOVIN	NADH-ubiquinone oxidored (318)	98	29.7	1.3	0.287	0.545	167
sp P24992 CYB_ANTAM	Cytochrome b (379)	99	29.9	1.4	0.192	0.565	193

23



Homology and Domains – Histone deacetylase PCAF

The best scores are:

		s-w bits	E(362341)	%_id	%_sim	alen
PCAF_HUMAN	Histone acetyltransferase PCAF;	(832)	4876 1092	0	1.000	832
PCAF_MOUSE	Histone acetyltransferase PCAF;	(813)	4507 1010	0	0.929	0.974 817
GCNL2_HUMAN	General control of amino acid synthesis protein 5-l	(837)	3535 793.	0	0.716	0.864 821
GCN5_YEAST	Histone acetyltransferase GCN5	(439)	1049 240.	5.2e-62	0.469	0.743 354
GCN5_ARATH	Histone acetyltransferase GCN5; AtGCN5	(568)	956 219.	1.2e-55	0.435	0.733 375
BPTF_HUMAN	Nucleosome-remodeling factor subunit BPTF	(3046)	369 88.3	2.4e-15	0.495	0.773 97
NU301_DROME	Nucleosome-remodeling factor subunit NURF301	(2669)	359 86.2	9.3e-15	0.511	0.787 94
CECR2_HUMAN	Cat eye syndrome critical region protein 2	(1484)	306 74.6	1.6e-11	0.371	0.771 105
BRD4_HUMAN	Bromodomain-containing protein 4; HUNK1 protein	(1362)	288 70.6	2.3e-10	0.379	0.681 116
BRDT_MACFA	Bromodomain testis-specific protein	(947)	270 66.7	2.3e-09	0.353	0.690 116
FSH_DROME	Homeotic protein female sterile; Fragile-chorion memb	(2038)	276 67.8	2.4e-09	0.341	0.651 129
BRDT_HUMAN	Bromodomain testis-specific protein; RING3-like prot	(947)	266 65.9	4.3e-09	0.345	0.690 116
Y0777_DICDI	Bromodomain-containing protein DDB_G0280777	(1823)	260 64.3	2.5e-08	0.385	0.725 109
BRDT_MOUSE	Bromodomain testis-specific protein; RING3-like prot	(956)	247 61.6	8.1e-08	0.328	0.647 116
BAZ2B_HUMAN	Bromodomain adjacent to zinc finger domain protein	(1972)	247 61.3	2e-07	0.343	0.695 105
TAF1_DROME	Transcription initiation factor TFIID subunit 1; Tra	(2129)	230 57.5	3.1e-06	0.349	0.689 106
B2_SCHPO	Bromodomain-containing protein C631.02	(727)	217 55.0	5.9e-06	0.320	0.587 172
BRD9_XENLA	Bromodomain-containing protein 9	(527)	214 54.5	6.2e-06	0.292	0.579 171
GTE6_ARATH	Transcription factor GTE6; Protein GENERAL TRANSCRIP	(369)	201 51.7	2.9e-05	0.290	0.601 183
BAZ1B_MOUSE	Bromodomain adjacent to zinc finger domain protein	(1479)	212 53.7	3.1e-05	0.302	0.583 139
K2_SCHPO	Bromodomain-containing protein C1450.02	(578)	204 52.2	3.3e-05	0.310	0.628 113
TAF1_HUMAN	Transcription initiation factor TFIID subunit 1; Tra	(1872)	212 53.6	4.2e-05	0.339	0.678 115
BAZ1B_HUMAN	Bromodomain adjacent to zinc finger domain protein	(1483)	209 53.0	5e-05	0.397	0.705 78
TIF1A_HUMAN	Transcription intermediary factor 1-alpha; TIF1-al	(1050)	206 52.5	5.1e-05	0.384	0.698 86
BDF2_YEAST	Bromodomain-containing factor 2	(638)	200 51.3	6.9e-05	0.304	0.607 168

25

Homology and Domains – Histone deacetylase PCAF

The best scores are:

		E(362341)	alen
PCAF_HUMAN	Histone acetyl (832)	0	832
GCN5_YEAST	Histone acetyl (439)	5.2e-62	354
BPTF_HUMAN	Nucleosome-rem (3046)	2.4e-15	97
CECR2_HUMAN	Cat eye syndr (1484)	1.6e-11	105
GTE6_ARATH	Transcription (369)	2.9e-05	183

26

Protein Evolution and Sequence Similarity

- What is Homology and how do we recognize it?
- How do we measure sequence similarity – alignments and scoring matrices?
- DNA vs protein comparison
- Alignment Algorithms/Local sequence alignments
- Similarity scoring matrices
- When are we certain that an alignment is significant - similarity score statistics?
- When to trust similarity statistics?
- Improving sensitivity with PSI-BLAST

27

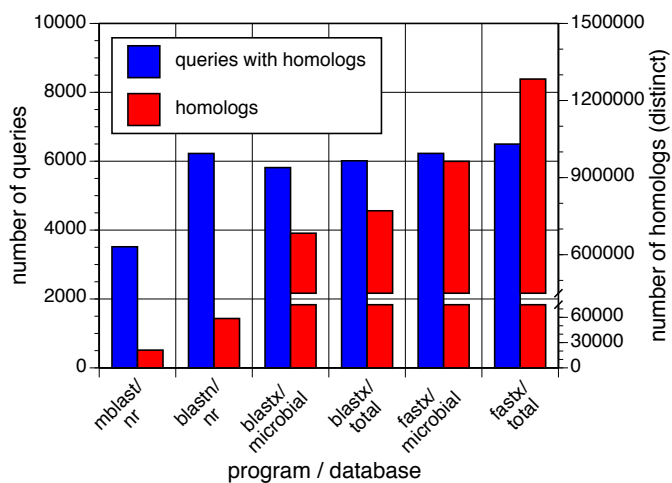
DNA vs protein sequence comparison

The best scores are:

		DNA E(188,018)	tfastx3 E(187,524)	prot. E(331,956)
DMGST	D.melanogaster GST1-1	1.3e-164	4.1e-109	1.0e-109
MDGST1	M.domestica GST-1 gene	2e-77	3.0e-95	1.9e-76
LUCGLTR	Lucilia cuprina GST	1.5e-72	5.2e-91	3.3e-73
MDGST2A	M.domesticus GST-2 mRNA	9.3e-53	1.4e-77	1.6e-62
MDNF1	M.domestica nf1 gene. 10	4.6e-51	2.8e-77	2.2e-62
MDNF6	M.domestica nf6 gene. 10	2.8e-51	4.2e-77	3.1e-62
MDNF7	M.domestica nf7 gene. 10	6.1e-47	9.2e-77	6.7e-62
AGGST15	A.gambiae GST mRNA	3.1e-58	4.2e-76	4.3e-61
CVU87958	Culicoides GST	1.8e-41	4.0e-73	3.6e-58
AGG3GST11	A.gambiae GST1-1 mRNA	1.5e-46	2.8e-55	1.1e-43
BMO6502	Bombyx mori GST mRNA	1.1e-23	8.8e-50	5.7e-40
AGSUGST12	A.gambiae GST1-1 gene	2.3e-16	4.5e-46	5.1e-37
MOTGLUSTRA	Manduca sexta GST	5.7e-07	2.5e-30	8.0e-25
RLGSTARGN	R.leguminosarum <i>gsta</i>	0.0029	3.2e-13	1.4e-10
HUMGSTT2A	H. sapiens GSTT2	0.32	3.3e-10	2.0e-09
HSGSTT1	H.sapiens GSTT1 mRNA	7.2	8.4e-13	3.6e-10
ECAE000319	E. coli hypothet. prot.	—	4.7e-10	1.1e-09
MYMDCMA	Methyl. dichlorometh. DH	—	1.1e-09	6.9e-07
BCU19883	Burkholderia maleylacetate red.	—	1.2e-09	1.1e-08
NFU43126	Naegleria fowleri GST	—	3.2e-07	0.0056
SP505GST	Sphingomonas paucim	—	1.8e-06	0.0002
EN1838	H. sapiens maleylaceto. iso.	—	2.1e-06	5.9e-06
HSU86529	Human GSTZ1	—	3.0e-06	8.0e-06
SYCCPNC	Synechocystis GST	—	1.2e-05	9.5e-06
HSEF1GMR	H.sapiens EF1g mRNA	—	9.0e-05	0.00065

28

Improving search strategies (windshield splatter metagenomics)



- always use protein/translated DNA comparisons
- smaller databases are more sensitive

Similarity Searching II

1. What question to ask?
2. What program to use?
3. What database to search?
4. How to avoid mistakes (what to look out for)
5. When to do something different (changing scoring matrices)

1. What question to ask?

- Is there an homologous protein (a protein with a similar structure)?
- Does that homologous protein have a similar function?
- Does XXX genome have YYY (kinase, GPCR, ...)?

Questions not to ask:

- Does this DNA sequence have a similar regulatory element (too short – never significant)?
- Does (non-significant) protein have a similar function/modification/antigenic site?

31

2. What program to run?

- What is your query sequence?
 - protein – BLAST (NCBI), SSEARCH (EBI)
 - protein coding DNA (EST) – BLASTX (NCBI), FASTX (EBI)
 - DNA (structural RNA, repeat family) – BLASTN (NCBI), FASTA (EBI)
- Does XXX genome have YYY (protein)?
 - TBLASTN YYY vs XXX genome
 - TFASTX YYY vs XXX genome
- Does my protein contain repeated domains?
 - LALIGN (UVa <http://fasta.bioch.virginia.edu>)

32

NCBI BLAST Server

blast.ncbi.nlm.nih.gov

BLAST Basic Local Alignment Search Tool

Home Recent Results Saved Strategies Help

NCBI/ BLAST Home

BLAST finds regions of similarity between biological sequences. [more...](#)

New Aligning Multiple Protein Sequences? Try the **COBALT** Multiple Alignment Tool.

BLAST Assembled Genomes

Choose a species genome to search, or [list all genomic BLAST databases](#).

- [Human](#)
- [Mouse](#)
- [Rat](#)
- [Arabidopsis thaliana](#)
- [Oryza sativa](#)
- [Bos taurus](#)
- [Danio rerio](#)
- [Drosophila melanogaster](#)
- [Gallus gallus](#)
- [Pan troglodytes](#)
- [Microbes](#)
- [Apis mellifera](#)

Basic BLAST

Choose a BLAST program to run.

nucleotide blast	Search a nucleotide database using a nucleotide query <i>Algorithms: blastn, megablast, discontinuous megablast</i>
protein blast	Search protein database using a protein query <i>Algorithms: blastp, psi-blast, phi-blast</i>
blastx	Search protein database using a translated nucleotide query
tblastn	Search translated nucleotide database using a protein query
tblastx	Search translated nucleotide database using a translated nucleotide query

Specialized BLAST

Choose a type of specialized search (or database name in parentheses.)

- Make specific primers with [Primer-BLAST](#)
- Search [trace archives](#)
- Find [conserved domains](#) in your sequence (cds)
- Find sequences with similar [conserved domain architecture](#) (cdart)

NCBI BLAST Server

blast.ncbi.nlm.nih.gov

Basic BLAST

Choose a BLAST program to run.

nucleotide blast	Search a nucleotide database using a nucleotide query <i>Algorithms: blastn, megablast, discontinuous megablast</i>
protein blast	Search protein database using a protein query <i>Algorithms: blastp, psi-blast, phi-blast</i>
blastx	Search protein database using a translated nucleotide query
tblastn	Search translated nucleotide database using a protein query
tblastx	Search translated nucleotide database using a translated nucleotide query

What is wrong with this picture?

Always compare protein sequences

NCBI BLAST Server

The screenshot shows the NCBI BLAST Basic Local Alignment Search Tool interface. The top navigation bar includes 'Home', 'Recent Results', 'Saved Strategies', and 'Help'. Below this, there are tabs for 'blastn', 'blastp', 'blastx', 'tblastn', and 'tblastx'. The main section is titled 'Enter Query Sequence' and contains a large text input field for 'Enter accession number, gi, or FASTA sequence'. To the right of this field are 'Clear' and 'Query subrange' options with 'From' and 'To' input boxes. Below the main input field, there is a section for 'Or, upload file' with a 'Choose File' button and 'no file selected' text. A 'Job Title' field is also present. A checkbox option 'Align two or more sequences' is available. The 'Choose Search Set' section includes a 'Database' dropdown menu set to 'Non-redundant protein sequences (nr)', an 'Organism' field with a search icon and an 'Exclude' checkbox, and an 'Entrez Query' field. The 'Program Selection' section has radio buttons for 'blastp (protein-protein BLAST)', 'PSI-BLAST (Position-Specific Iterated BLAST)', and 'PHI-BLAST (Pattern Hit Initiated BLAST)'. At the bottom, there is a 'BLAST' button and a 'Search database Non-redundant protein sequences (nr) using Blastp (protein-protein BLAST)' option, with a checkbox for 'Show results in a new window' and a link for 'Algorithm parameters'.

Searching at the EBI www.ebi.ac.uk/Tools/sss/

EBI > Tools > Sequence Similarity Searching

Sequence Similarity Searching

BLAST

NCBI BLAST NCBI BLAST Sequence Similarity Search using the NCBI BLAST (blastall) program. This tool is available for the following databases:

Protein Nucleotide Vectors

WU-BLAST Sequence Similarity Search using the Washington University (WU) BLAST2 program (BLAST 2.0 with gaps). This tool is available for the following databases:

Protein Nucleotide Parasites

PSI-BLAST Position Specific Iterative **BLAST (PSI-BLAST)** refers to a feature of BLAST 2.0 in which a profile is automatically constructed from the first set of BLAST alignments.

Launch PSI-BLAST

FASTA

FASTA Sequence Similarity Search using the FASTA program. This tool is available for the following databases:

Protein Nucleotide Proteomes Genomes Whole Genome Shotgun

ASD Protein ASD Nucleotide LGIC Protein LGIC Nucleotide

SSEARCH Sequence Similarity Search using the SSEARCH program. This tool is available for the following databases:

Protein Nucleotide Proteomes Genomes Whole Genome Shotgun

ASD Protein ASD Nucleotide LGIC Protein LGIC Nucleotide

PSI-Search PSI-Search combines the sensitivity of the Smith-Waterman search algorithm (SSEARCH) with the PSI-BLAST (blastpgp) iterative profile construction strategy to find distantly related protein sequences.

Launch PSI-Search

GGSEARCH GGSEARCH performs a sequence search using alignments that are global in the query and global in the database (Needleman-Wunsch).

Protein Nucleotide

36

Searching at the EBI – ssearch

EBI > Tools > Similarity & Homology

FASTA/SSEARCH/GGSEARCH/GLSEARCH - Protein Similarity Search

Provides sequence similarity searching against protein databases using the FASTA and SSEARCH programs. **SSEARCH** does a rigorous Smith-Waterman search for similarity between a query sequence and a database. **GGSEARCH** compares a protein or DNA sequence to a sequence database producing global-global alignment (Needleman-Wunsch). **GLSEARCH** compares a protein or DNA sequence to a sequence database. **FASTA** can be very specific when identifying long regions of low similarity especially for highly diverged sequences. You can also conduct sequence similarity searching against [nucleotide databases](#) or complete [proteome/genome](#) databases using the [FASTA programs](#).

[Download Software](#)

PROGRAM	DATABASES	RESULTS	SEARCH TITLE	YOUR EMAIL
SSEARCH	Protein	Interactive	Sequence	
	UniProt Knowledgebase UniProtKB/Swiss-Prot UniProt Clusters 100% UniProt Clusters 100% (SEG filter)			
MATRIX	GAP OPEN	GAP EXTEND	EXPECTATION UPPER VALUE	EXPECTATION LOWER VALUE
BLOSUM62	-10	-2	10.0	default
SCORES	ALIGNMENTS	SEQUENCE RANGE	DATABASE RANGE	FILTER
50	50	START-END	START-END	none
				STATISTICAL ESTIMATES
				Regress

Enter or Paste a Sequence in any format:

Upload a file: no file selected

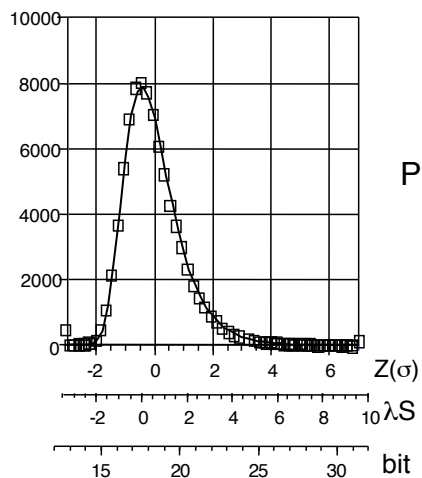
37

3. What database to search?

- Search the smallest comprehensive database likely to contain your protein
 - vertebrates – human proteins (40,000)
 - fungi – *S. cerevisiae* (6,000)
 - bacteria – *E. coli*, gram positive, etc. (<100,000)
- Search a richly annotated protein set (SwissProt, 450,000)
- Always search NR (> 12 million) *LAST*
- Never Search “GenBank” (DNA)

38

Why smaller databases are better – statistics



$$S' = \lambda S_{\text{raw}} - \ln K m n$$

$$S_{\text{bit}} = (\lambda S_{\text{raw}} - \ln K) / \ln(2)$$

$$P(S' > x) = 1 - \exp(-e^{-x})$$

$$P(S_{\text{bit}} > x) = 1 - \exp(-mn2^{-x})$$

$$E(S' > x \text{ ID}) = P D$$

$$P(B \text{ bits}) = m n 2^{-B}$$

$$P(40 \text{ bits}) = 1.5 \times 10^{-7}$$

$$E(40 \mid D=4000) = 6 \times 10^{-4}$$

$$E(40 \mid D=12E6) = 1.8$$

39

Statistical Significance and Database Size

atp6_human vs E. coli
 >>ref|NP_290377.1| F0F1 ATP synthase subunit [E. coli] (271 aa)
 s-w opt: 178 Z-score: 188.8 bits: 42.4 E(): 4.4e-05
 Smith-Waterman score: 178; 23.3% identity (58.5% similar) in 236 aa overlap (8-222:45-264)

Database	Entries	Length	E()	hits	time (s)
E. coli	4,237	1.3 E 06	1.5 E-06*	1	< 0.5
S. cerevisiae	5,866	2.9 E 06	2.1 E-06	1	< 0.5
Human	38,114	18.4 E 06	1.2 E-05	1	1.1
Swiss Prot	4.3 E 05	1.5 E 08	2.4 E-05*	393	7.1
Refseq NP only	7.1 E 05	2.6 E 08	0.00017*	504	10.8
Refseq	7.3 E 06	2.5 E 09	0.0017*	2767	124
NR	9.9 E 06	3.4 E 09	0.0032*	7773	151

40

NCBI – selecting sequences with Entrez

NCBI/ BLAST/ blastp suite

blastn blastp blastx tblastn tblastx

BLASTP programs search protein databases using a protein query. [more...](#)

Enter Query Sequence

Enter accession number, gi, or FASTA sequence [Clear](#) **Query subrange**

From

To

Or, upload file no file selected

Job Title

Enter a descriptive title for your BLAST search

Align two or more sequences

Choose Search Set

Database

Organism Exclude

Optional Enter organism common name, binomial, or tax id. Only 20 top taxa will be shown.

Entrez Query

Optional Enter an Entrez query to limit search

41

Similarity Searching II

1. What question to ask?
2. What program to use?
3. What database to search?
4. How to avoid mistakes (what to look out for)
5. When to do something different

42

Inferring Homology from Statistical Significance

- Real **UNRELATED** sequences have similarity scores that are indistinguishable from **RANDOM** sequences
- If a similarity is NOT **RANDOM**, then it must be NOT **UNRELATED**
- Therefore, NOT **RANDOM** (statistically significant) similarity must reflect **RELATED** sequences

43

Smith-Waterman (ssearch)

The best scores are:

			s-w	bits	E(115640)	%_id	alen
GTM1_MOUSE	Glutathione S-trans	(218)	1497	363.5	2e-100	1.000	218
GTM2_CHICK	Glutathione S-trans	(220)	958	234.9	1.1e-61	0.619	218
GTP_HUMAN	Glutathione S-trans	(210)	356	91.2	1.8e-18	0.308	211
PGD2_MOUSE	Glutathione-req.	(199)	262	68.8	9.7e-12	0.319	204
GTA1_MOUSE	Glutathione S-trans	(223)	229	60.9	2.6e-09	0.284	225
SC1_OCTDO	S-crystallin 1 OL1	(215)	228	60.7	3.0e-09	0.269	219
GTS_MUSDO	Glutathione S-trans	(241)	228	60.6	3.4e-09	0.264	201
GTS1_CAEEL	Prob. Glut. S-trans	(210)	220	58.8	1.1e-08	0.284	225
GTS_OMMSL	Glutathione S-trans	(203)	196	53.0	5.5e-07	0.258	209
GTH3_ARATH	Glutathione S-trans	(215)	142	40.1	0.0045	0.310	126
GTT2_HUMAN	Glutathione S-trans	(244)	132	37.7	0.027	0.257	167
GT24_DROME	Glutathione S-trans	(216)	131	37.5	0.028	0.255	153
YFCG_ECOLI	Hypothetical GST	(215)	112	33.0	0.64	0.235	187
YJY1_YEAST	hypothetical 30.5	(261)	110	32.4	*1.1*	0.248	149
DCMA_METS1	dichloromethane DM	(267)	103	30.8	3.7	0.214	210
YA42_HAEIN	Hypothetical prot.	(617)	108	31.7	*4.6*	0.283	120
GTO1_RAT	Glutathione trans	(241)	100	30.1	5.4	0.234	158
DP41_BACHD	DNA polymerase I	(413)	104	30.8	*5.4*	0.234	184
GTH1_WHEAT	Glutathione S-trans	(229)	98	29.6	7.0	0.246	171
LGUL_SOYBN	Lactoylglutathione	(219)	97	29.4	7.8	0.200	190

Highest scoring unrelated sequence E() ~ 1.0

44

Looking for mistakes; BLASTP at NCBI

NCBI/BLAST/blastp suite BLAST Human Sequences

blastn **blastp** blastx tblastn tblastx

Enter Query Sequence BLASTP programs search protein databases using a protein query. [more...](#)

Enter accession number(s), gi(s), or FASTA sequence(s) Clear Query subrange

121694 From

To

Or, upload file Choose File no file selected

Job Title P20432:RecName: Full=Glutathione S-transferase...

Enter a descriptive title for your BLAST search

Choose Search Set

Database RefSeq protein 29315 sequences

Exclude Models (XM/XP) Uncultured/environmental sample sequences

Entrez Query human[orgn]

Optional Enter an Entrez query to limit search

Program Selection

Algorithm

blastp (protein-protein BLAST)

PSI-BLAST (Position-Specific Iterated BLAST)

PHI-BLAST (Pattern Hit Initiated BLAST)

Choose a BLAST algorithm

BLAST Search database RefSeq protein using Blastp (protein-protein BLAST)

Show results in a new window

45

BLAST results 1 - gstt1_drome

BLAST Results - 7PAPKRC016

1 record to records matching entrez query: human[orgn].

[Search Strategies](#) [Formatting options](#) [Download](#)

1) [gi|594139|P20432.1|GSTT1_DROME](#) Database Name [GP/9606.9558/RefSeq_protein](#)

Accession: P20432.1; Description: Full=Glutathione S-transferase 1-1; AltName: Full=DDT-rochlorinase; AltName: Full=GST class-theta acid; Description: Homo sapiens RefSeq protein; Program: BLASTP 2.2.25+ [Citation](#)

[Summary](#) [Taxonomy reports](#) [Distance tree of results](#) [Multiple alignment](#)

Putative conserved domains have been detected, click on the image below for detailed results.

Protein sequence: 1 25 50 75 100 125 150 175 218

Conserved domains:

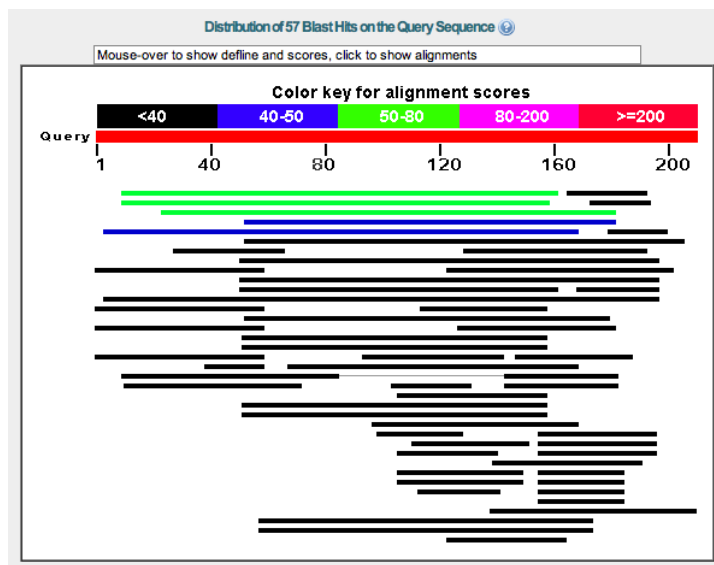
- Thioredoxin-like superfamily** (residues 1-100)
- GST_C_family superfamily** (residues 100-218)

Specific motifs:

- binding site (G-site)
- dimer interface
- substrate binding pocket (G-site)
- N-terminal domain interface

46

BLAST results 2 - gstt1_drome



47

BLAST results 3 - gstt1_drome

Sequences producing significant alignments:

Accession	Description	Max score	Total score	Query coverage	E value	Links
NP_000844.2	glutathione S-transferase theta-1 [Homo sapiens]	76.3	76.3	72%	2e-19	UGM
NP_000845.1	glutathione S-transferase theta-2 [Homo sapiens] >ref[NP_665877.1]	71.6	71.6	71%	1e-17	GM
NP_665877.1	maleylacetoacetate isomerase isoform 1 [Homo sapiens]	56.6	56.6	75%	2e-12	UGM
NP_001504.2	maleylacetoacetate isomerase isoform 3 [Homo sapiens]	47.4	47.4	61%	2e-09	UGM
NP_001503.1	glutathione S-transferase A4 [Homo sapiens]	40.0	40.0	78%	1e-06	UGM
NP_004271.1	eukaryotic translation elongation factor 1 epsilon-1 isoform	38.5	38.5	73%	3e-06	UGM
NP_006294.2	aminoacyl tRNA synthase complex-interacting multifunctio	37.7	37.7	30%	1e-05	UGM
NP_665683.1	glutathione S-transferase A1 [Homo sapiens]	37.4	37.4	69%	1e-05	UGM
NP_036270.1	protein AATF [Homo sapiens]	35.0	35.0	37%	8e-05	UGM
NP_000837.3	glutathione S-transferase A2 [Homo sapiens]	34.7	34.7	69%	8e-05	UGM
NP_000838.3	glutathione S-transferase A3 [Homo sapiens]	33.9	33.9	53%	2e-04	UGM
NP_714543.1	glutathione S-transferase A5 [Homo sapiens]	32.7	32.7	92%	3e-04	UGM
NP_671488.1	blood vessel epicardial substance [Homo sapiens] >ref[NP_001129122.1]	33.1	33.1	21%	4e-04	UGM
NP_001129122.1	eukaryotic translation elongation factor 1 epsilon-1 isoform	31.6	31.6	60%	6e-04	GM
NP_665878.2	maleylacetoacetate isomerase isoform 2 [Homo sapiens]	30.4	30.4	26%	0.002	UGM
NP_671489.1	glutathione S-transferase Mu 4 isoform 2 [Homo sapiens]	28.1	28.1	50%	0.012	UGM
NP_000841.1	glutathione S-transferase Mu 4 isoform 1 [Homo sapiens]	28.1	28.1	50%	0.014	UGM
NP_000245.2	methionine synthase [Homo sapiens]	27.3	27.3	23%	0.031	UEGM
NP_001182566.1	hypothetical protein LOC100500938 [Homo sapiens]	25.4	25.4	18%	0.036	UG
XP_003403539.1	PREDICTED: glutathione S-transferase theta-4-like [Homo	26.6	26.6	48%	0.038	G
NP_061845.2	ganglioside-induced differentiation-associated protein 1 iso	26.6	52.7	55%	0.053	UGM
NP_057460.3	ankyrin repeat and FYVE domain-containing protein 1 isofo	26.6	26.6	13%	0.055	UGM
NP_001035808.1	ganglioside-induced differentiation-associated protein 1 iso	26.2	26.2	19%	0.067	UGM
NP_150648.2	vacuolar protein sorting-associated protein 13A isoform A	26.2	26.2	28%	0.068	UGM
NP_001018047.1	vacuolar protein sorting-associated protein 13A isoform C	26.2	26.2	28%	0.071	UGM
NP_001280.3	chloride intracellular channel protein 2 [Homo sapiens]	25.8	25.8	19%	0.071	UGM

48

BLAST results 4 - gstt1_drome

Homolog? [>ref|NP_036270.1| UGM](#) protein AATF [Homo sapiens]
 Length=560

[GENE ID: 26574 AATF](#) | apoptosis antagonizing transcription factor
 [Homo sapiens] (Over 10 PubMed links)

Score = 35.0 bits (79), Expect = 8e-05, Method: Compositional matrix adjust.
 Identities = 24/79 (30%), Positives = 34/79 (43%), Gaps = 7/79 (9%)

```

Query 123 ADPEAFKKIEAAFEFLNTFLEGGDYAAGDSLTVADIALVATVSTFEVAKFEISKYANVNR 182
          + A ++ F EG+D GD L V I +A+ S + K K +
Sbjct 22 ADPEADPEEATAARVIDRFDEGED-GEGLVVGSIKRLASASLLDTRKYCGKTSRKA 80

Query 183 WYENAKKVTGWENWAGC 201
          W E+ WE+ G
Sbjct 81 WNEDE-----WEQTLPGS 93
    
```

Homolog? [>ref|NP_000837.3| UGM](#) glutathione S-transferase A2 [Homo sapiens]
 Length=222

[GENE ID: 2939 GSTA2](#) | glutathione S-transferase alpha 2 [Homo sapiens]
 (Over 10 PubMed links)

Score = 34.7 bits (78), Expect = 8e-05, Method: Compositional matrix adjust.
 Identities = 42/181 (23%), Positives = 71/181 (39%), Gaps = 58/181 (32%)

```

Query 51 HTIPTLVNDGFALWESRAIQVYLVEKYKTDLSLYPKCPKRAVINQRLL--YFDMGTLY-- 106
          +P + +G L ++RAI Y+ KY +LY K K++A+I+ + D+G +
Sbjct 53 QQVPMVEIDGMKLVQTRAILNYIASKY----NLYGKDIKEKALIDMYIEGIADLGEMILL 108

Query 107 -----QSFANYYPQVFAKAPADPEAFKKIEAAFEFLNTFLEGGDYAA 149
          + N Y+P AF+K+ + GQDY
Sbjct 109 LPFSQPEEQDAKLALIQEKTKNRYFP-----AFEKVLKS-----HGQDYLV 149

Query 150 GDSLTVADIALV-----ATVSTF---EVAKFEISKYANVNRWYENAKKVTGPWE 195
          G+ L+ ADI LV + +S+F + K IS V ++ + P +
Sbjct 150 GNKLSRADIHVVELLYVEELDSSLISFLLKALKTRISNLPVTKKFLQGPSRKPMPD 209

Query 196 E 196
          E
Sbjct 210 E 210
    
```

49

BLAST results – validating statistics

Re-search vs SwissProt at NCBI site
 Query: AATF – initial $E() < 10^{-4}$

Sequences producing significant alignments:

Accession	Description	Max score	Total score	Query coverage	E value	Links
Q9N961.1	RecName: Full=Protein AATF; AltName: Full=Apoptosis-an	1137	1137	100%	0.0	G
Q9JXC4.1	RecName: Full=Protein AATF; AltName: Full=Apoptosis-an	794	794	99%	0.0	G
Q9QYW0.1	RecName: Full=Protein AATF; AltName: Full=Apoptosis-an	752	752	99%	0.0	G
Q5ZIM6.1	RecName: Full=Protein AATF; AltName: Full=Apoptosis-an	643	643	97%	0.0	G
Q9VH95.1	RecName: Full=Protein AATF-like	140	140	58%	1e-34	M
Q55E65.2	RecName: Full=Putative uncharacterized protein DDB_G02	129	129	59%	5e-31	
Q6BXX1.2	RecName: Full=Protein BFR2	111	111	60%	6e-25	
Q7S6P8.1	RecName: Full=Protein bfr-2	112	112	61%	9e-25	
Q6C9G2.1	RecName: Full=Protein BFR2	109	109	59%	2e-24	
Q9US05.1	RecName: Full=Protein bfr2	100	100	59%	1e-21	G
Q6CTS8.1	RecName: Full=Protein BFR2	94.0	94.0	67%	4e-19	
Q5ACL9.2	RecName: Full=Protein BFR2	92.8	92.8	58%	1e-18	
Q6FSD4.1	RecName: Full=Protein BFR2	91.7	91.7	58%	2e-18	
Q4WMI1.1	RecName: Full=Protein bfr2	88.6	88.6	70%	2e-17	G
Q06631.1	RecName: Full=Protein BFR2; AltName: Full=Brefeldin A r	87.4	87.4	60%	5e-17	G
Q3AW04.1	RecName: Full=Protein bfr2	86.7	86.7	64%	1e-16	
Q4P9V5.1	RecName: Full=Protein BFR2	85.5	85.5	61%	3e-16	
Q4I3Z7.1	RecName: Full=Protein BFR2	82.8	82.8	60%	2e-15	
Q7SEZ2.1	RecName: Full=Protein BFR2	81.6	81.6	60%	3e-15	G
POCL91.1	RecName: Full=Protein BFR2 >sp POCL90.1 BFR2_CRYNJ R	43.9	82.4	21%	0.003	

50

BLAST results – validating statistics

Re-search vs SwissProt at NCBI site

Query: GSTA1 – initial $E() < 10^{-4}$

Sequences producing significant alignments:

Accession	Description	Max score	Total score	Query coverage	E value	Links
P09210.4	RecName: Full=Glutathione S-transferase A2; AltName: Fu	459	449	100%	4e-162	G M
P08263.3	RecName: Full=Glutathione S-transferase A1; AltName: Fu	427	427	100%	2e-153	G M
Q78TV2.1	RecName: Full=Glutathione S-transferase A5; AltName: Fu	399	399	100%	1e-142	G
Q16772.3	RecName: Full=Glutathione S-transferase A3; AltName: Fu	399	399	100%	2e-142	G M
Q28035.3	RecName: Full=Glutathione S-transferase A1; AltName: Fu	374	374	100%	1e-132	G M
Q18929.4	RecName: Full=Glutathione S-transferase A2; AltName: Fu	373	373	100%	4e-132	G M
P51781.2	RecName: Full=Glutathione S-transferase alpha M14; AltN	367	367	99%	7e-130	G M
Q08892.2	RecName: Full=Glutathione S-transferase Yc; AltName: Fu	350	350	99%	4e-123	G M
Q64200.1	RecName: Full=Glutathione S-transferase A6; AltName: Fu	349	349	100%	8e-123	G M
Q08863.1	RecName: Full=Glutathione S-transferase alpha I; AltNam	340	340	99%	3e-119	G M
P30115.2	RecName: Full=Glutathione S-transferase A3; AltName: Fu	339	339	99%	1e-118	G
P04904.3	RecName: Full=Glutathione S-transferase alpha-3; AltNam	338	338	99%	3e-118	G M S
P13745.2	RecName: Full=Glutathione S-transferase A1; AltName: Fu	338	338	99%	4e-118	G M
P46418.2	RecName: Full=Glutathione S-transferase alpha-5; AltNam	331	331	99%	1e-115	G M
P10648.2	RecName: Full=Glutathione S-transferase A2; AltName: Fu	326	326	100%	2e-113	G M
P04903.2	RecName: Full=Glutathione S-transferase alpha-2; AltNam	323	323	100%	2e-112	G M
P09592.3	RecName: Full=Glutathione S-transferase alpha-1; AltNam	323	323	100%	2e-112	G M
P81706.1	RecName: Full=Glutathione S-transferase A; Short=GST A	313	313	97%	1e-108	G
Q08393.2	RecName: Full=Glutathione S-transferase; AltName: Full=	305	305	100%	4e-105	G
Q08392.1	RecName: Full=Glutathione S-transferase; AltName: Full=	295	295	99%	4e-101	G M
P26697.2	RecName: Full=Glutathione S-transferase 3; AltName: Full	287	287	99%	4e-98	G M
P24472.3	RecName: Full=Glutathione S-transferase A4; AltName: Fu	283	283	100%	1e-96	G M
P14942.2	RecName: Full=Glutathione S-transferase alpha-4; AltNam	278	278	100%	1e-94	G M
P80894.1	RecName: Full=Glutathione S-transferase; AltName: Full=	267	267	100%	3e-90	G M
Q5E900.1	RecName: Full=Glutathione S-transferase A4; AltName: Fu	264	264	99%	7e-89	G M
Q15217.1	RecName: Full=Glutathione S-transferase A4; AltName: Fu	261	261	99%	4e-88	G
P80931.2	RecName: Full=Glutathione S-transferase P; AltName: Full	96.3	96.3	86%	2e-23	G
Q29408.3	RecName: Full=Glutathione S-transferase P 10; AltName:	95.9	95.9	87%	3e-23	G
P12354.2	RecName: Full=Glutathione S-transferase P; AltName: Full	95.1	95.1	85%	4e-23	G
P81842.1	RecName: Full=Glutathione S-transferase P 1; AltName: F	94.0	94.0	90%	1e-22	G
P46426.1	RecName: Full=Glutathione S-transferase; AltName: Full=	93.2	93.2	91%	3e-22	G
Q60550.3	RecName: Full=Glutathione S-transferase P; AltName: Full	92.8	92.8	85%	4e-22	G
P46424.2	RecName: Full=Glutathione S-transferase P; AltName: Full	92.4	92.4	85%	6e-22	G
P46427.1	RecName: Full=Glutathione S-transferase 2; AltName: Full	89.7	89.7	91%	6e-21	G
P28801.2	RecName: Full=Glutathione S-transferase P; AltName: Full	89.7	89.7	85%	6e-21	G M
Q18598.3	RecName: Full=Glutathione S-transferase; AltName: Full=	89.0	89.0	94%	1e-20	G

51

Unrelated ≠ Random (low complexity)

Search with complete grou_drome:

The best scores are:

				opt	bits	E(14548)
RGHUB1	GTP-binding regulatory protein beta-1	chai	(341)	237	46.6	3.5e-05
RGBOB1	GTP-binding regulatory protein beta-1	chai	(341)	237	46.6	3.5e-05
RGHUB3	GTP-binding regulatory protein beta-3	chai	(341)	233	46.0	5.2e-05
RGMSB4	GTP-binding regulatory protein beta-4	chai	(341)	232	45.8	5.7e-05
PIHUPF	salivary proline-rich glycoprotein precurs		(252)	224	44.5	*0.00010*
RGFFB	GTP-binding regulatory protein beta chain		(347)	223	44.5	0.00014
PIRT3	acidic proline-rich protein precursor - rat		(207)	199	40.8	*0.0011*
PIHUB6	salivary proline-rich protein precursor PR		(393)	203	41.6	*0.0012*
CGBO2S	collagen alpha 2(I) chain - bovine (fragme		(403)	195	40.5	*0.0027*
WMBEW6	capsid protein - human herpesvirus 1 (stra		(636)	192	40.2	*0.0051*

Search with seg-ed grou_drome: (low complexity regions removed)

The best scores are:

				opt	bits	E(14548)
RGHUB3	GTP-binding regulatory protein beta-3	chai	(341)	233	56.5	3.6e-08
RGMSB4	GTP-binding regulatory protein beta-4	chai	(341)	232	56.3	4.1e-08
RGHUB2	GTP-binding regulatory protein beta-2	chai	(341)	228	55.5	7.2e-08
RGBOB1	GTP-binding regulatory protein beta-1	chai	(341)	225	54.9	1.1e-07
RGFFB	GTP-binding regulatory protein beta chain		(347)	223	54.5	1.5e-07
BVBVMS	MSI1 protein - yeast (Saccharomyces cerevi		(423)	135	37.0	*0.033*
ERHUAH	coatomer complex alpha chain homolog - hum		(1225)	134	37.1	*0.088*
A28468	chromogranin A precursor - human		(458)	122	34.4	*0.21*
RGOOBE	GTP-binding regulatory protein beta chain		(342)	120	33.9	0.22

52

pseg removes low-complexity regions

>gi|17380405|sp|P16371|GROU_DROME Groucho protein (Enhancer of split M9/10)

```

1-8      MYSPVVRH
paagppppgpp 9-19
20-131   IKFTIADTLERIKEEFNFLQAQYHSIKLEC
          EKLSNEKTEMQRHYVVMYEMSYGLNVMHK
          QTEIAKRLNLTINQLLPFLQADHQQVQLQA
          VERAKQVTMQELNLIIGQQIHA
qqvpppppppmg 132-143
144-281  ALNPPGALGATMGLPHGPQGLLNKPPPEHHR
          PDIKPTGLEGPAAAEERLRNSVSPADREKY
          RTRSPLDIENDSKRRKDEKLEDEGEKSDQ
          DLVVDVANEMESHSPRPNGEHVSMVDRRE
          SLNGERLEKPSSSGIKQE
rppsrsgsssrstps 282-297
298-310  LKTKDMEKPGTPG
akartptpnaaapagvnpk 311-330
qmmpqgppppagypgapyqrpa 331-351
352-719  DPYQRPPSPAYGRPPMPYDPHAHVRTNG
          IPHPSALTGGKPAYSFHMNGESLQPVVPPP
          PDALVGVGIPRHARQINTLSHGEVCAVTI
          SNPTKYVYTGKGCVKVWDISQPGNKNPVS
          QLDCLQRDNYIRSVKLLPDGRTLIVGGEAS
          NLSIWDLASPTPRIKAELTSAAPACYALAI
          SPDSKVCFCSCSDGNI AVWDLHNEILVRQF
          QGHTDGASCIDISPDGSRLLWTGGLDNTVRS
          WDLREGRLQQHDFSSQIFSLGYCPTGDWL
          AVGMENSHVEVLHASKPKDYQLHLHESCVL
          SLRFAACGKWFVSTGKDNLLNARWTPYGAS
          IFQSKETS SVLSCDISTDDKYIVTGS GDKK
          ATVYEVII
    
```

53

BLAST remove low complexity - gstt1_drome

Scoring Parameters

Matrix: BLOSUM62

Gap Costs: Existence: 11 Extension: 1

Compositional adjustments: Conditional compositional score matrix adjustment

Filters and Masking

Filter: Low complexity regions

Mask: Mask for lookup table only Mask lower case letters

Sequences producing significant alignments:

Accession	Description	Max score	Total score	Query coverage	E value	Links
NP_000844.2	glutathione S-transferase theta-1 [Homo sapiens]	76.3	76.3	72%	8e-17	UGM
NP_000845.1	glutathione S-transferase theta-2 [Homo sapiens] >ref[NP_000845.1]	71.6	71.6	71%	5e-15	UGM
NP_665877.1	maleylacetoacetate isomerase isoform 1 [Homo sapiens]	56.6	56.6	75%	7e-10	UGM
NP_001504.2	maleylacetoacetate isomerase isoform 3 [Homo sapiens]	47.4	47.4	61%	8e-07	UGM
NP_001903.1	glutathione S-transferase A4 [Homo sapiens]	49.0	40.0	78%	5e-04	UGM
NP_004271.1	eukaryotic translation elongation factor 1 epsilon-1 isoform	38.5	38.5	73%	0.001	UGM
NP_006294.2	aminoacyl tRNA synthase complex-interacting multifunctional	37.7	37.7	30%	0.004	UGM
NP_665683.1	glutathione S-transferase A1 [Homo sapiens]	37.4	37.4	69%	0.004	UGM
NP_036270.1	protein AATF [Homo sapiens]	35.0	35.0	37%	0.030	UGM
NP_000837.3	glutathione S-transferase A2 [Homo sapiens]	34.7	34.7	69%	0.031	UGM
NP_000838.3	glutathione S-transferase A3 [Homo sapiens]	33.9	33.9	53%	0.056	UGM
NP_714543.1	glutathione S-transferase A5 [Homo sapiens]	32.7	32.7	92%	0.13	UGM
NP_671488.1	blood vessel epicardial substance [Homo sapiens] >ref[NP_671488.1]	33.1	33.1	21%	0.13	UGM
NP_001129122.1	eukaryotic translation elongation factor 1 epsilon-1 isoform	31.6	31.6	60%	0.24	UGM
NP_665878.2	maleylacetoacetate isomerase isoform 2 [Homo sapiens]	30.4	30.4	26%	0.73	UGM
NP_671489.1	glutathione S-transferase Mu 4 isoform 2 [Homo sapiens]	28.1	28.1	50%	4.6	UGM
NP_000841.1	glutathione S-transferase Mu 4 isoform 1 [Homo sapiens]	28.1	28.1	50%	5.2	UGM

Validating homologs/statistics

- In general, BLASTP statistical estimates are accurate
- The most common errors occur because of low-complexity regions, or biased amino-acid composition
- To confirm statistical accuracy, find the highest scoring non homolog
 - No need to test every hit, test hits that are surprising
 - Confirm homology/non-homology by searching against a different comprehensive database, e.g. SwissProt, or refseq.
 - Non-homologs will find many significant members of other families, but not the family you are testing for
- Statistical estimates can be confirmed with shuffles (see ISMB2000 tutorial, fasta.bioch.virginia.edu/fasta_www2 shuffle link)

55

Scoring matrices

- Scoring matrices can set the evolutionary look-back time for a search
 - Lower PAM (PAM10/MDM10 ... PAM60) for closer (10% ... 50% identity)
 - Higher BLOSUM for higher conservation (BLOSUM50 distant, BLOSUM80 conserved)
- Shallow scoring matrices for short domains/short queries (metagenomics)
 - Matrices have “bits/position” (score/position), 40 aa at 0.7 bits/position (BLOSUM62) means 28 bit max score (50 bits significant)
- Deep scoring matrices allow alignments to continue, possibly outside the homologous region

56

Finding Domains – Local alignments: calmodulin

```

46.1% identity in 76 aa overlap (1-76:77-149); score: 222 E(10000): 2.7e-10
      10      20      30      40      50      60
mchu MADQLTEEQIAEFKEAFSLFDKDGDTITTKELGTVMRSLGQNPTEAELQDMINEVDADG
: : .::: .::: .::: .::: .::: .::: .::: .::: .::: .::: .::: .::: .:::
mchu MKDTSSEEEI---REAFRVFDKNGYISAAELRHVMTNLGKLTDEEVDEMIREADIDG
      80      90      100     110     120     130

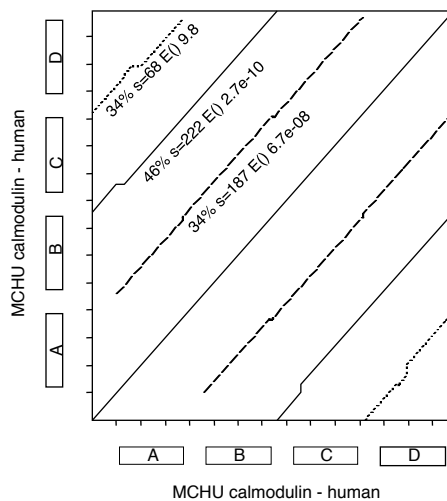
      70
mchu NGTIDFPEFLTMMARK
.: .: .: .: .: .:
mchu DGQVNYEEFVQMMTAK
      140

34.3% identity in 105 aa overlap (11-111:47-147); score: 187 E(10000): 6.7e-08
      20      30      40      50      60
mchu AEFKEAFSLFDKDGDTITTKELGTVM-RSLGQNPTEAELQDMINEVDADGNGTIDFPEF
: : .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .:
mchu AELQDMINEVDADGNGTIDFPEFLTMMARKMKDTSSEEEIREAFRVFDKNGYISAAEL
      50      60      70      80      90      100
      70      80      90      100     110
mchu ---LTMMARKMKDTSSEEEIREAFRVFDKNGYISAAELRHVMT
.: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .:
mchu RHVMTNLGKLTDEEVDEMIREA---DIDGGQVNYEEFVQMMT
      110     120     130     140

34.2% identity in 38 aa overlap (1-37:113-146); score: 68 E(10000): 9.8
      10      20      30
mchu MADQLTEEQIAEF-KEAFSLFDKDGDTITTKELGTVM
.: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .: .:
mchu LGEKLTDEEVDEMIREA---DIDGGQVNYEEFVQMM
      120     130     140
    
```

57

Repeated domains with local alignments



58

More about scoring matrices ...

PAM series:

- Evolutionary model - extrapolated from PAM1
- PAM20: 20% change (mammals)
- PAM250: 250% change (<20% identity)
- Gap penalties should vary
- shallow matrices (PAM10-40) for short sequences and short distances

BLOSUM series

- Empirically determined, no extrapolation (no model)
- BLOSUM45-50 - distant (1/3 bits)
- BLOSUM80 -very highly conserved (not small change), high info/position
- BLOSUM62 - 1/2 bits

59

Where do scoring matrices come from?

Pam40

```

A R N D E I L
A 8
R -9 12
N -4 -7 11
D -4 -13 3 11
E -3 -11 -2 4 11
I -6 -7 -7 -10 -7 12
L -8 -11 -9 -16 -12 -1 10

```

Pam250

```

A R N D E I L
A 2
R -2 6
N 0 0 2
D 0 -1 2 4
E 0 -1 1 3 4
I -1 -2 -2 -2 -2 5
L -2 -3 -3 -4 -3 2 6

```

q_{ij} : replacement frequency at PAM40, 250

$$q_{R:N(40)} = 0.000435$$

$$p_R = 0.051$$

$$q_{R:N(250)} = 0.002193$$

$$p_N = 0.043$$

$$I_2 S_{ij} = \lg_2 (q_{ij}/p_i p_j) \quad I_e S_{ij} = \ln(q_{ij}/p_i p_j) \quad p_R p_N = 0.002193$$

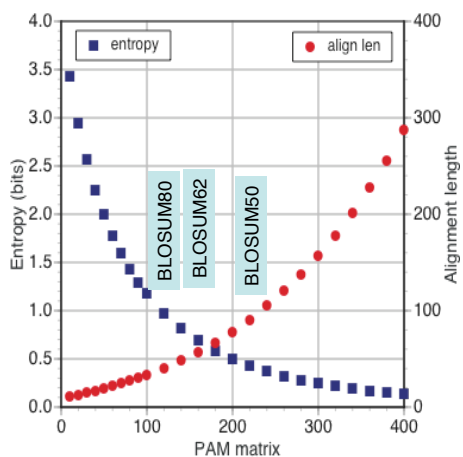
$$I_2 S_{R:N(40)} = \lg_2 (0.000435/0.00219) = -2.333$$

$$I_2 = 1/3; S_{R:N(40)} = -2.333/I_2 = -7$$

$$I S_{R:N(250)} = \lg_2 (0.002193/0.002193) = 0$$

60

PAM matrices and alignment length



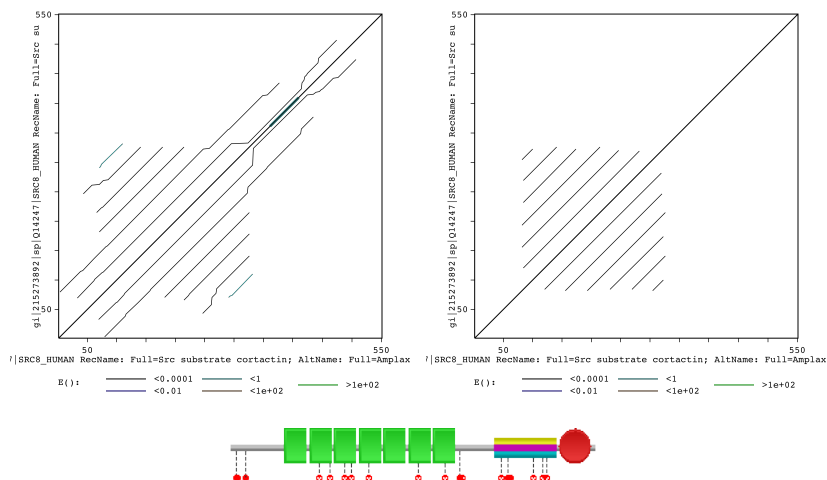
Short domains require “shallow” scoring matrices

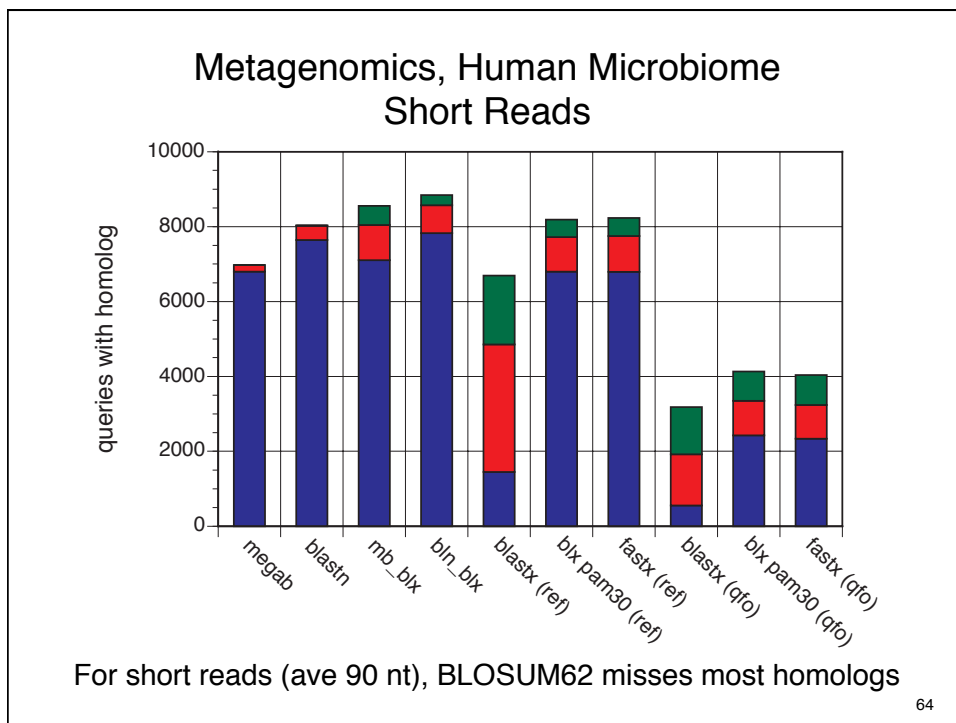
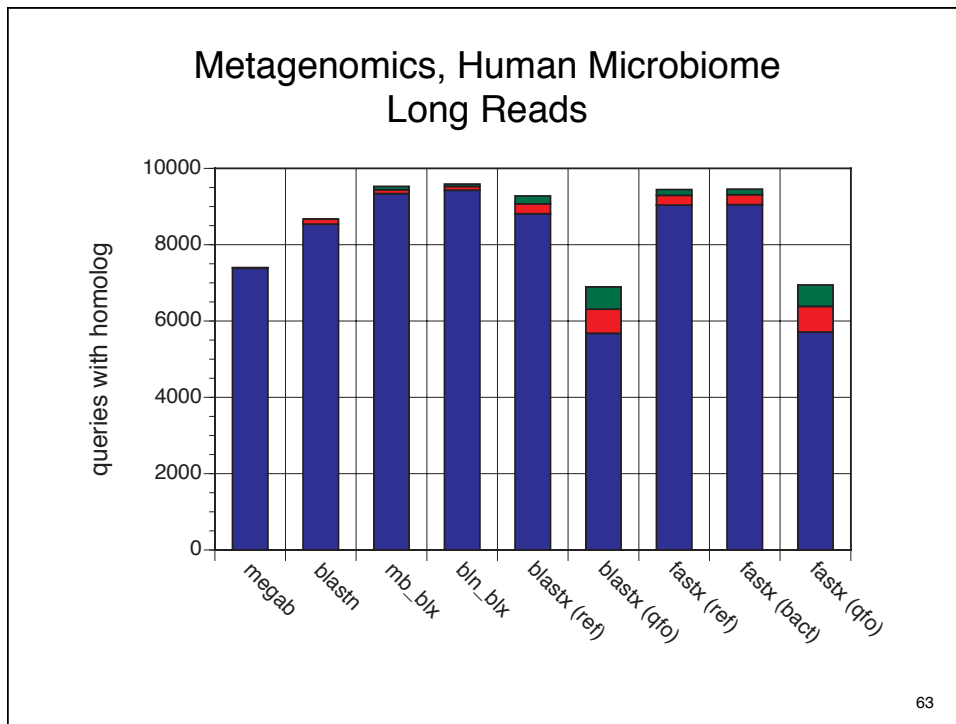
61

Scoring matrices affect alignment lengths

BLOSUM62 -11/-1

MD20 -26/-4





Scoring Matrices - Summary

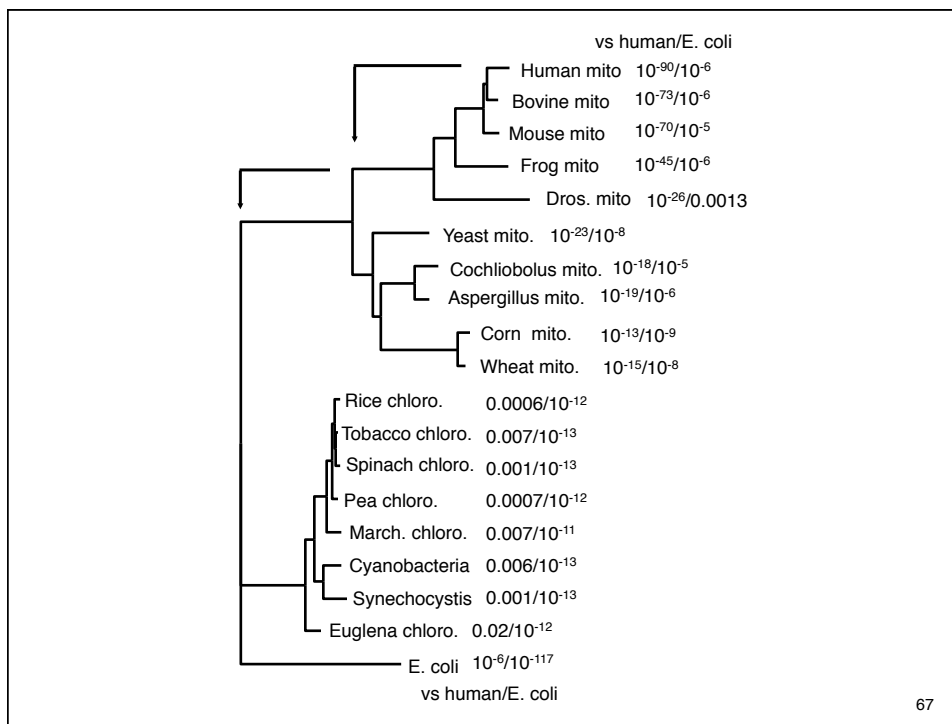
- PAM and BLOSUM matrices greatly improve the sensitivity of protein sequence comparison – low identity with significant similarity
- PAM matrices have an evolutionary model - lower number, less divergence – lower=closer; higher=more distant
- BLOSUM matrices are sampled from conserved regions at different average identity – higher=more conservation
- Short alignments require shallow matrices
- Shallow matrices set maximum look-back time

65

Similarity Searching II

1. What question to ask?
2. What program to use?
3. What database to search?
4. How to avoid mistakes (what to look out for)
5. When to do something different
6. PSI-BLAST – the most sensitive method

66



ATP synthase - matrices, gaps, algorithms

```

Matrix:
Gap open/extend      BLOSUM50      BLOSUM62      BLASTP
                    -10/-2        -11/-1        -11/-1
The best scores are:  bits E(13351) bits E(13351) bits E( )
ATP6_HUMAN ATP synthase a chai 297.7 1.7e-81 373.6 2.4e-104 296 3e-81
ATP6_BOVIN ATP synthase a chai 252.4 7.2e-68 310.7 2.0e-85 253 2e-68
ATP6_MOUSE ATP synthase a chai 246.4 4.5e-66 302.9 4.4e-83 245 5e-66
ATP6_XENLA ATP synthase a chai 111.9 1.4e-25 125.9 8.7e-30 142 9e-35
ATP6_YEAST ATP synthase a ch 78.7 1.6e-15 90.1 5.7e-19 93 5e-20
ATP6_EMENI ATP synthase a chai 66.3 8.4e-12 76.6 6.8e-15 75 2e-14
ATP6_DROYA ATP synthase a chai 65.6 1.2e-11 75.4 1.4e-14 101 2e-22
ATP6_COCHE ATP synthase a cha 53.6 5.5e-08 60.6 4.6e-10 75 1e-14
ATP6_ECOLI ATP synthase a ch 45.1 2.2e-05 49.1 1.4e-06 42 1e-04
ATP6_TRITI ATP synthase a ch 45.0 3.3e-05 50.7 6.5e-07 83 5e-17
ATP6_TOBAC ATP synthase a chai 40.4 0.00084 47.0 8.6e-06 80 3e-16
ATP6_MAIZE ATP synthase a chai 39.6 0.001 44.9 2.6e-05
ATPI_PEA Chloroplast ATP syn 35.8 0.013 38.0 0.0028
ATPI_SPIOL Chloroplast ATP syn 35.5 0.015 38.0 0.0028
ATPI_ATRBE Chloroplast ATP s 34.0 0.044 36.3 0.0086
ATPI_MARPO Chloroplast ATP syn 33.2 0.075 34.3 0.036
*HBA_ODOVI Hemoglobin subunit a 31.9 0.11*
*AROP_ECOLI Aromatic amino ac 32.1 0.31 31.4 0.5 *
ATPI_EUGGR Chloroplast ATP syn 31.1 0.32 32.2 0.15
ATP6_SYN6 ATP synthase a chai 31.1 0.34 31.8 0.21
TLCA_RICPR ADP,ATP carrier pro 31.5 0.49 29.7 1.7
ATP6_SYNY3 ATP synthase a chai 30.6 0.51 31.8 0.22 28 1.9
ATPI_ORYSA Chloroplast ATP 30.1 0.65 32.2 0.15
*GLUC_MYOSC Glucagon precursor 28.7 0.65 34.4 0.013*
*VP6_BPPH6 Protein P6 29.1 0.85 28.6 1.3*
*GLUC_LEFSP Glucagon precursor 27.7 1. 32.7 0.033*
*ADH1_MOUSE Alcohol dehydrogena 29.8 1.2 34.4 0.013*

```

68

Metazoan ATP Synthases

CLUSTAL W (1.81) multiple sequence alignment

```

ATP6_BOVIN  MNENLFTSFITPVLGLPLVTLIVLFPSSLF--PTSNRLVSNRFVTLQQWMLQVSKQMMSIHNSKGQTWT-LML
ATP6_MOUSE  MNENLFASFITPTMMGFPIVVAIIMPSSLF--PSSKRLINNRLHSFQHWLVKLIKQMMLIHTPKGRTWT-LMI
ATP6_HUMAN  MNENLFASFIAPTILGLPAAVLIIILFPLLII--PTSKYLINNRLITQQWLKIKLSKQMMTMMHTKGRWTS-LML
ATP6_XENLA  MNLSFFDQFMSPVILGIPLIAMIADPFTLISWPIQSNGFNNRLITLQSWFLHNFITFYQLTSP-GHKWA-LLL
ATP6_DROYA  MMTNLFVDFPSAIFNLSNLWLSLFLGLLMI--PSIYWLMPSRYNIFWNSILLTLHKEFKTLGLGSGHNGSTFIF
*  .* * ...:.. :  : *  .*  :  :  : *  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :

```

```

ATP6_BOVIN  MSLILFIGSTNLLGLLPHSFTPTTQLSMNLGMAIPLWAGAVITGFRNKTKASLAHFLPQGTPTPLIPMLVIEIETI
ATP6_MOUSE  VSLIMFIGSTNLLGLLPHFTPTTQLSMNLSMAIPLWAGAVITGFRHKLKSSLAHFLPQGTPTPLIPMLVIEIETI
ATP6_HUMAN  VSLIIFIAITNLLGLLPHSFTPTTQLSMNLAMAIPWAGTVIMGFRSKIKNALAHFLPQGTPTPLIPMLVIEIETI
ATP6_XENLA  TSLMLLLSLNLLGLLPYTFPTTQLSLNMLAVPLWLATVIMASKP-TNYALGHLLPEGTPTPLIPVLIIEIETI
ATP6_DROYA  ISLFLILFNNFMGLPPYIFTSTSHLTLTLALPLWLCFMYLWGWINHQTQHMFAHLPQGTPTPLIPMLVIEIETI
**  :  :  *  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :

```

```

ATP6_BOVIN  SLFIQPMALAVRLTANITAGHLLIHLGGATLALMSISTTTALITFTTILLILEFAVAMIQAYVFTLLVSLYLHDNT
ATP6_MOUSE  SLFIQPMALAVRLTANITAGHLLMHLIGGATLVMNISPPTATITPIILLLLILEFAVALIQAYVFTLLVSLYLHDNT
ATP6_HUMAN  SLLIQPMALAVRLTANITAGHLLMHLIGSATLAMSTINLPSTLIIPTIILLILEFAVALIQAYVFTLLVSLYLHDNT
ATP6_XENLA  SLFIRPLALGVRLTANITAGHLLIQLIATAAFVLLSIMPVAILTSIVFLFLTLEIAVAMIQAYVFTLLVSLYLQENV
ATP6_DROYA  SNIIRPGTLAVRLTANMIAGHLLLTLLGNTGSPMSYLLVFTLLVAQIALLVL--ESAVTMIQSYVFAVLSTLYSSEV
*  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :  :

```

69

PSI-BLAST ATP6_HUMAN - 4 iterations

Results from round:

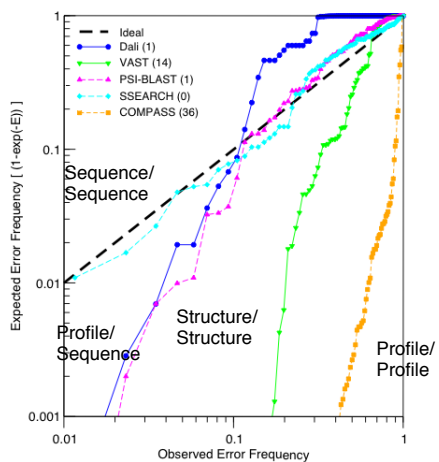
Sequences producing significant alignments:	(1)		(2)		(3)		(4)	
	Score (bits)	E Value	Score (bits)	E Value	Score (bits)	E Value	Score (bits)	E Value
ATP6_HUMAN ATP synthase a chain (ATPase protein 6)	296	3e-81	257	1e-69	241	2e-62	222	5e-59
ATP6_BOVIN ATP synthase a chain (ATPase protein 6)	253	2e-68	257	2e-69	239	8e-65	230	2e-61
ATP6_MOUSE ATP synthase a chain (ATPase protein 6)	245	5e-66	247	3e-66	234	4e-64	225	6e-60
ATP6_XENLA ATP synthase a chain (ATPase protein 6)	142	9e-35	227	1e-60	189	3e-49	177	2e-45
ATP6_DROYA ATP synthase a chain (ATPase protein 6)	101	2e-22	206	3e-54	209	5e-55	196	4e-51
(2)								
ATP6_YEAST ATP synthase a chain precursor (ATPase prot	93	5e-20	97	3e-21	199	4e-52	191	2e-49
ATP6_TRITTI ATP synthase a chain (ATPase protein 6)	83	5e-17	96	5e-21	218	1e-57	236	4e-63
(3)								
ATP6_TOBAC ATP synthase a chain (ATPase protein 6)	80	3e-16	90	4e-19	200	2e-52	230	3e-61
ATP6_MAIZE ATP synthase a chain (ATPase protein 6)	76	5e-15	88	1e-18	198	1e-51	219	5e-58
ATP6_COCHE ATP synthase a chain (ATPase protein 6)	75	1e-14	86	9e-18			197	2e-51
ATP6_EMENI ATP synthase a chain precursor (ATPase prot	75	2e-14	84	3e-17	123	5e-29	181	2e-46
(4)								
ATP6_ECOLI ATP synthase a chain (ATPase protein 6)	42	1e-04	40	5e-04	46	8e-06	49	1e-06
ATPI_SPIOL Chloroplast ATP synthase a chain precursor			32	0.12	36	0.006	39	0.001
ATP6_SYNY3 ATP synthase a chain (ATPase protein 6)	28	1.9	32	0.16	44	5e-05	45	1e-05
ATPI_MARPO Chloroplast ATP synthase a chain precursor			31	0.21	44	4e-05	44	3e-05
ATPI_PEA Chloroplast ATP synthase a chain precursor (A			31	0.32	37	0.005		
LAMA2_MOUSE Laminin subunit alpha-2 precursor (Laminin			31	0.34				
ATPI_ATRBE Chloroplast ATP synthase a chain precursor			31	0.39	41	2e-04		
ATP6_SYNP6 ATP synthase a chain (ATPase protein 6)			28	1.7	41	2e-04		
ATPI_EUGHR Chloroplast ATP synthase a chain precursor					39	0.001		
ATPI_ORYSA Chloroplast ATP synthase a chain precursor			28	1.9	36	0.008		
ATPI_ATRBE Chloroplast ATP synthase a chain precursor					36	0.009	38	0.002
ATP6_ASPAM ATP synthase a chain (ATPase protein 6)							36	0.008
POLG_KUNJM Genome polyprotein [Contains: Capsid protei...	27	5.0						
POL_HTLIC Gag-Pro-Pol polyprotein (Pr160Gag-Pro-Pol) [...	27	5.0						
POLG_DEN2J Genome polyprotein [Contains: Capsid protei...	27	5.2	26	7.0				

70

Position-Specific Scores ATP Synthase, 4 iterations

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	bits/pos
BL62 Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0.70
46 Q	-2	-1	-2	-2	-4	6	0	1	0	-4	-3	-1	-2	-1	-3	-1	-2	6	4	-3	0.74
%	0	0	0	0	0	54	0	12	0	0	0	0	0	0	0	0	0	13	20	0	
47 Q	-1	-1	3	3	-3	3	3	-2	3	-4	-4	-1	-3	-4	-2	2	-1	-4	-2	-3	0.51
%	0	0	13	20	0	16	19	0	8	0	0	0	0	0	0	24	0	0	0	0	
56 Q	-2	-1	-2	-2	-3	5	2	-4	-1	4	-1	-1	-1	-2	-3	-2	-2	-3	-2	0	0.51
%	0	0	0	0	0	46	13	0	0	41	0	0	0	0	0	0	0	0	0	0	
97 Q	-2	-1	0	-2	-4	4	0	-3	8	-4	-4	-1	-2	-3	-3	-1	-2	-3	0	-4	1.11
%	0	0	0	0	0	35	0	0	65	0	0	0	0	0	0	0	0	0	0	0	
131 Q	3	-1	-1	-1	-2	5	2	-2	-1	-3	-3	0	-2	-4	-2	1	-1	-3	-3	-2	0.52
%	44	0	0	0	0	36	11	0	0	0	0	0	0	0	0	9	0	0	0	0	
152 Q	-2	6	-1	-2	-4	4	0	-3	-1	-4	-3	1	-2	-4	-3	-1	-2	-4	-3	-3	1.00
%	0	77	0	0	0	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
210 Q	-2	0	-1	-1	-4	7	1	-3	0	-4	-3	1	-1	-4	-2	-1	-2	-3	-2	-3	1.13
%	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Accuracy of statistical estimates



- SSEARCH (Smith-Waterman) provides very accurate statistical estimates
- PSI-BLAST can provide estimates that off by 10–100-fold

Why does PSI-BLAST fail?

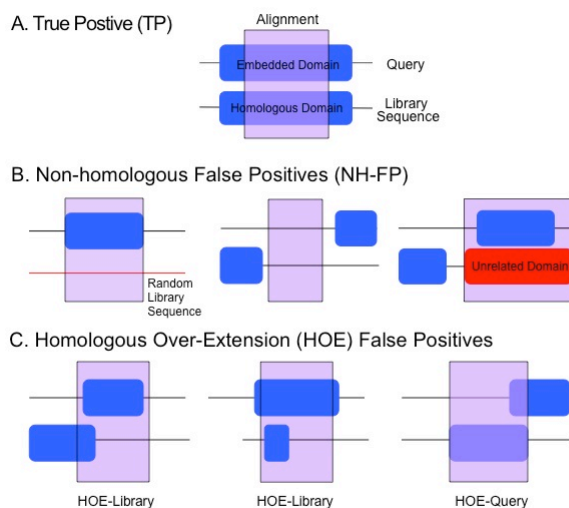


Figure 1

Sensitive searches with PSI-BLAST

- PSI-BLAST improves sensitivity by building a Position Specific Scoring Matrix (PSSM)
 - models ancestral sequence (consensus distribution)
 - similar to PFAM HMM (but less sophisticated weights, gaps)
- Sensitivity improves with additional iterations
 - model moves to base of tree
- Statistical estimates are difficult
 - once a sequence is in, it is “significant” - validation must be done before a sequence is included
- Very diverse families may not produce a well defined PSSM
 - similar problems with HMMs have led to “clans”

Sequence Similarity - Conclusions

- Homologous sequences share a common ancestor, but most sequences are non-homologous
- Always compare Protein Sequences
- Sequence Homology can be reliably inferred from statistically significant similarity (non-homology cannot from non-similarity)
- Homologous proteins share common structures, but not necessarily common functions
- Sequence statistical significance estimates are accurate (verify this yourself) $10^{-6} < E() < 10^{-3}$ is statistically significant
- Scoring matrices set evolutionary look back horizons - not every discovery is distant
- PSI-BLAST can be more sensitive, but with lower statistical accuracy

75

Discussion questions

1. What is the difference between similarity and homology? When does high identity not imply homology? What conclusions can be drawn from homology?
2. What is the difference between homology and common ancestry?
3. When the *M. janaschii* genome was first sequenced, Venter and his colleagues stated that almost 60% of the open reading frames (proteins or genes) were novel to this organism. (For eubacterial like *E. coli* or *H. influenzae*, a similar number would be 20 - 40%.) On what would they base such a statement? Is it likely to be correct?
4. Name two reasons why protein sequence comparison is more effective (longer evolutionary look-back time) than DNA sequences?
5. What is the range of an expectation value (E()-value)? If you compare a sequence to 50,000 random (unrelated) sequences, what should the expectation value for the highest of the 50,000 similarity scores be (on average)?
6. In a sequence similarity database search, you identify a statistically significant similarity ($E() < 0.005$), but the alignment is relatively short (50 aa). How might you determine whether the alignment reflects a genuine homology, or a random sequence match?
7. How can a sequence be homologous if you search a small database (e.g. human, 40,000 sequences), but not share significant similarity if you search a complete database (>4 million sequences)?
8. What scoring matrix should be used to identify protein orthologs that have diverged over the past 100 My (e.g. human/mouse)?
9. What scoring matrix should be used when comparing Illumina 90 nt reads against a protein database?

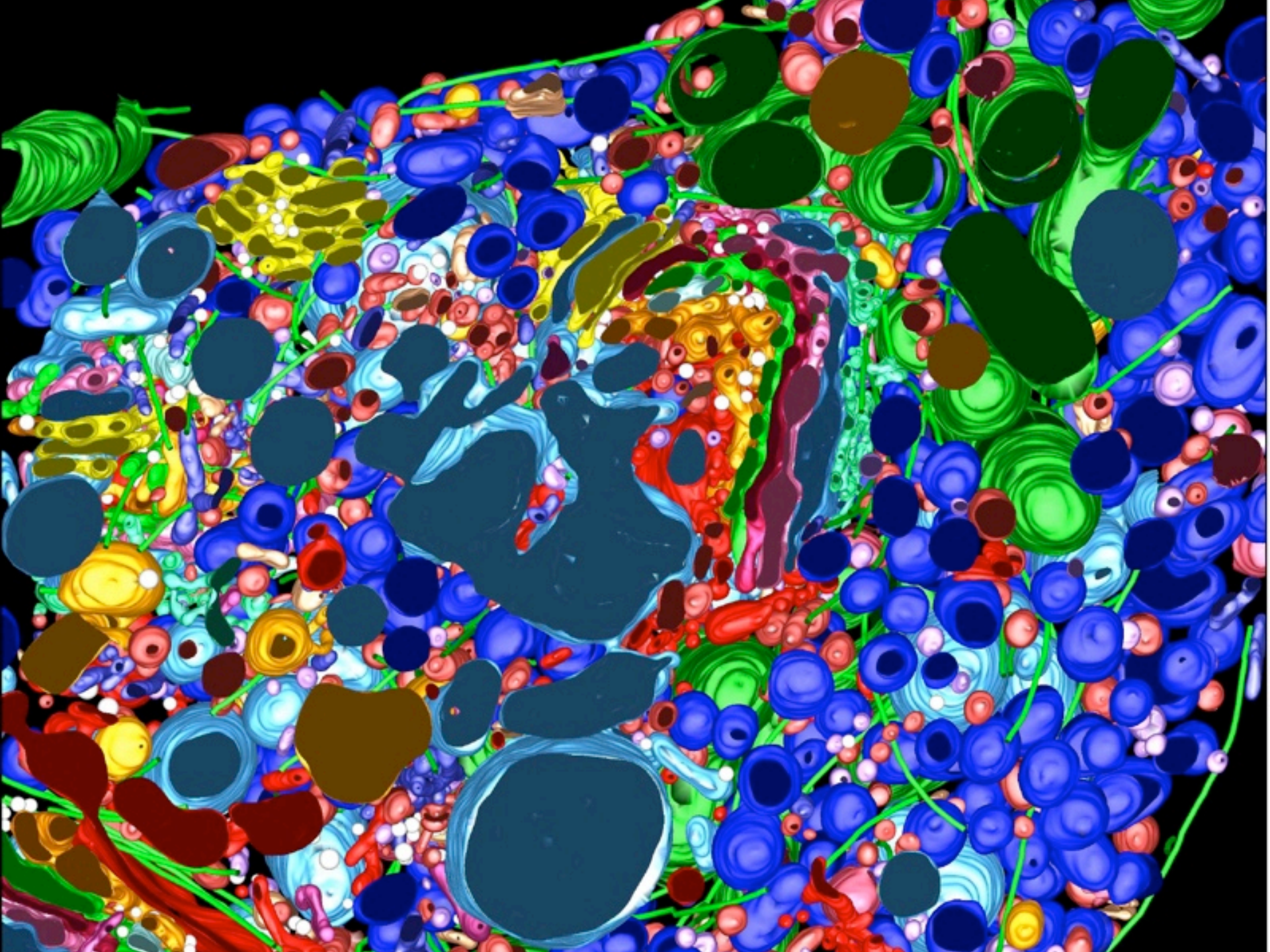
76

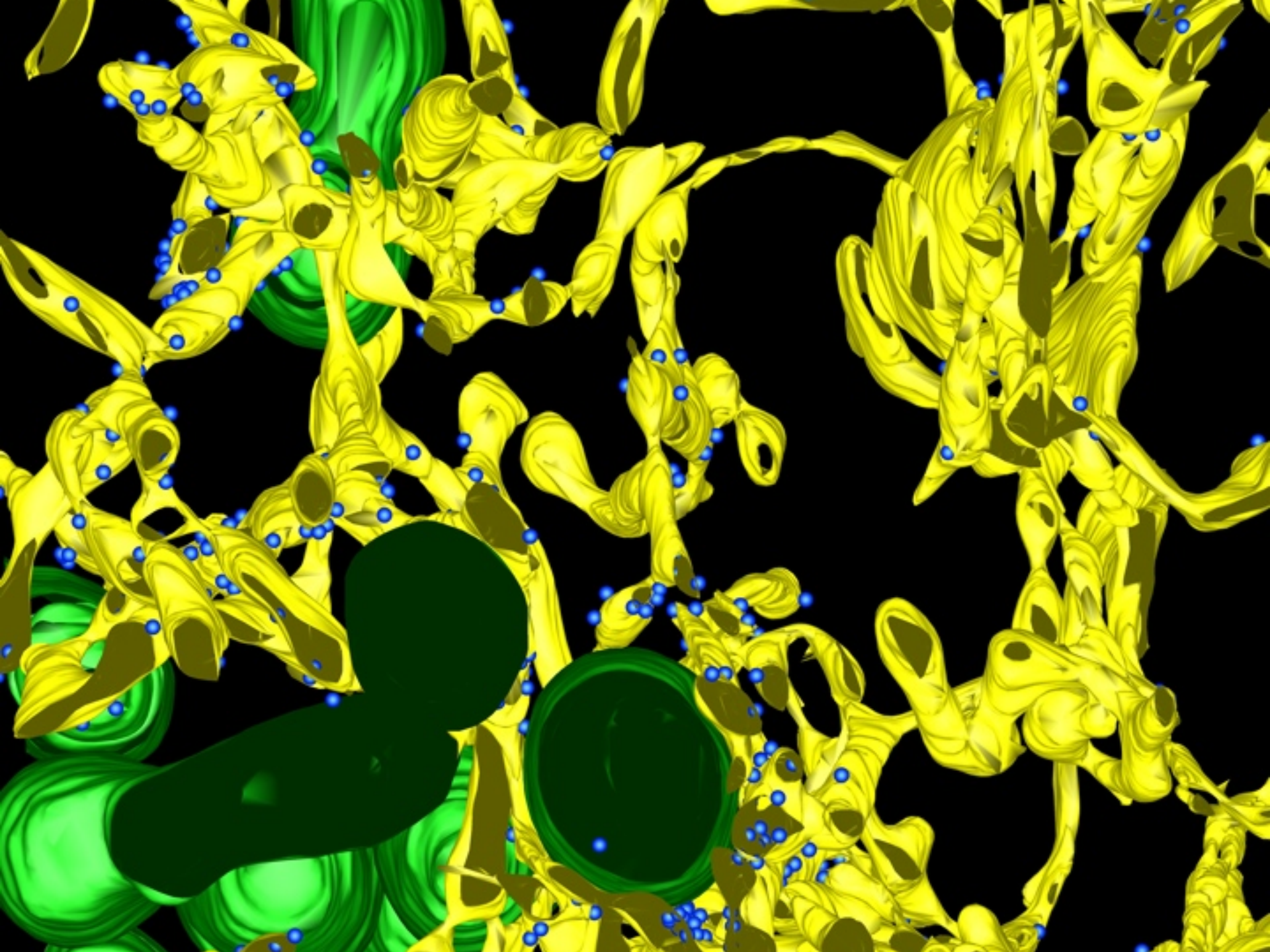
Gene expression

**Transcript Reconstruction and
Analysis**

CSHL October 2011

Win Hide whide@hsph.harvard.edu





Where to begin...

- Improved gene finding tools
 - Unlikely to be perfect
 - Will never identify the expression location, and function
- Comprehensive full-length mRNA sequencing
 - Technically difficult and time consuming
 - Less likely to detect rare/ highly restricted transcripts.
- Reconstruction of expressed genes using computational and experimental methods
 - Technically simpler
 - RNAseq/Fragment databases contain a large portion of the transcriptome of numerous organisms – prime resource
 - Variety of tissues, developmental states and libraries = good chance to detecting variant / rare/ restricted transcripts

SEP
14

RNA-Seq Data Cont miRBase Update

Filed Under [News](#)

The [miRBase](#) [microRNA](#) [Sequence](#) [Database](#) is a
repository for published microRNA sequen

Which sequencing platform/technology is best suited for RNA-Seq applications?

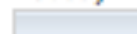
Pacific Biosystems (0%, 0 Votes)



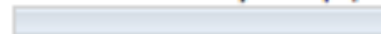
Helicos - HeliScope (0%, 0 Votes)



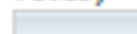
Applied Biosystems - SOLiD (17%, 1
Votes)



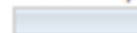
Illumina - HiSeq (50%, 3 Votes)



Illumina - Genome Analyzer (17%, 1
Votes)



Roche/454 - GS FLX (16%, 1 Votes)



Transcript Reconstruction Evolution

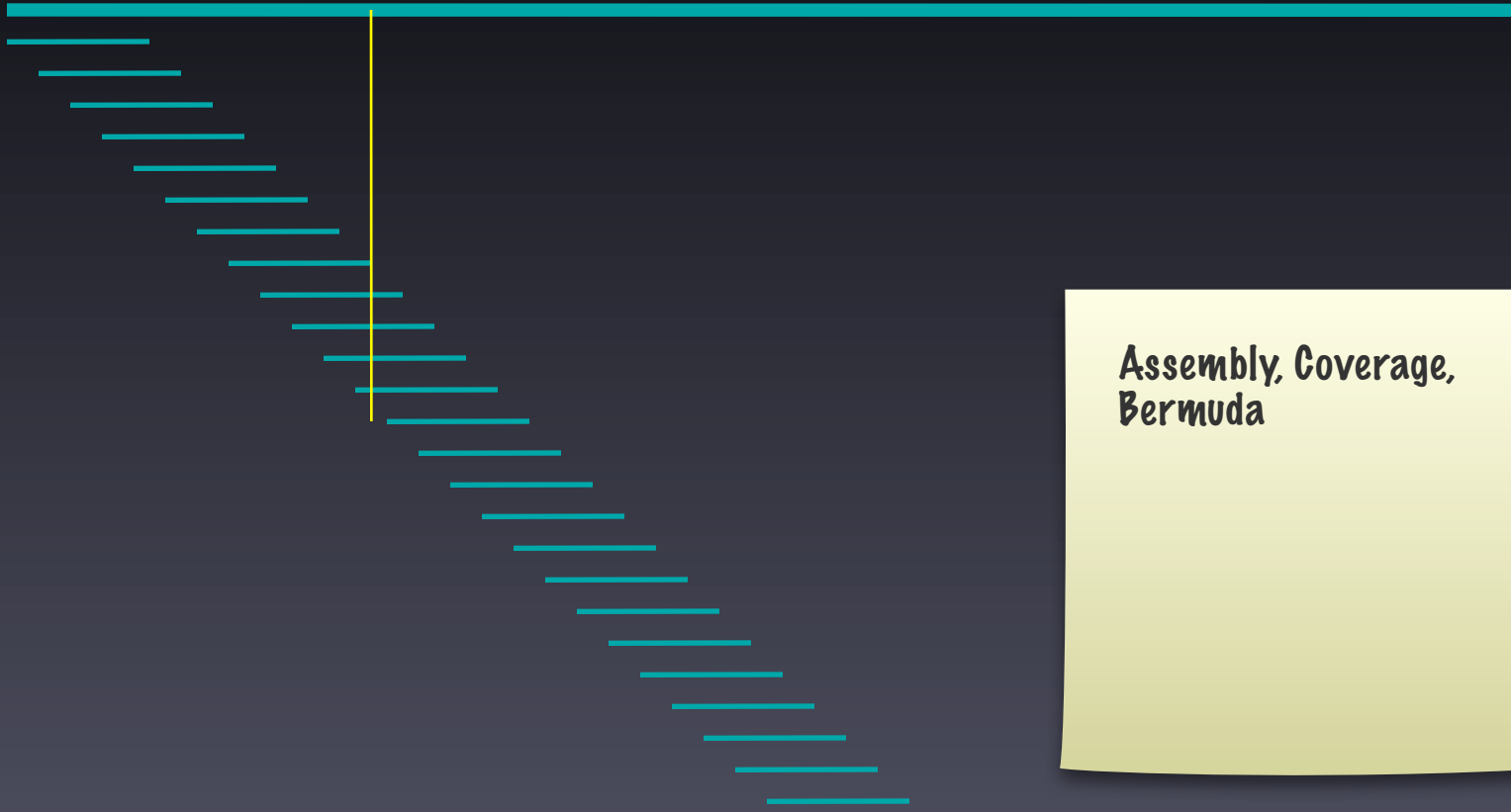
- **Gene-based detection of RNAs**
 - Northern blotting
- **High throughput expression profiling**
 - Microarrays.
 - Next-generation sequencing technologies
multidimensional examination of cellular transcriptomes
single-base resolution.

History

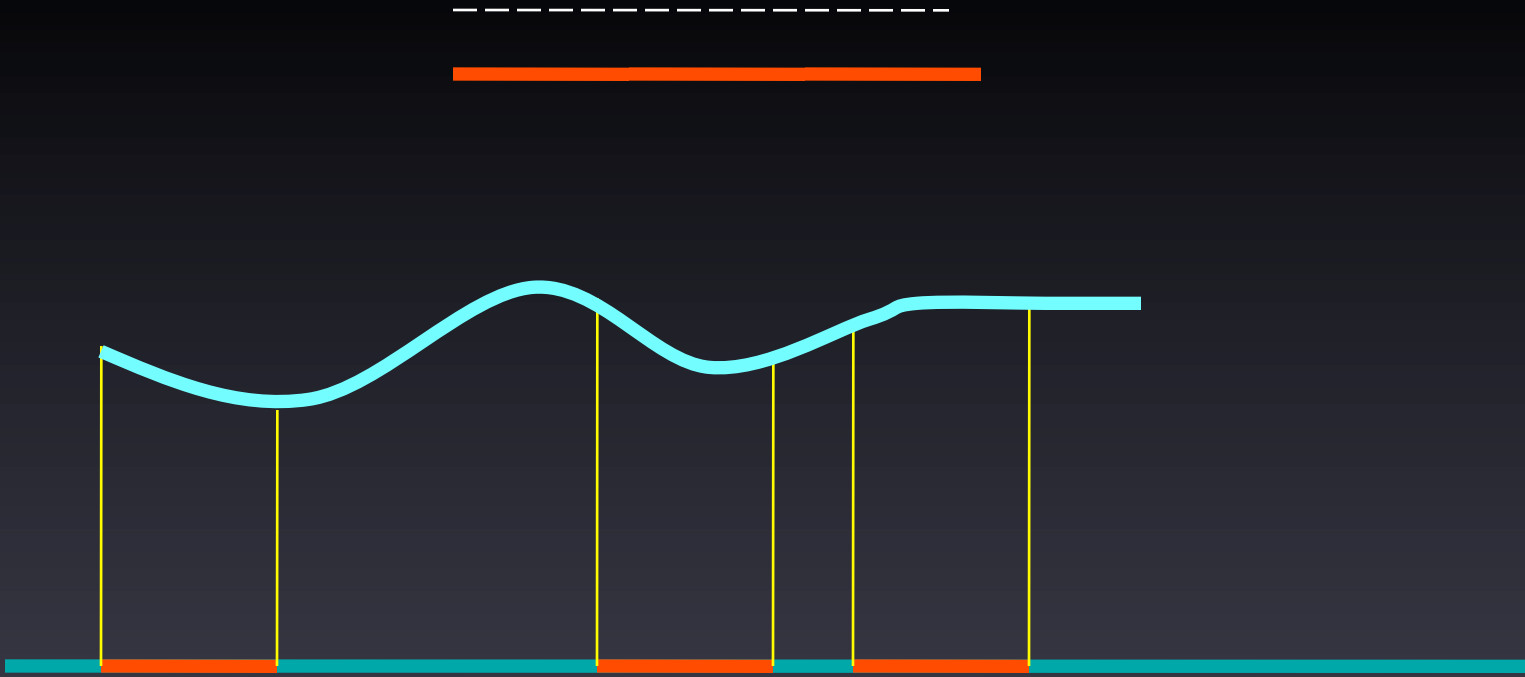
- 1965 Sequence of the first RNA molecule
- 1977 Northern blot and Sanger sequencing
- 1989 RT-PCR experiments for transcriptome analysis
- 1991 First high-throughput EST sequencing study

- 1992 Differential Display (DD) : differentially expressed genes
- 1995 Microarray and Serial Analysis of Gene Expression (SAGE)
- 2001 Draft of the Human Genome “*completed*”
- 2005 First next-generation sequencing technology (454/Roche)
- 2006 First transcriptome sequencing using a next-generation technology (454/Roche)
- 2008 First stem cell transcriptome NGS (SOLiD) Applied Biosystems

Assembly and Transcript reconstruction



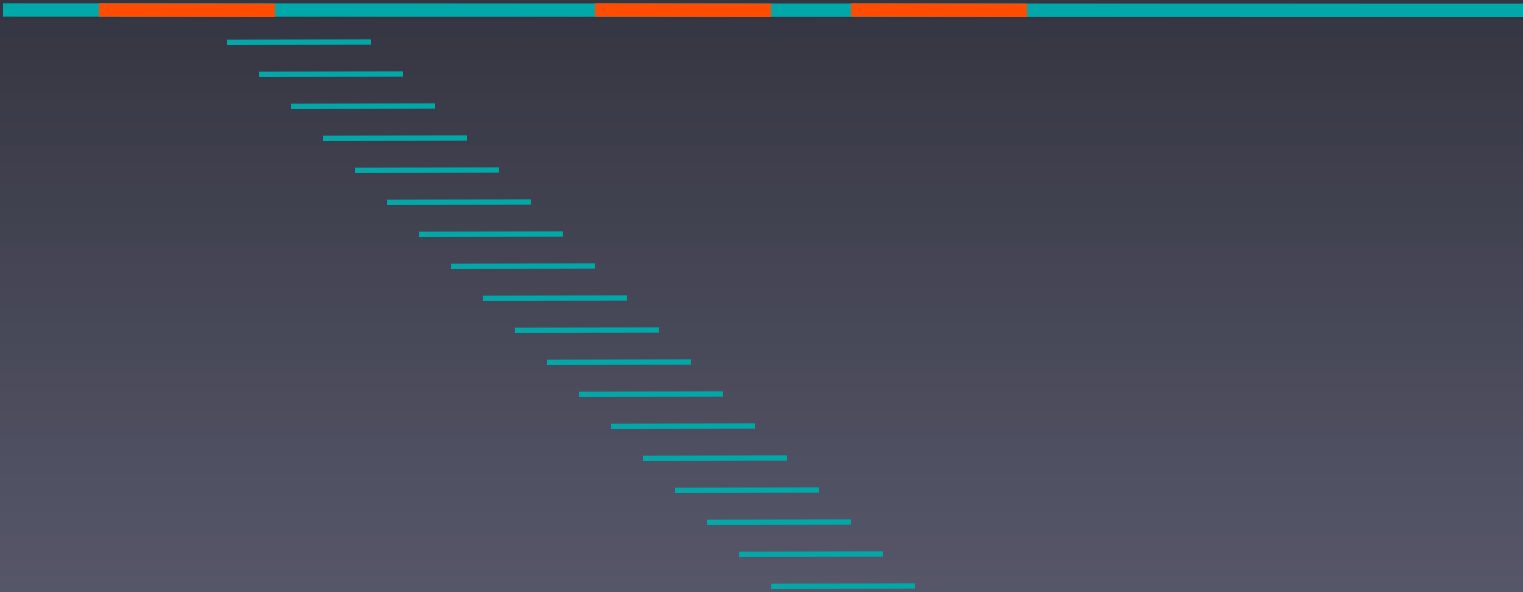
Assembly, Coverage,
Bermuda



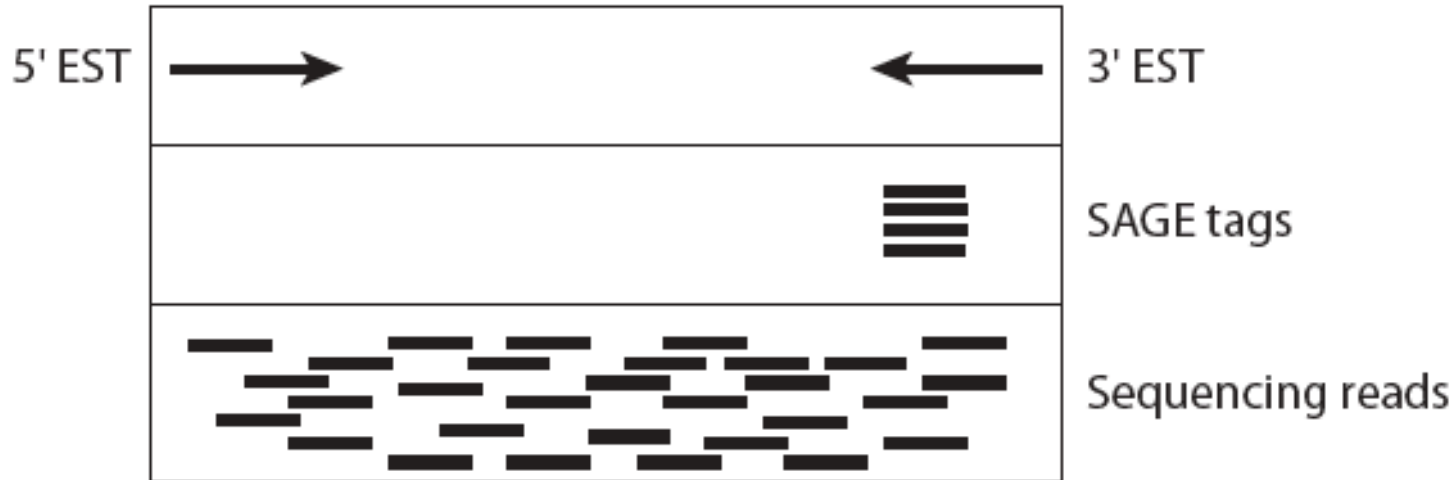
exons
information to finished
transcript
coverage of finished
transcript



**Making assembly tools
work for transcript
reconstruction**



Reference genome sequence

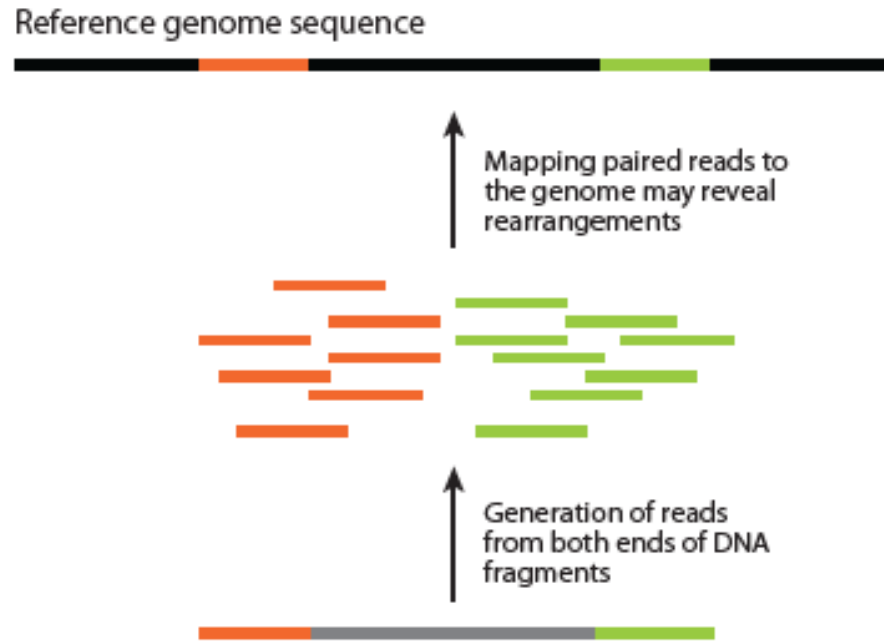


Morozova, Annu. Rev. Genom. Human Genet.
2009

Protein coding exon discovery



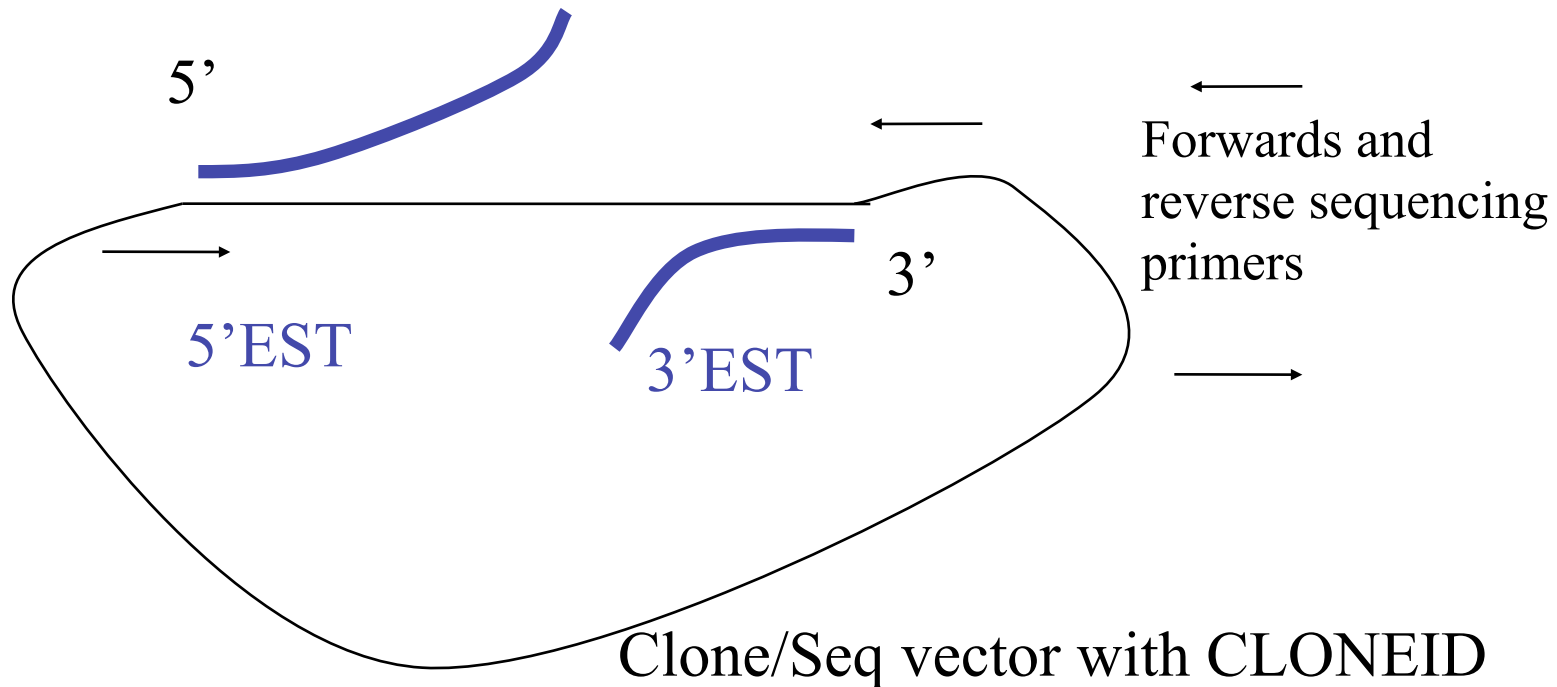
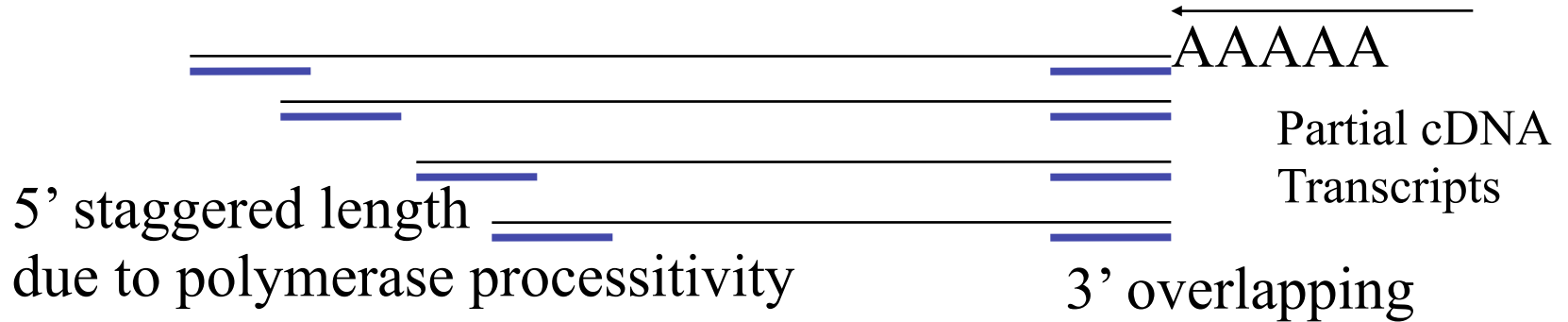
Transcript abberancy



Techno Acronymymania

- EST
- SAGE
- Array
- Northern
- RT-PCR
- CAGE
- DiTag
- RACE
- MPSS
- flcDNA

What is an EST?



What potential do ESTs hold?

- Expression counts
- Consensus sequences
- Alternate expression-form characterisation
- Identification of genes expressed in a pilot gene discovery project
- Identification of genes specifically expressed in a chosen library or tissue

EST Data

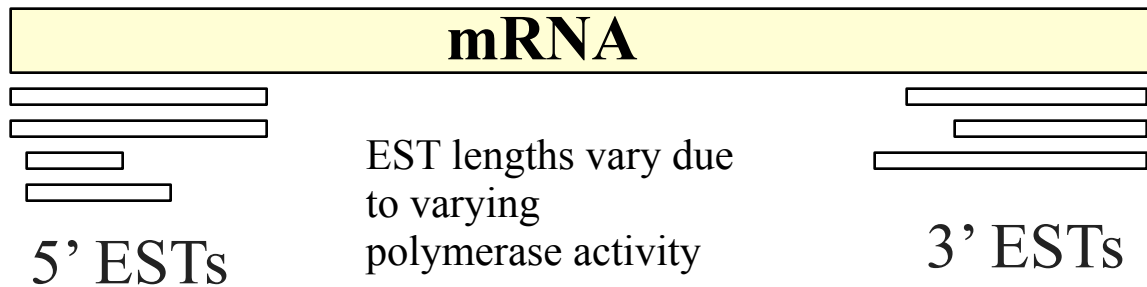
Expressed Sequence Tags

Single pass sequences of cDNA clones from different libraries

High error rate (>1%) mainly frameshifts and insertions/deletions

Redundant sampling of 5' and 3' ends

Large number in public databases

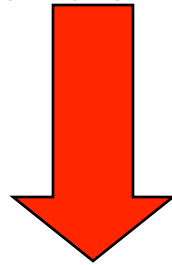


No single mRNA/transcript depository

Extracting Value from Transcript Data

Clustering

Large amount of unorganised, poor quality data



Smaller amount of indexed, “good” quality data

What is a Cluster?

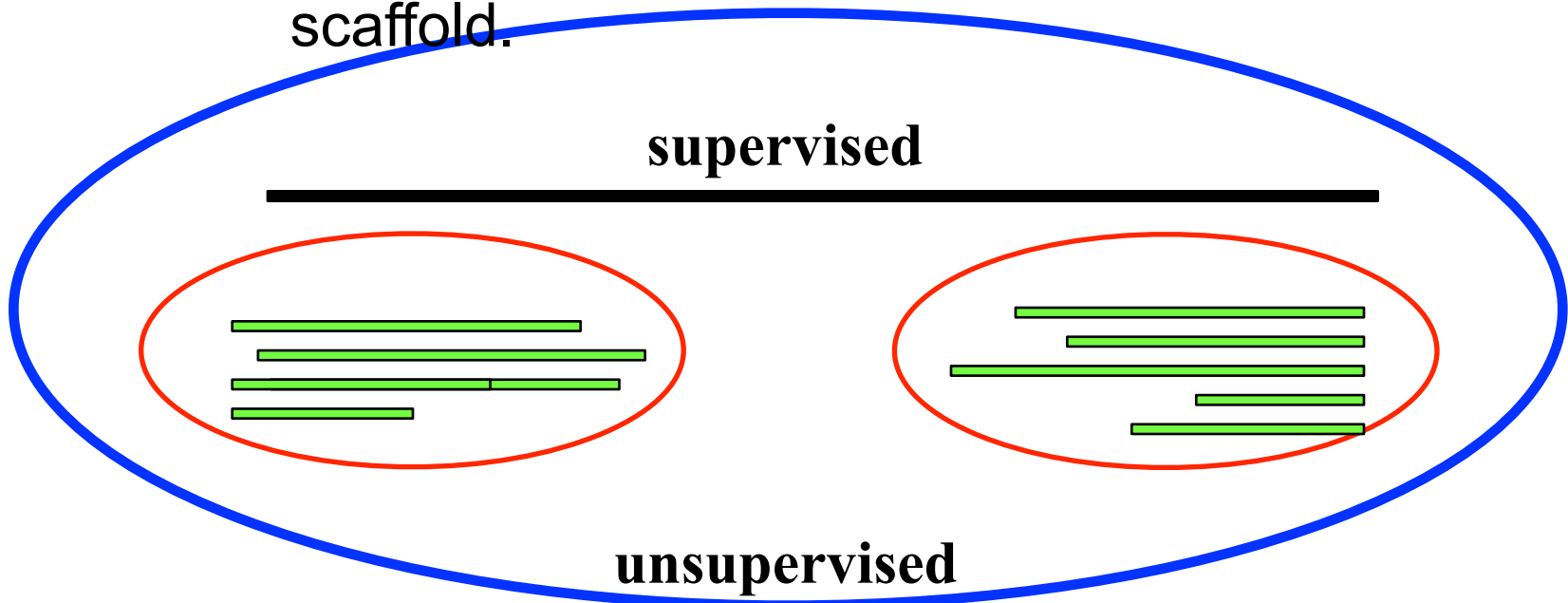
Grouping of expressed sequences such that:

- 1. All expressed sequences representing a single gene are in a single index class (1 cluster)**
- 2. Each index class (cluster) contains the information for only one gene**

1 cluster = 1 gene

Approaches

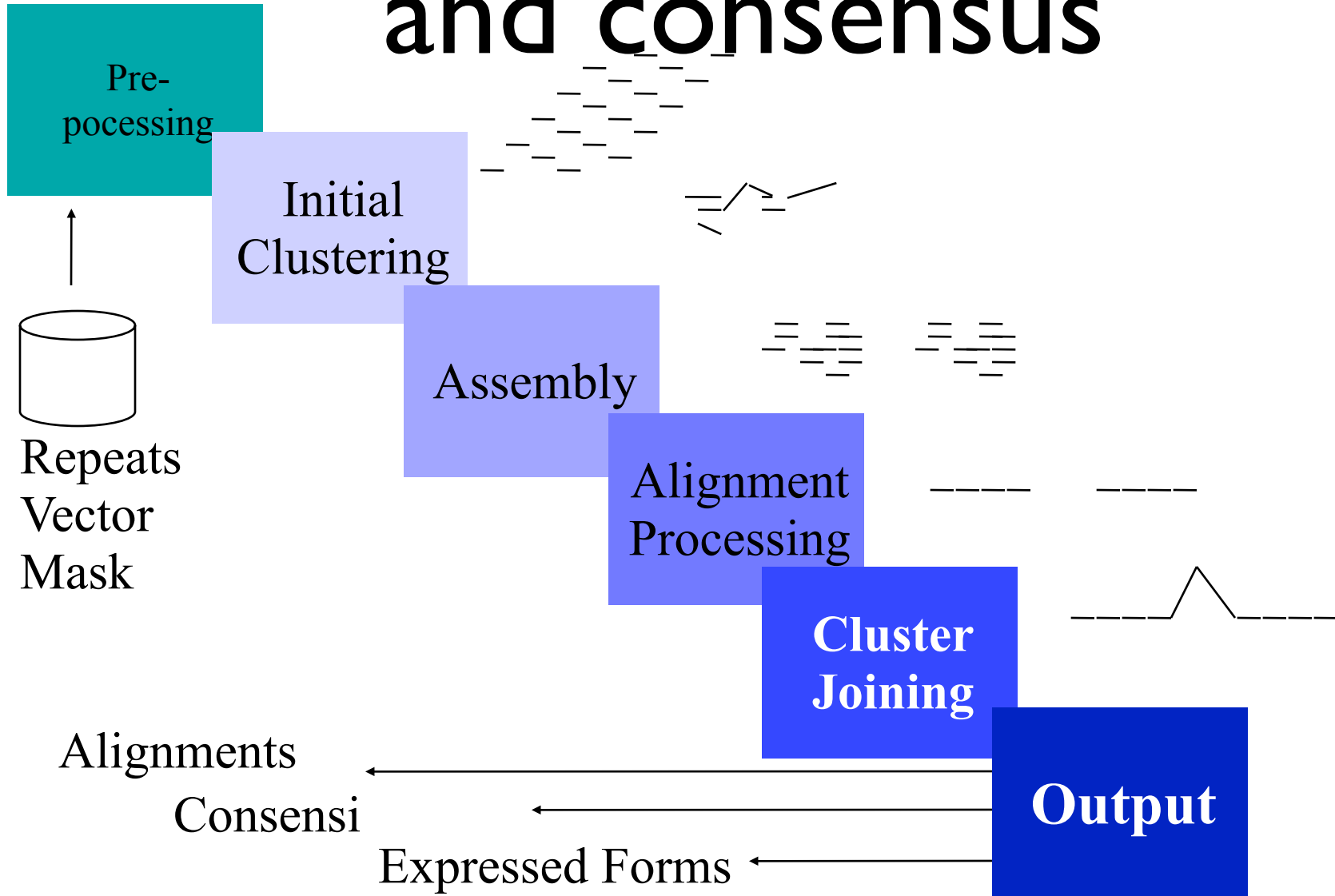
- Unsupervised Transcript Clustering
 - Cluster ESTs with other ESTs
- Supervised Transcript Clustering
 - Cluster ESTs using an mRNA / genomic sequence as a scaffold.

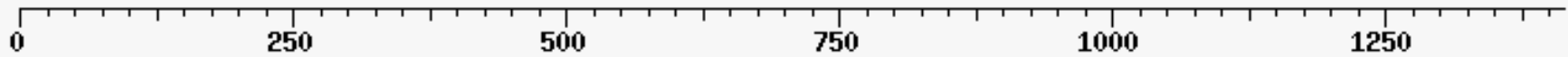


Levels of clustering

- Transcript level
 - Transcripts are compared for genetic similarity
 - Shared word frequencies or short aligned regions (40bp)
- Gene Level
 - Consensus sequences from transcript comparisons
 - Grouped assembly consensus sequences from transcript clusters
 - Should cover the genome equivalent

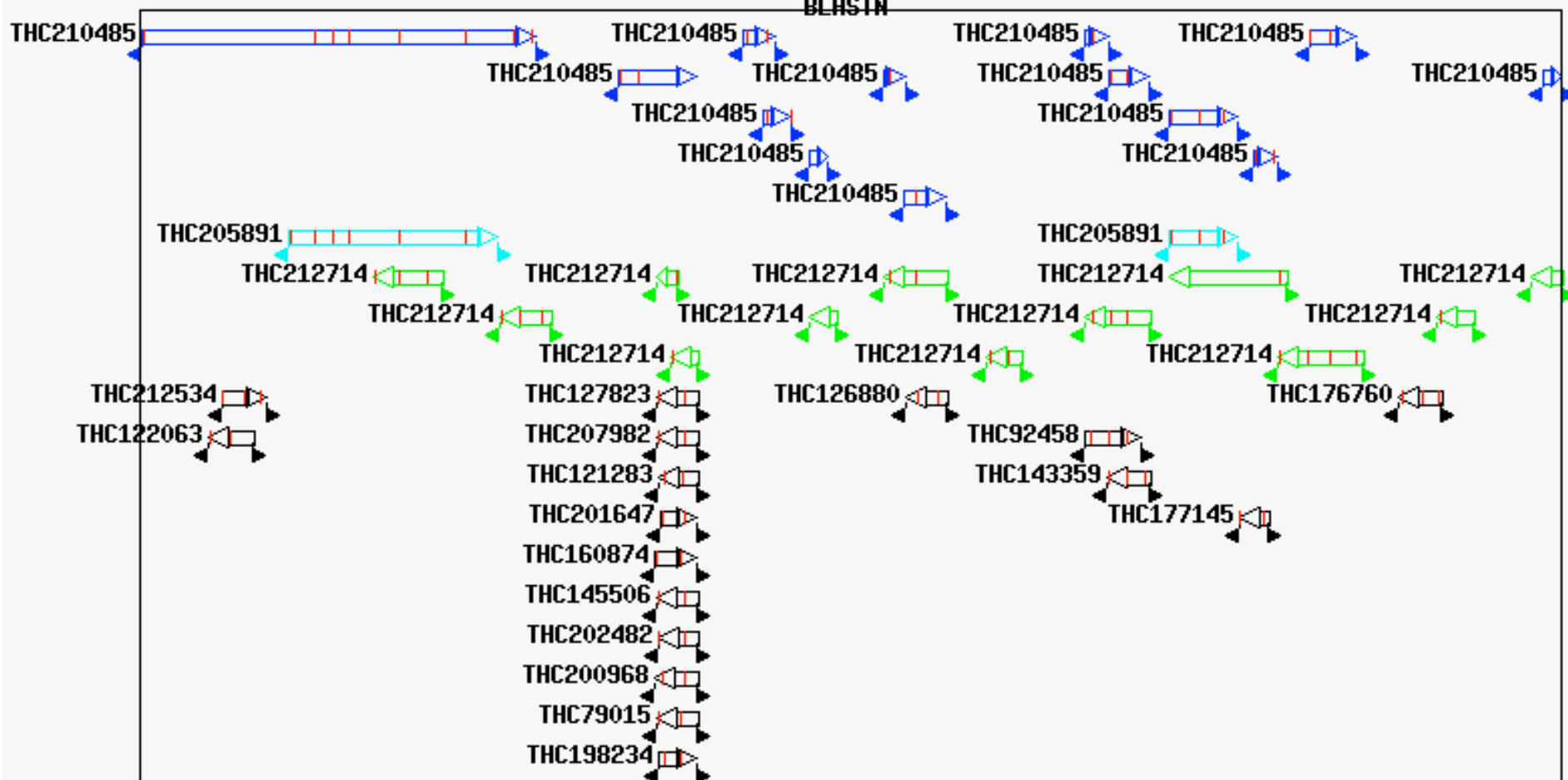
Overview of clustering and consensus

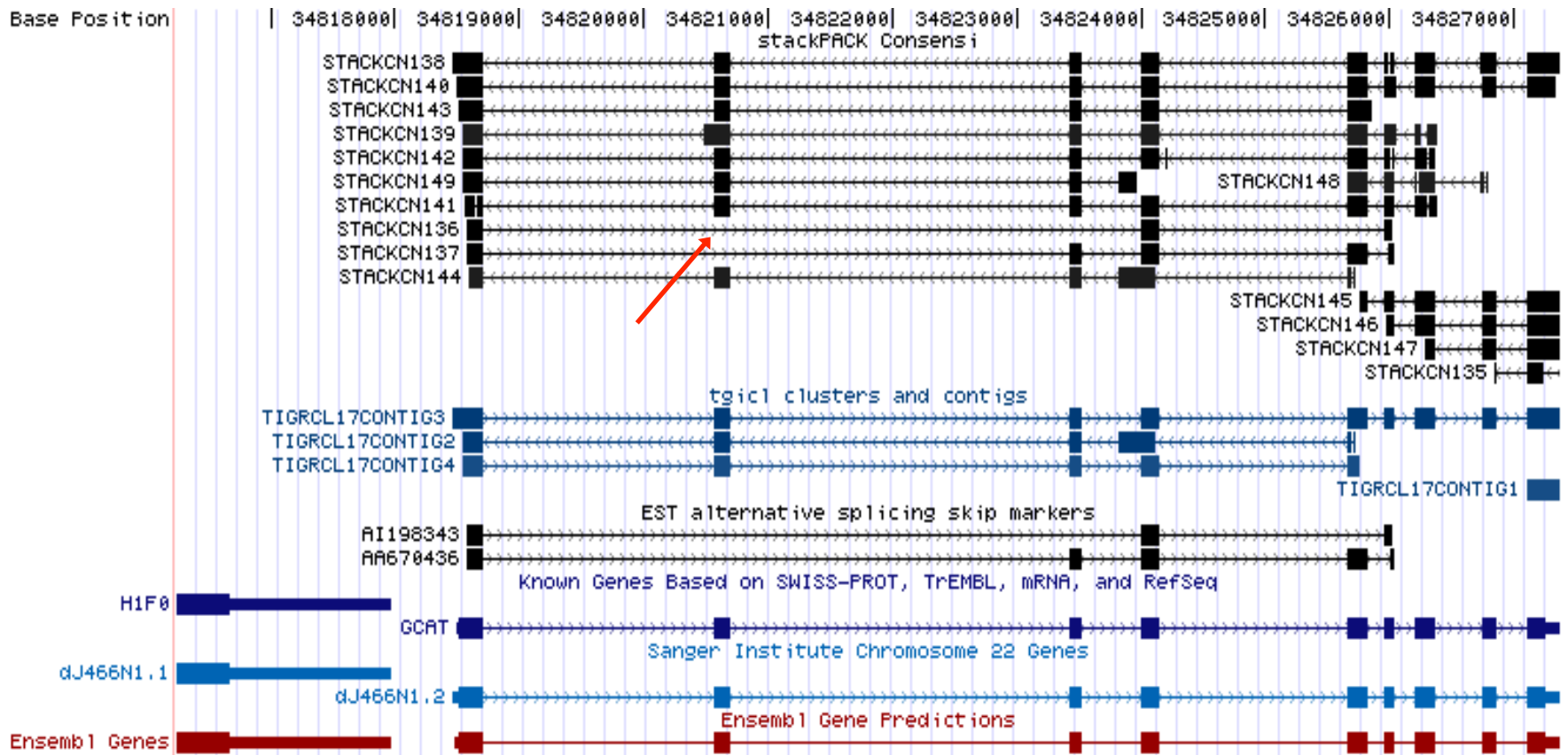




eye2

BLASTN





UCSC human genome browser view of GCAT on chromosome 22
 -Exon 5 skipped in stackPACK transcript
 -Missing in TGICL transcripts (red arrow).

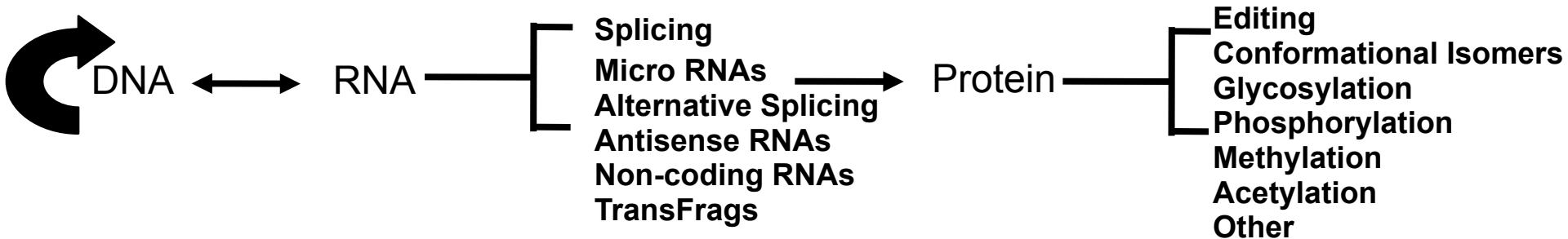
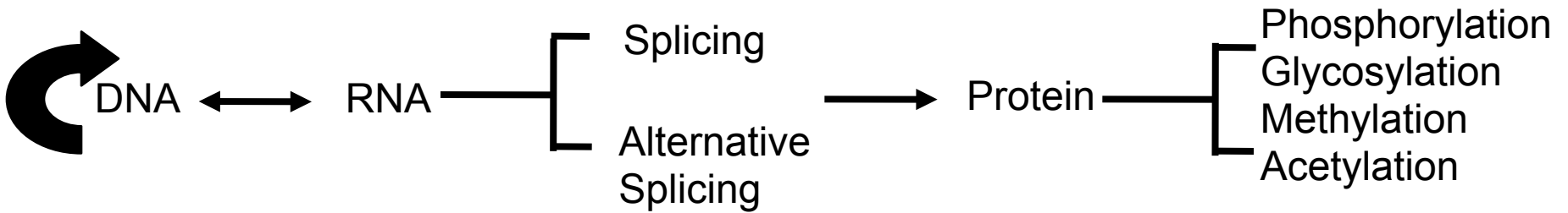
Gene

- A protein coding region of the genome?
 - Transcript contains a protein coding sequence
- One transcript for each protein coding sequence
- Transcript diversity = protein coding diversity

Gene history

- Physical element connected to a phenotype (Johannsen)
- Central dogma

Evolving Dogma (Paul Silverman)



Transcript

- Transcripts that do not have coding sequence
- Transcripts that initiate at different locations upstream of the CDS
- Different CDS
 - Skips
 - Altered donor and acceptor site
 - Altered poly-adenylation

Genome Products?

- Diaspora of transcripts
- Inconsistent description and organisation prevents large scale discovery
- Poor understanding of gene structure provides new challenges
 - Affymetrix analysis of chromosome 21,22 yields 10X greater number of transcribed regions than ‘known’ protein coding genes.

- Kapranov et al *Science*, 2002: 296:916-919

Characterising diversity

- Examine gene expression product diversity at alternate splicing level
- Capture gene expression products under well defined conditions

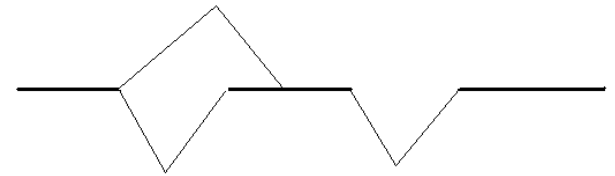
Expression Forms

Exon boundary variation

Exon extension/truncation

Alternative transcription start sites

Alternative polyadenylation



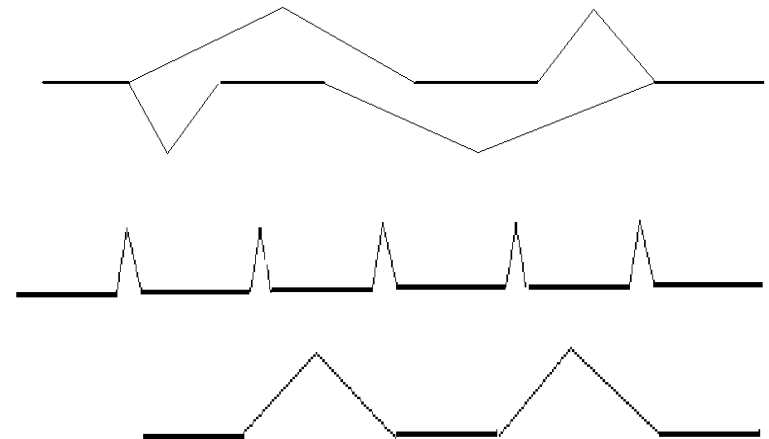
Whole-exon events

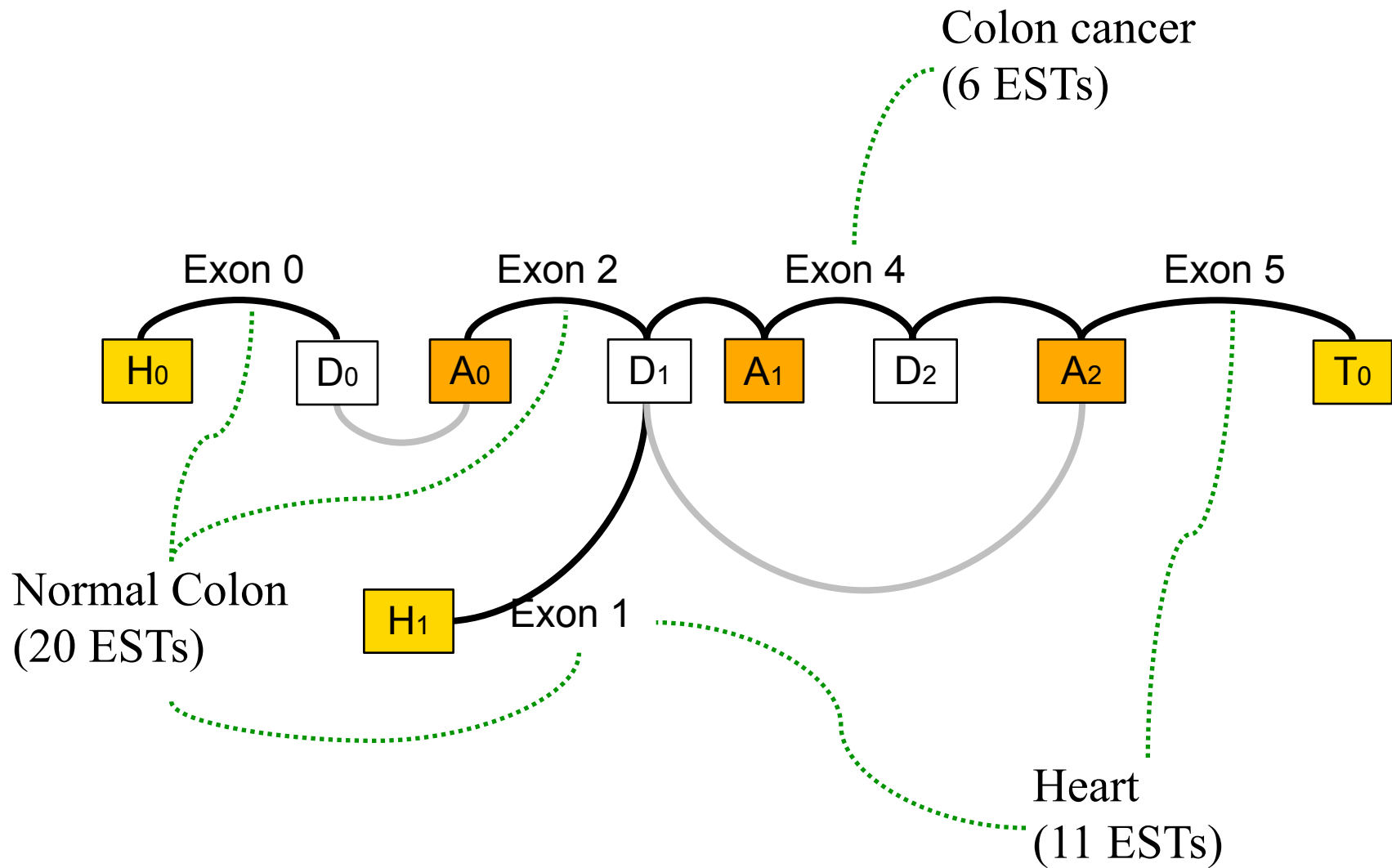
Skipping

Cryptic exons/introns

Exon repetition

Complex events





comparing splice events, not fragments of transcript isoforms

Expression Capture

- Serial Analysis of Gene Expression
 - DNA fragments that act as unique markers of gene transcripts.
 - Assay of numbers of each marker in a set of sequence yields a measure of gene expression
- Array
 - Laydown of sequence clones to provide an organised series for hybridisation

Resolution of Captured Expression

ESTs Low resolution, broad capture, provides template for SAGE and Array

SAGE Medium resolution, need template, noise can be an issue, stoichiometry is revealed but standardisation a problem

ARRAY High resolution, need template, noise, stoichiometric resolution highest, standardisation a problem.

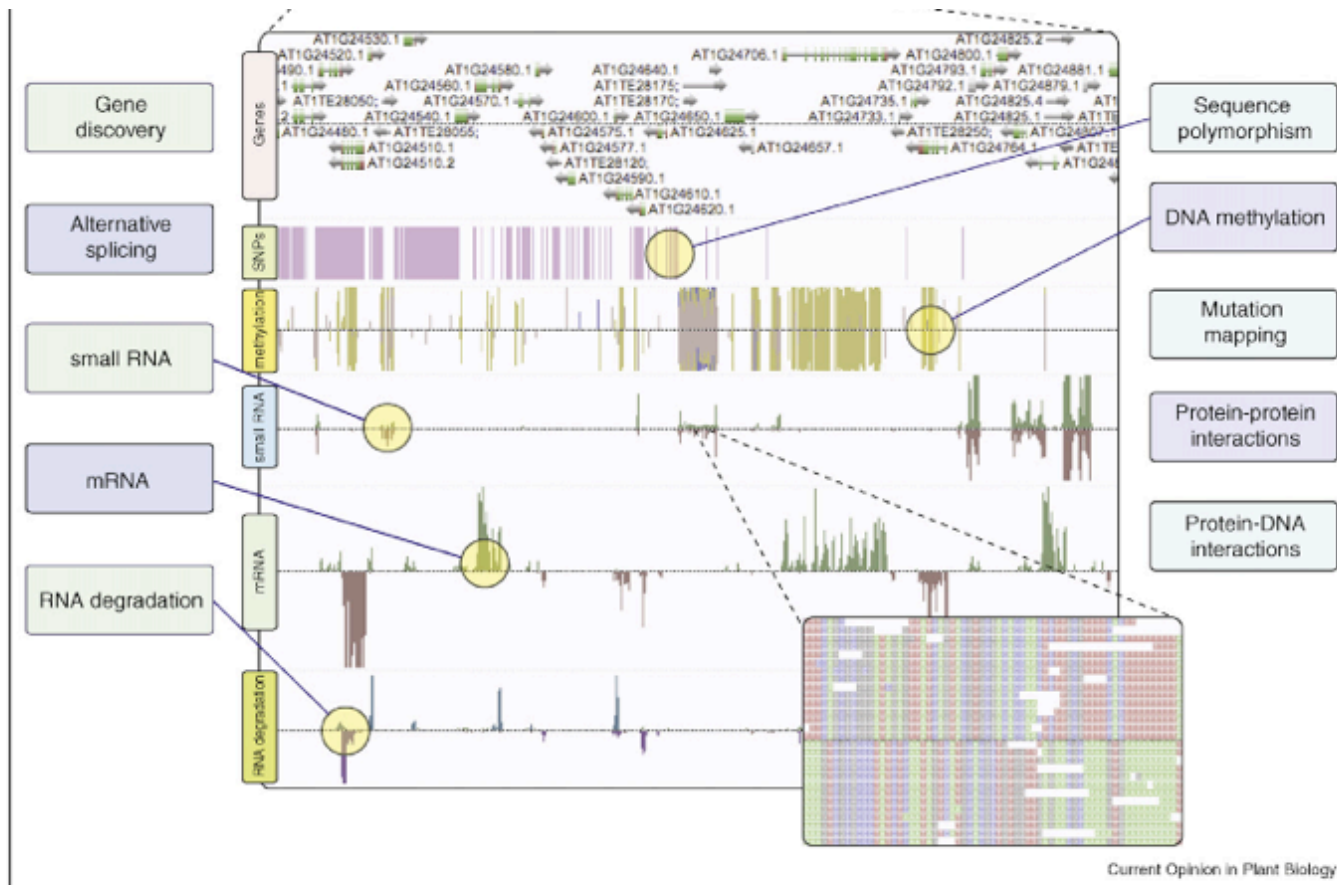
Captured Expression

- RNAseq

High coverage, high resolution, multiple transcripts for each parent region, identification of previously unknown genes and alternative splice variants; read mapping dependent

- Capped Analysis of Gene Expression

Comprehensive capture of transcription start sites. Deep coverage, discovery of novel start sites. Read mapping dependent. Gene mapping dependent.



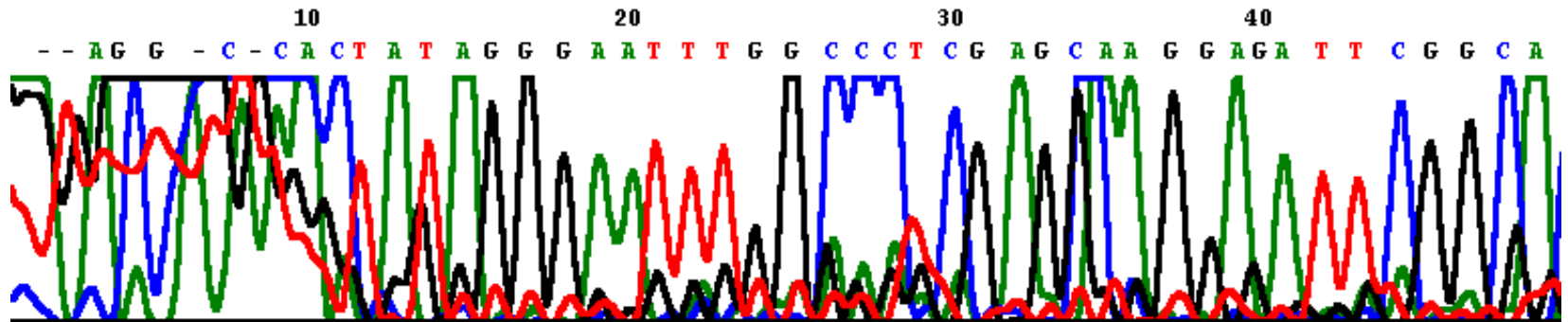
Current Opinion in Plant Biology

Advanced DNA sequencing technologies underly diverse approaches to unravel plant cellular activities. Massively parallel DNA sequencing of complex nucleic acid populations now enables numerous subsets of genomic and cellular information to be rapidly characterized at unprecedented resolution and breadth. The AnnoJ genome browser (www.anno.j.org) excerpt shown above represents approximately 100 kb of *Arabidopsis thaliana* chromosome 1. Single nucleotide polymorphisms between Col-0 and Ler-1 ecotypes (Lister, O'Malley, Ecker, unpublished), single-base DNA methylation maps, strand-specific smRNA and mRNA components of the transcriptome, and RNA-degradation products from *Arabidopsis thaliana* flower buds, all generated by ultra high-throughput DNA sequencing, have been integrated to illustrate the holistic views of genomic and transcriptional regulation and variation that can now be routinely captured [42,49].

Why is transcript data a problem?

data quality

```
>T27784   g609882 | T27784 CLONE_LIB: Human Endothelial cells. LEN: 337
b.p. FILE gbest3.seq 5-PRIME DEFN: EST16067 Homo sapiens cDNA 5' end
AAGACCCCGTCTCTTTAAAAATATATATATTTTAAATATACTTAAATATATATTTCTAATATCTTTAAAT
ATATATATATATTTNAAAGACCAATTTATGGGAGANTTGCACACAGATGTGAAATGAATGTAATCTAATAG
ANGCCTAATCAGCCCACCATGTTCTCCACTGAAAAATCCTCTTTCTTTGGGGTTTTTCTTTCTTTCTTTTT
TGATTTTGCACCTGGACGGTGACGTCAGCCATGTACAGGATCCACAGGGGTGGTGTCAAATGCTATTGAAAT
TNTGTTGAATTGTATACTTTTTCACTTTTTGATAATTAACCATGTAAAAAATG
```



Single read errors

Vector

Repeat MASK

Individual items are prone to error but an entire collection contains valuable genetic information

Search SRA for baboon [Save Search](#)

Limits Preview/Index History Clipboard Details
 Display Summary Show 20 Send to


All: 1

1: [SRX000433](#) 454 sequencing of Papio anubis placenta transcript fragment library [Links](#)

Submitter: UWBL, University of Washington, Bumgarner Lab
Study: Evolutionary insights from large scale Papio anubis cDNA sequencing project (SRP000221) • [Summary](#) • [Genome Project](#) • [All experiments](#)
Sample: [olive baboon \(SRS000538\)](#)
Instrument: GS FLX

 [Download data for this experiment SRX000433](#)

Total: 3 runs, 1M spots, 268M bases			
#	Run	# of Spots	# of Bases
1.	SRR001693	315,491	81.1M
2.	SRR001694	341,165	87.9M
3.	SRR001695	382,568	99M

[Write to the Help Desk](#)
[NCBI](#) | [NLM](#) | [NIH](#)
 Department of Health & Human Services 
[Privacy Statement](#) | [Freedom of Information Act](#) | [Disclaimer](#)

RIP

Sampling

- RNA hybridization on high density arrays
 - Specified probes
 - Tiling array
 - Representation of splice junctions
 - Data interpretation
- Digital transcript counting
 - Avoids complex normalization
 - Captures lows abundance

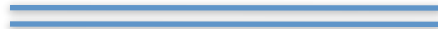
RNA-Seq

- Short read
- High throughput
 - Colony multiplex
 - 0.3 RNA copies/cell
 - 27bp length

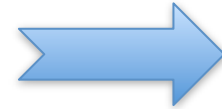
mRNA



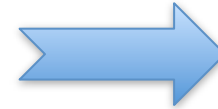
cDNA



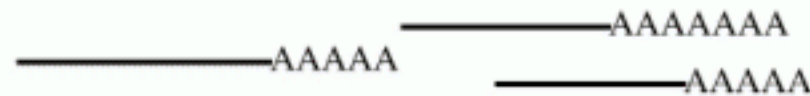
Sheared cDNA



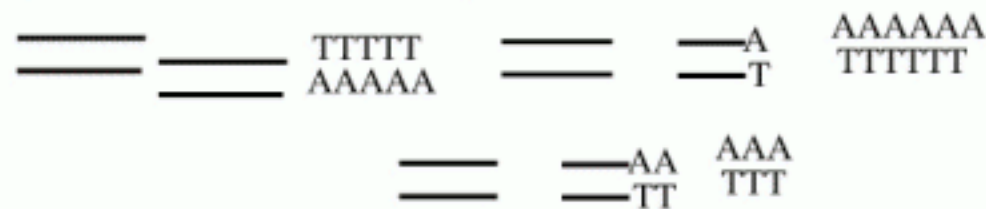
cDNA



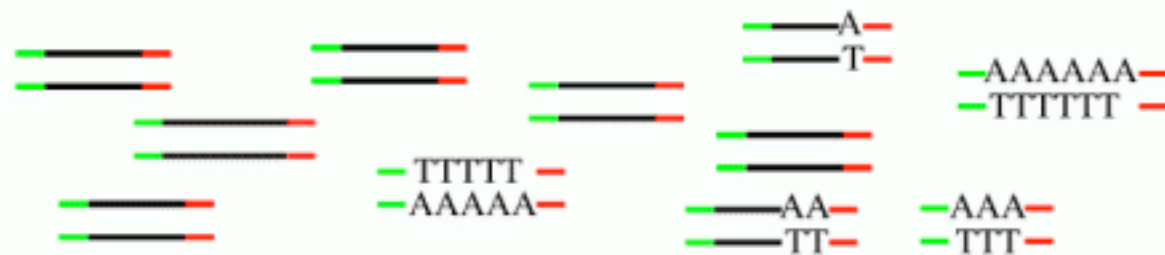
extraction of poly-A RNAs



conversion into ds-cDNA
and shearing

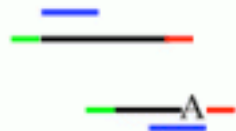


amplification and
adapter ligation

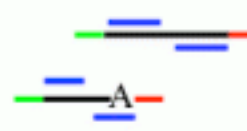


sequencing

single end (SET)

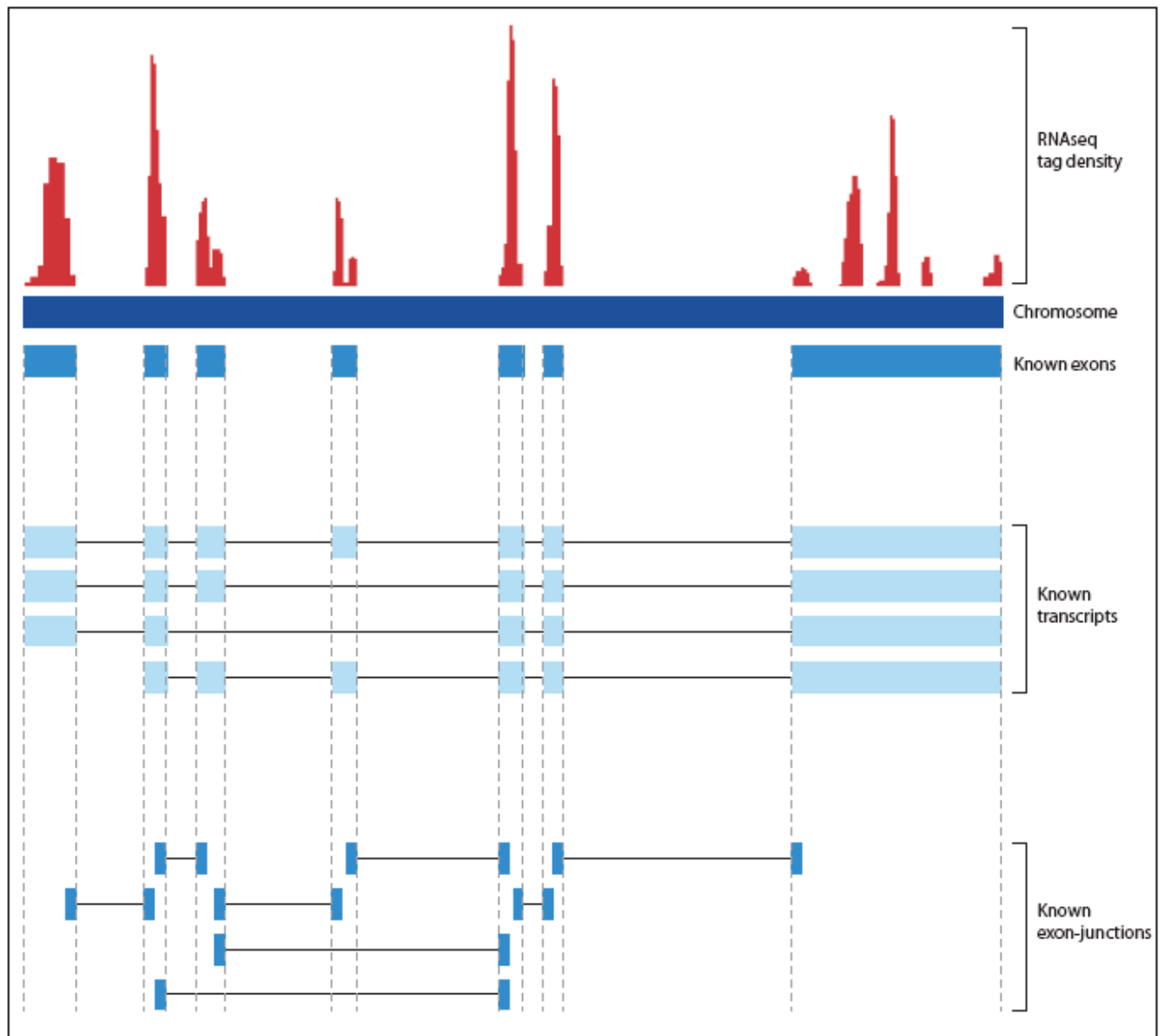


paired-end (PET)



Human RNA-seq

- 2 mismatch
- 50% unique map
 - 80% known exons
- 18% multiple map
- 25% no match
- Detects 25% more genes than array



Grimond et al 2008

Mapping example

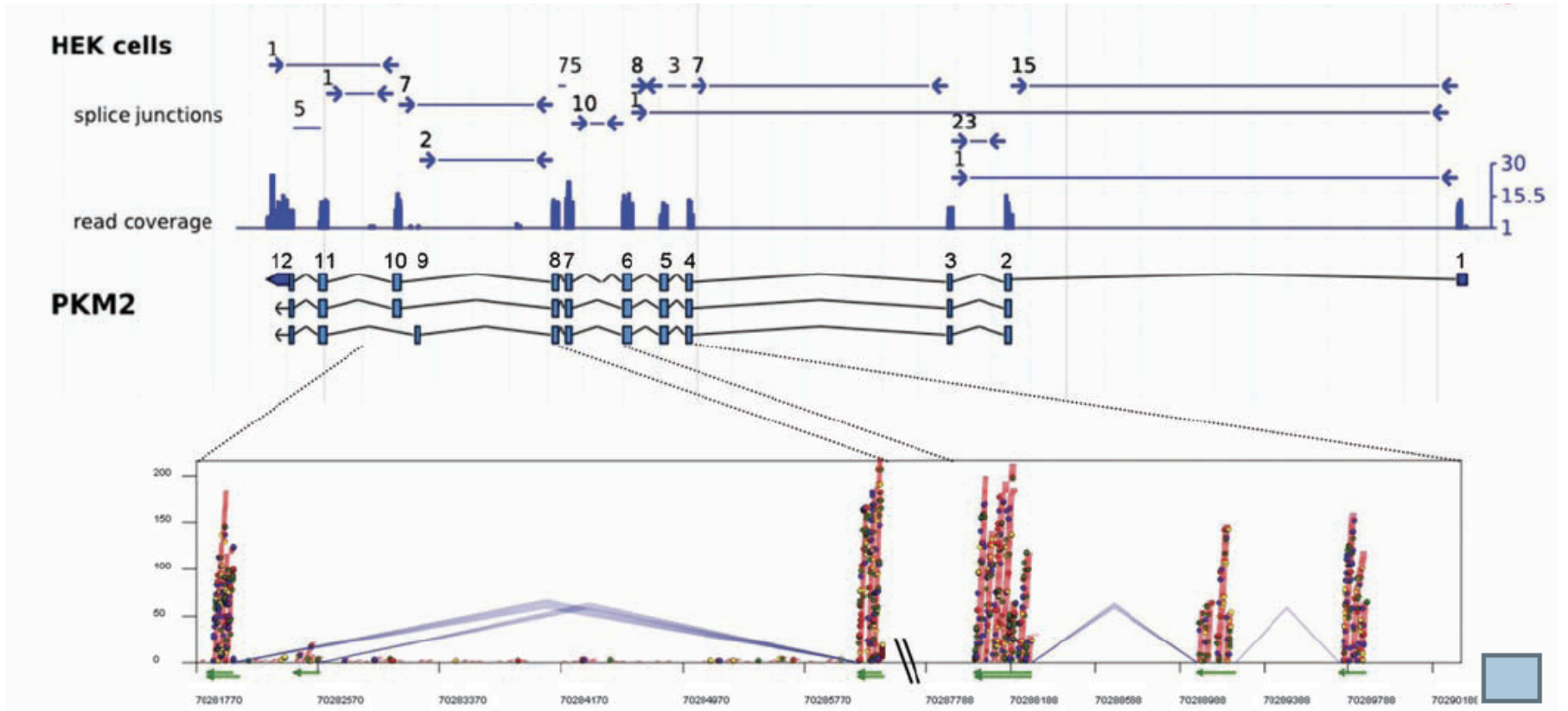
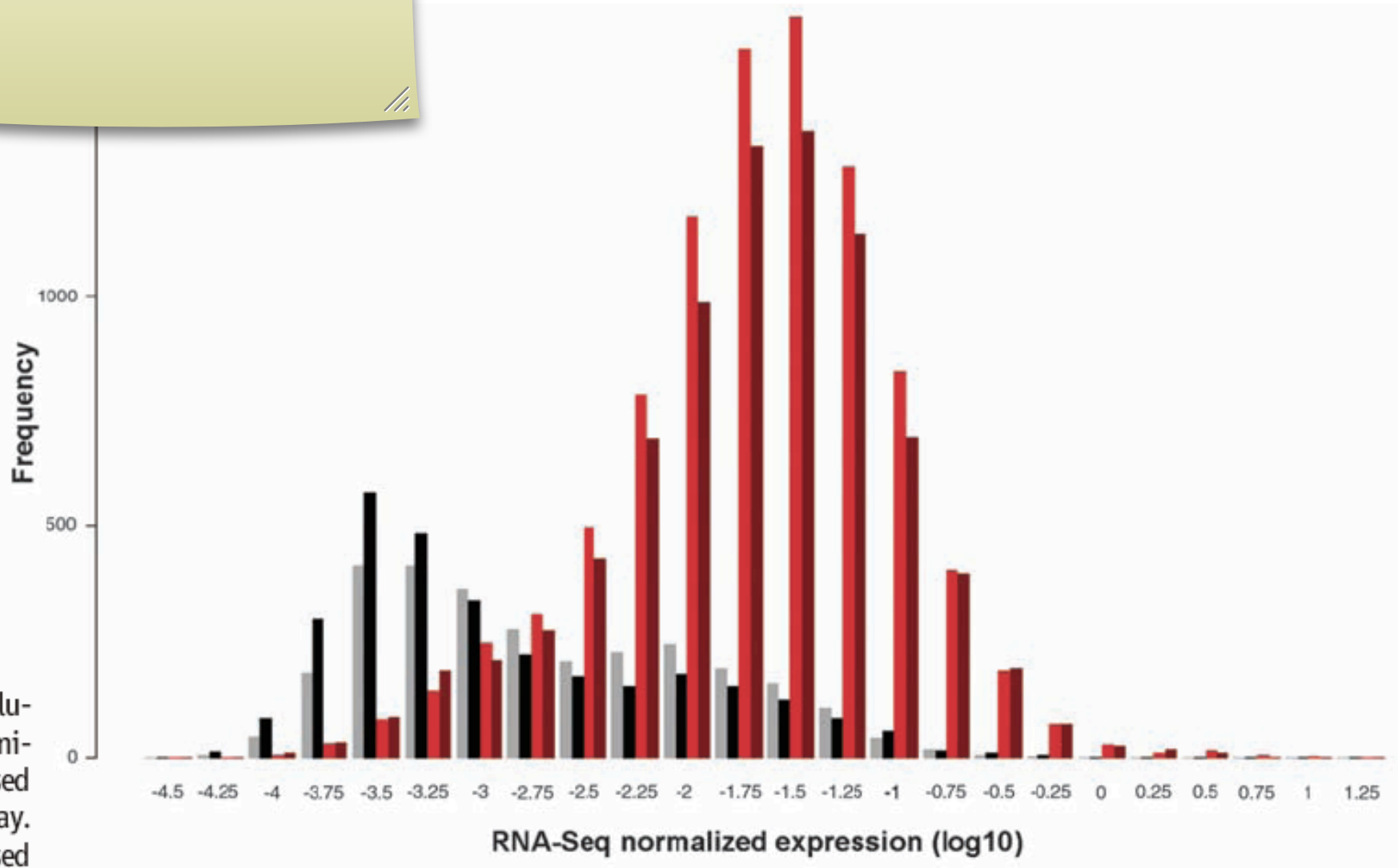


Fig. 3. ΔC events observed by junction reads. (A) Distribution of the three junction reads are shown as arrows; the numbers above the arrows

Winston Hide
10/21/08
gray missed on array

relative sampling



Reproducibility

- 0.99 pearson for replicate RNA runs
- Raw tag \Leftrightarrow Q-RT-PCR
- Array dynamic range
 - 4-5 orders
 - Saturates

Sensitivity

- 10-100 Mill reads/sample
- 1-4Gb reads for mammalian complexity
- 75% cross-hyb with probes

RNA-seq vs Array

- no probe cross-hybridization
- greater dynamic range
- higher sensitivity and specificity
- detection of more, shorter, low abundance transcripts
- discrimination between similar sequences.

RNA-seq

- Sequence-level transcript information
- distinguish between paralogous parent genes
- replicable digital quantification based upon counting of sequence reads
- identify transcript sequence polymorphisms

Limits

- new-generation sequencers
 - short read length
 - high error
- Read Mappnig
 - Consistency and standards
 - sequence assembly

Limits II

- data analysis
 - computational tools
- massive sequence data volume

Next Next

- single reads will grow in length and number
- accurate reads with length of 100,000 bp.
(Pacific Biosciences)

Data manipulation and interpretation

- How do you look for mutations in an HIV sequence when 75 000 reads cover the same base pair?

How to map billions of short reads onto genomes

Cole Trapnell & Steven L Salzberg

Mapping the vast quantities of short sequence fragments produced by next-generation sequencing platforms is a challenge. What programs are available and how do they work?

A new generation of DNA sequencers that can rapidly and inexpensively sequence billions of bases is transforming genomic science. These new machines are quickly becoming the technology of choice for whole-genome sequencing and for a variety of sequencing-based assays,

Table 1 A selection of short-read analysis software

Program	Website	Open source?	Handles ABI color space?	Maximum read length
Bowtie	http://bowtie.cbcb.umd.edu	Yes	No	None
BWA	http://maq.sourceforge.net/bwa-man.shtml	Yes	Yes	None

The no-longer uncharted territory



Results of a genome-wide, collaborative effort to characterize the mouse transcriptome have

approaches, including CAGE (new cap gene expression) and two ditag techniques named as GIS/GSC (GIS, gene identification signature; GSC, gene signature cloning) identify transcriptional start sites and termination sites. Corresponding pairs of sites were identified for 181,047 independent transcripts — this number is an order of magnitude greater than the estimated number of genes in the mouse genome. The discovery can be attributed, at least in part, to the fact that alternative promoters and polyadenylation sites are associated with most transcription units. And at least 65% of transcription units contain several splice variants.

More than a third of the cDNA sequences in the FANTOM3 data set represent non-coding RNAs. The discovery of thousands of

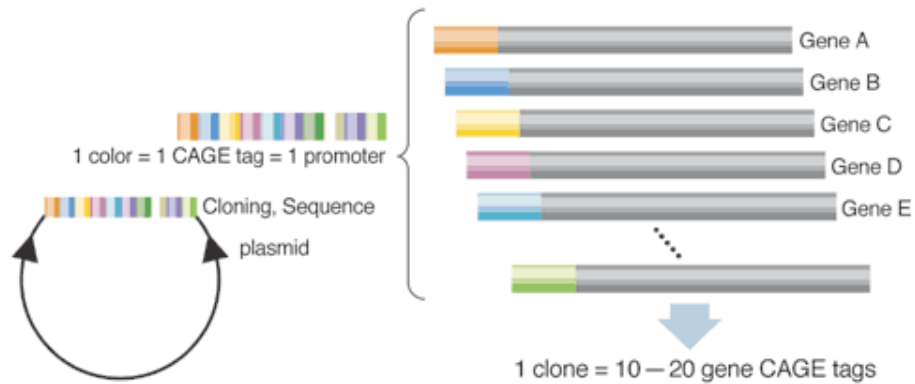
Capped Analysis of Gene Expression (CAGE) technology

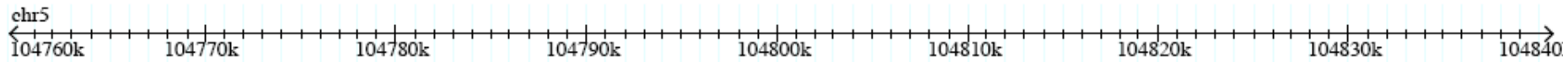
- High-throughput sequencing of tags derived from 5' end of transcript
- Quantify transcript expression levels with internal consistency across different experiments
- Characterize promoter usage

Cap analysis gene expression for high-throughput analysis of transcriptional starting point and identification of promoter usage (Shiraki et al., *PNAS* 2003)

CAGE: cap analysis of gene expression (Kodzius et al., *Nature Methods* 2006)

Capped Analysis of Gene expression

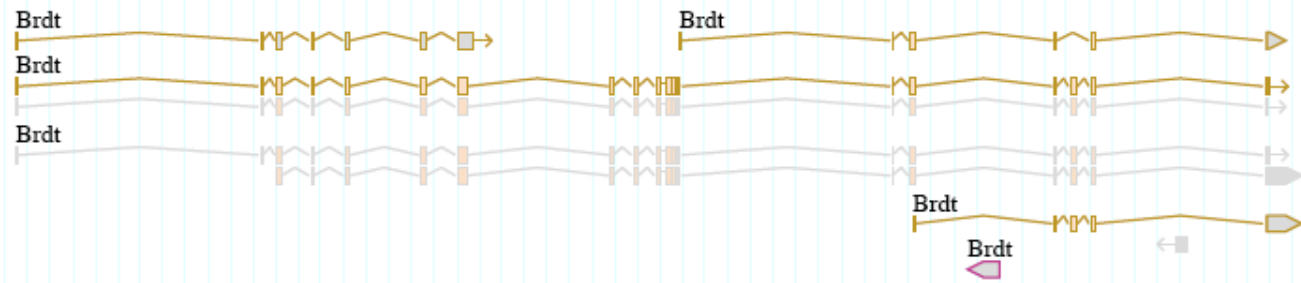




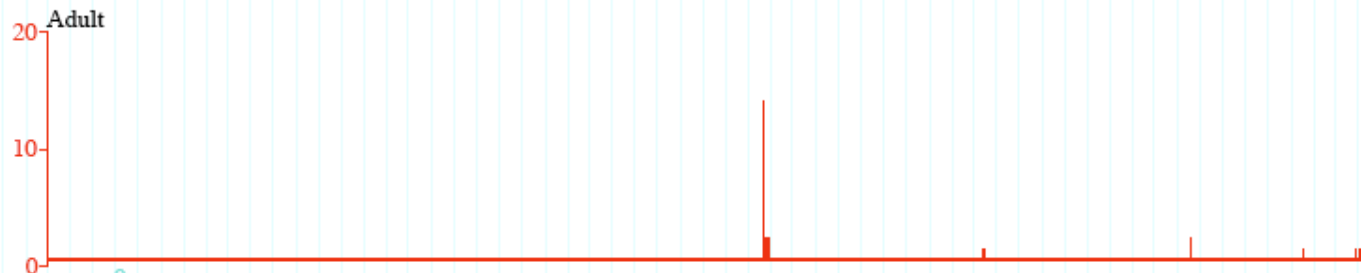
A1. Tag Cluster (TC)



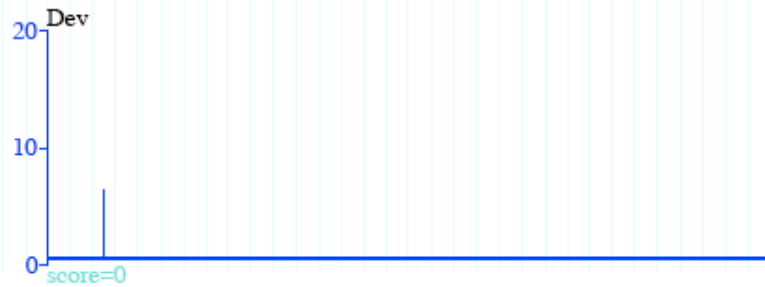
B1. Transcripts

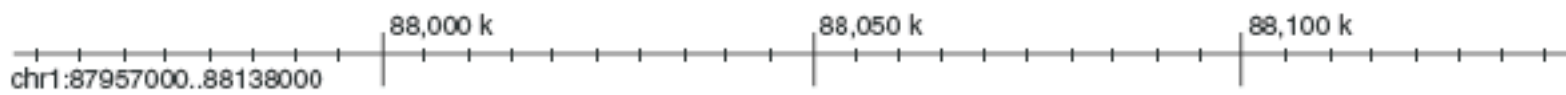


CAGE Tags



CAGE Tags





mRNA mappings

Liver (T01F053E7A52) 76

Adipose (T01F053ED8F9)
Adipose, macrophage (T01F053ED940) 150

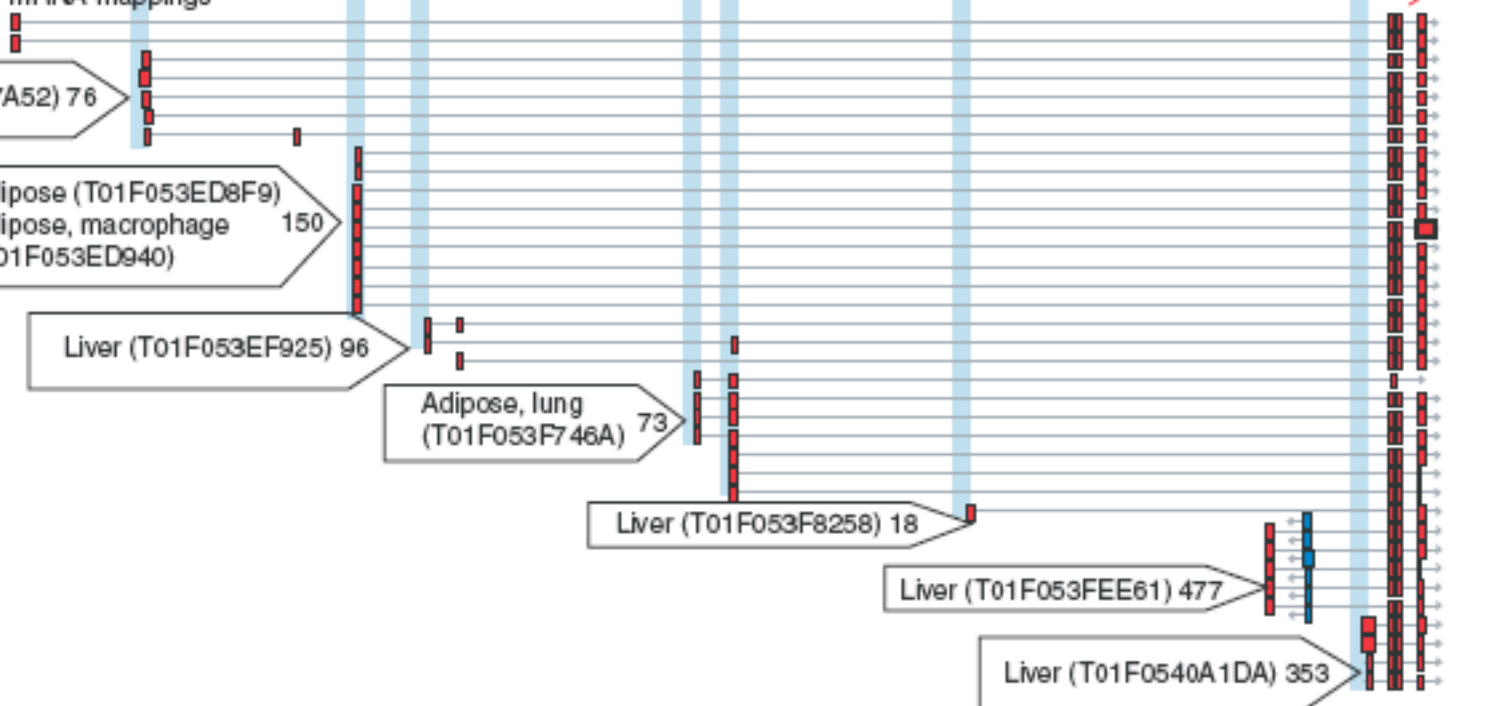
Liver (T01F053EF925) 96

Adipose, lung (T01F053F746A) 73

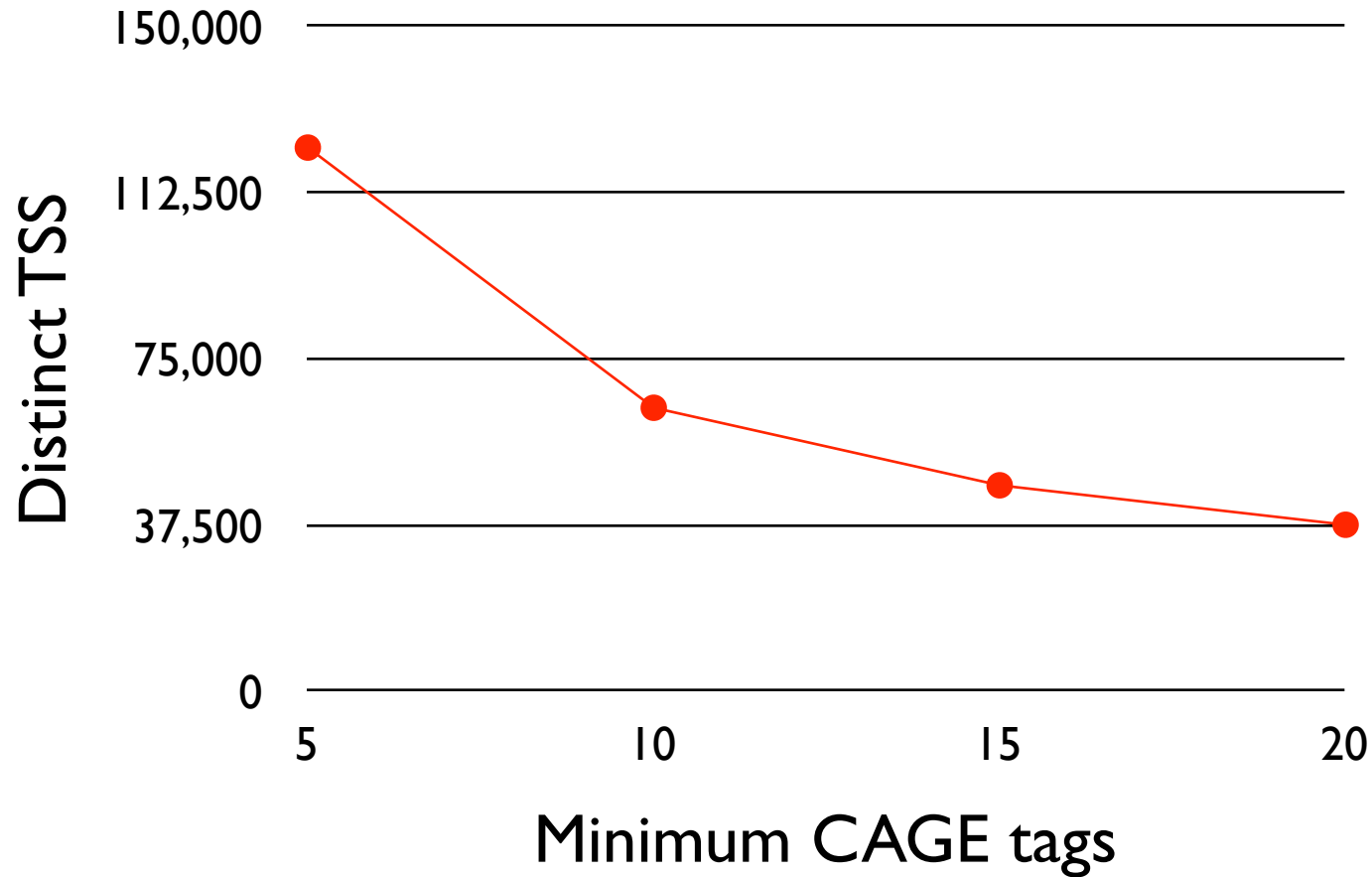
Liver (T01F053F8258) 18

Liver (T01F053FEE61) 477

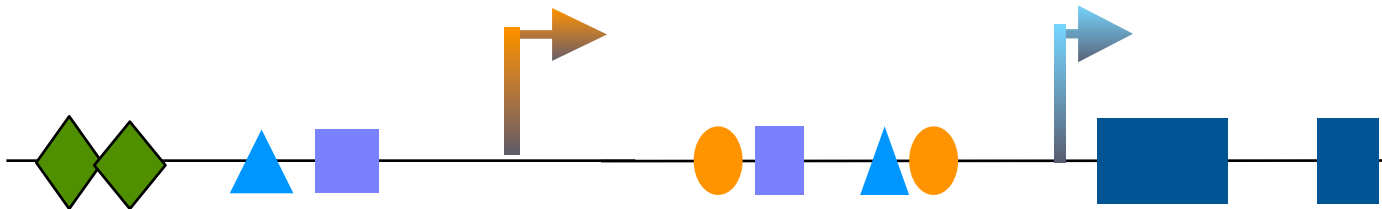
Liver (T01F0540A1DA) 353



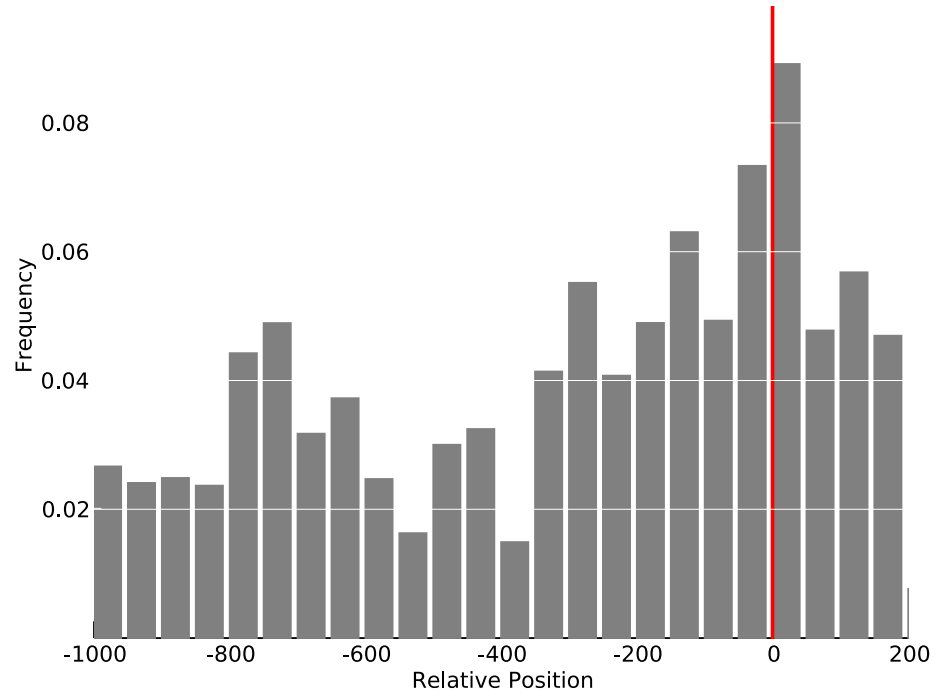
Number of alternative promoter regions



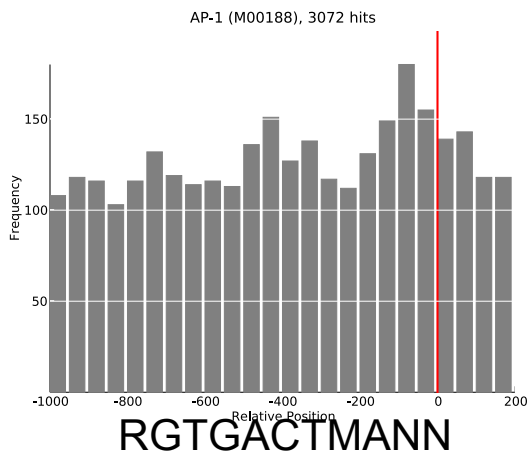
Multiple TSS



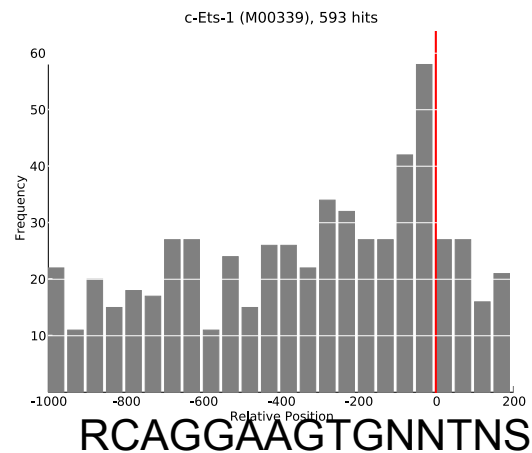
All hits



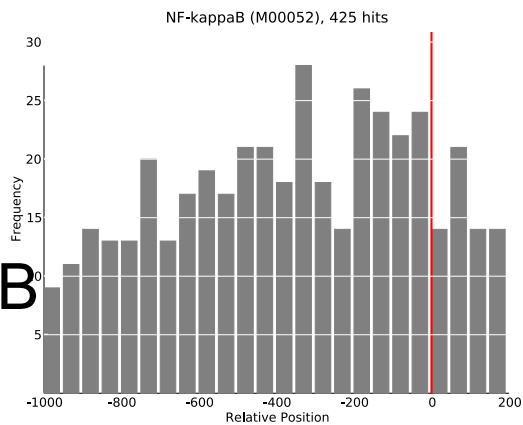
AP-1



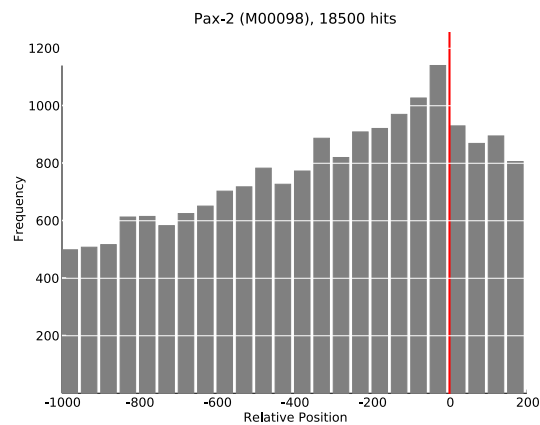
c-Ets-1



NF-kappaB



Pax-2



Exon Painting



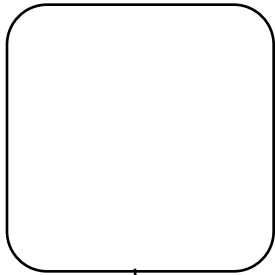
NexGen

- 454 - 900 longest perfect read
- SOLiD - short but plentiful
- Helicos - paired end and cost effective
- Dover (Polonator) - short read low cost

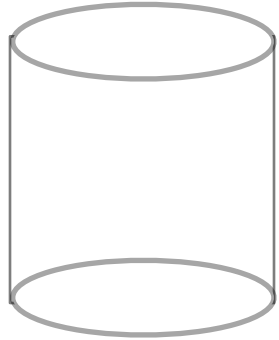
Transcript trends



○ Single Cell Transcriptome



Ever more prolific sequencers



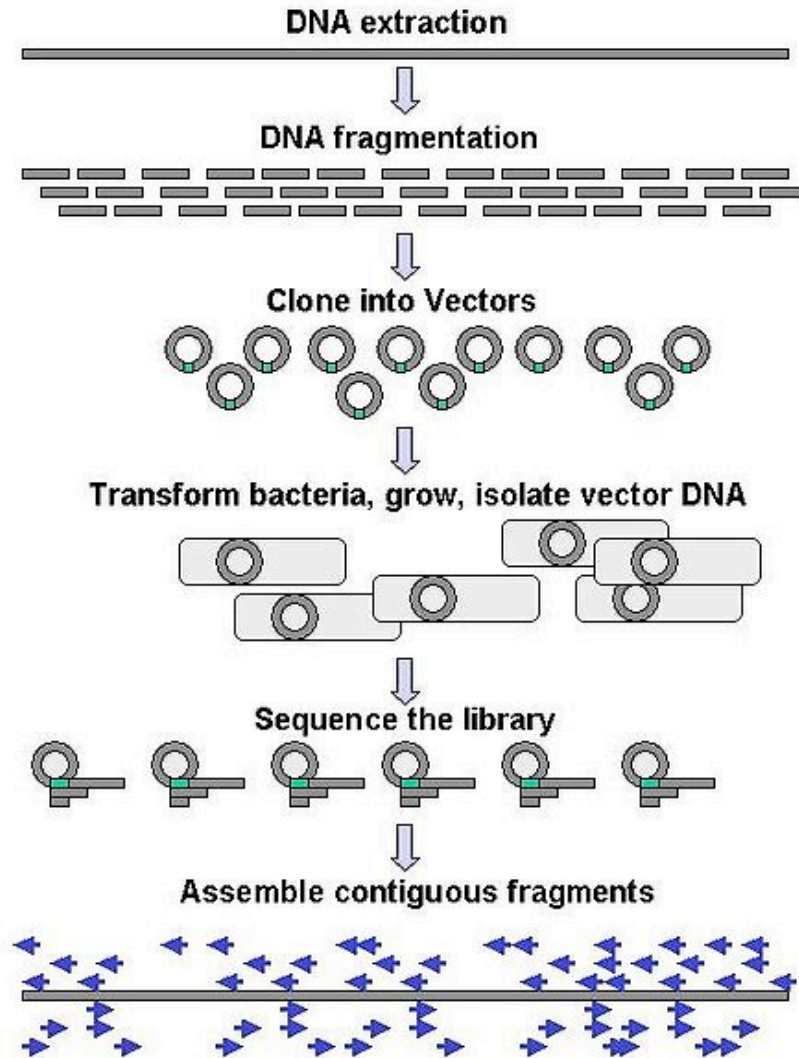
Amazon

The Biologist's Wishlist

- A complete and accurate set of all genes and their genomic positions
- A set of all the transcripts produced by each gene
- The location and timing of expression of each transcript
- The protein produced from each transcript
- The location and timing of each protein's expression
- The complete structure of each protein
- The functions of each protein

NGS technologies

The development of DNA sequencing



fragment

amplify

sequence

reconstruct

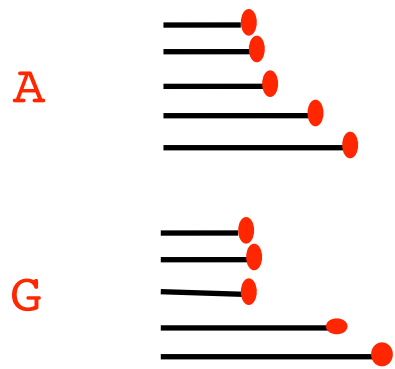
5' AACCGTCCGATC 3'
TTGGCAGGCTAG



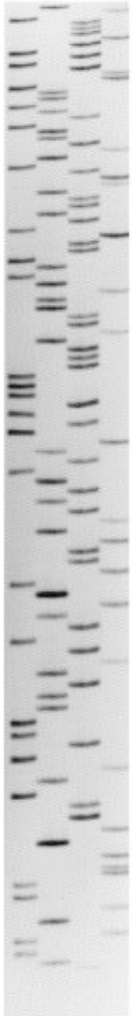
5' AACCGTCCGATC 3'

T
TA
TAG
TAGG
TAGGC
TAGGCA

A A T
T A G C
G C C

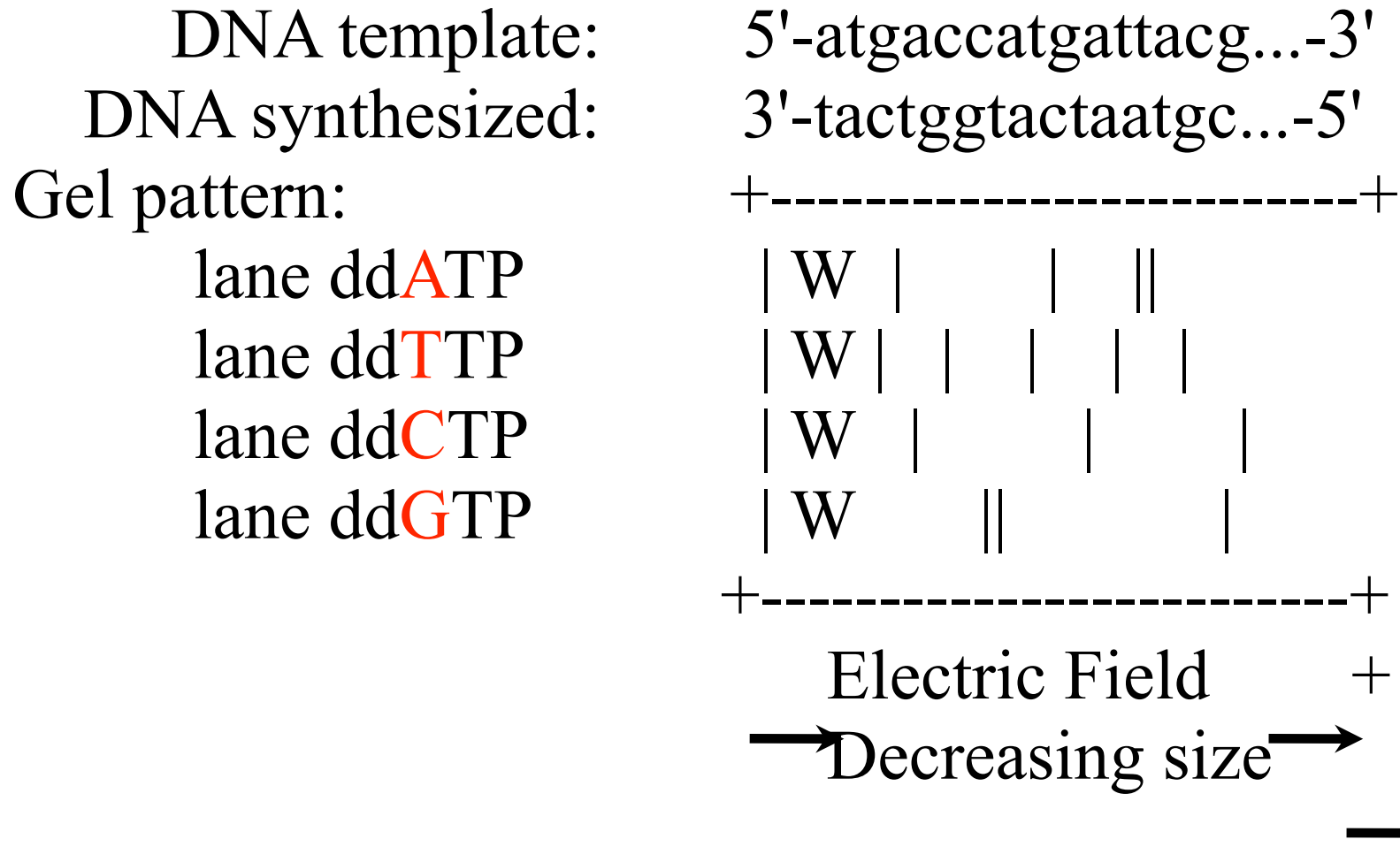


A sequencing gel autoradiogram

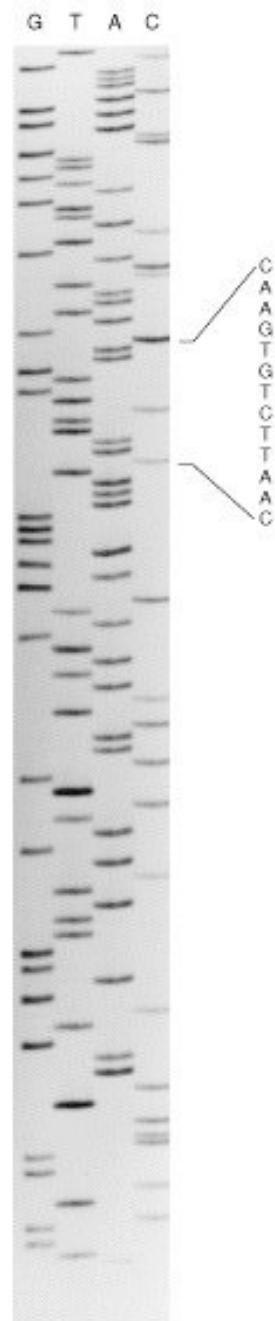


Autoradiograph. The dark color of the lines is proportional to the radioactivity from ^{32}P labeled adenonsine in the transcribed DNA sample.

Sanger chain termination sequencing

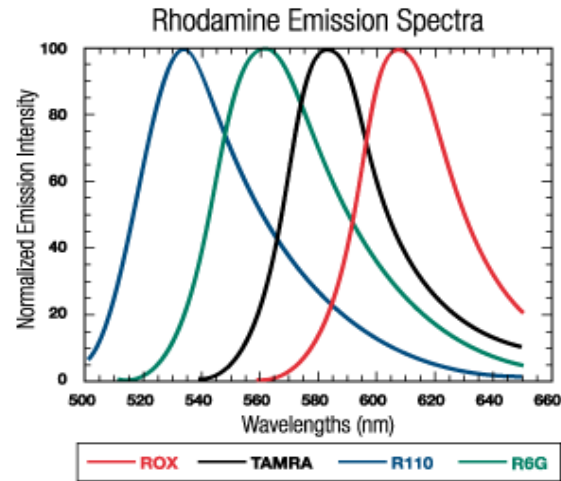


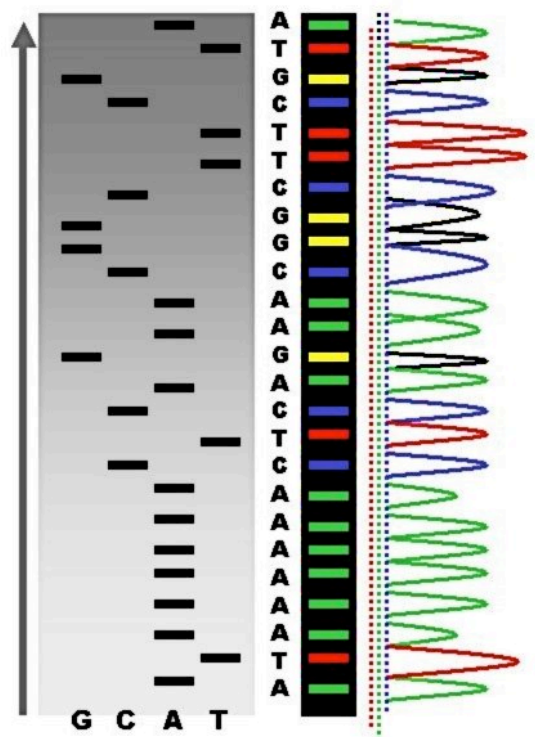
"W" indicates the well position, and "|" denotes the DNA bands on the sequencing gel.



Reading the sequencing ladder

Flourescent end labelling



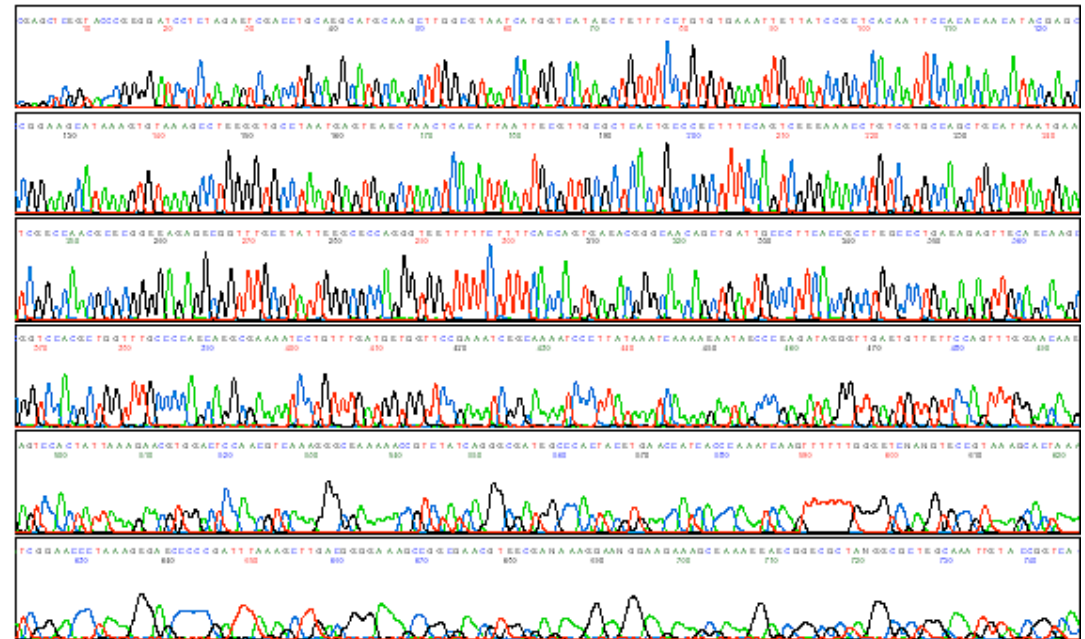
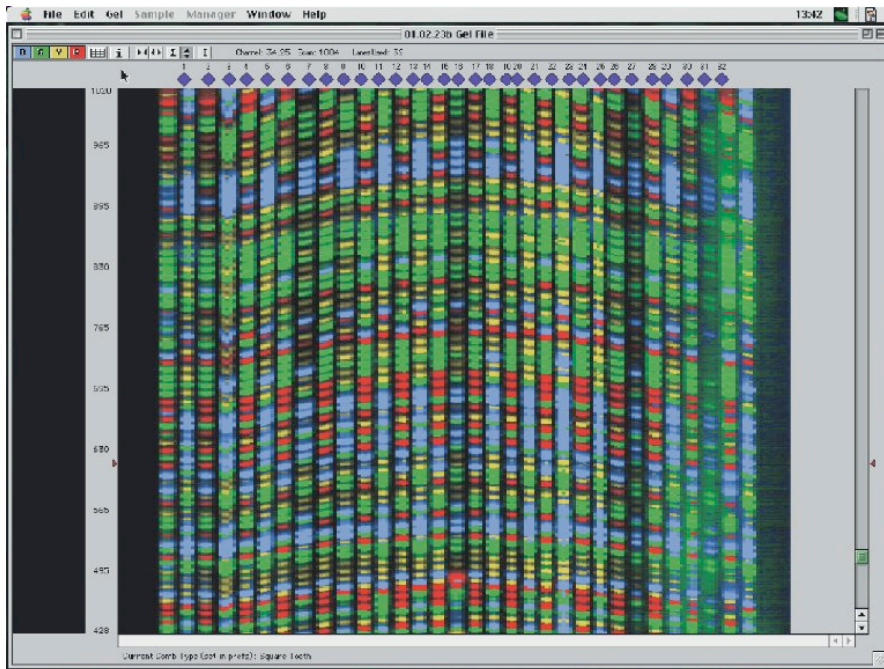


Dye Terminator sequencing

- Labelling of the chain terminator ddNTPs, which permits *sequencing in a single reaction*

Dye-terminator automated DNA sequencing

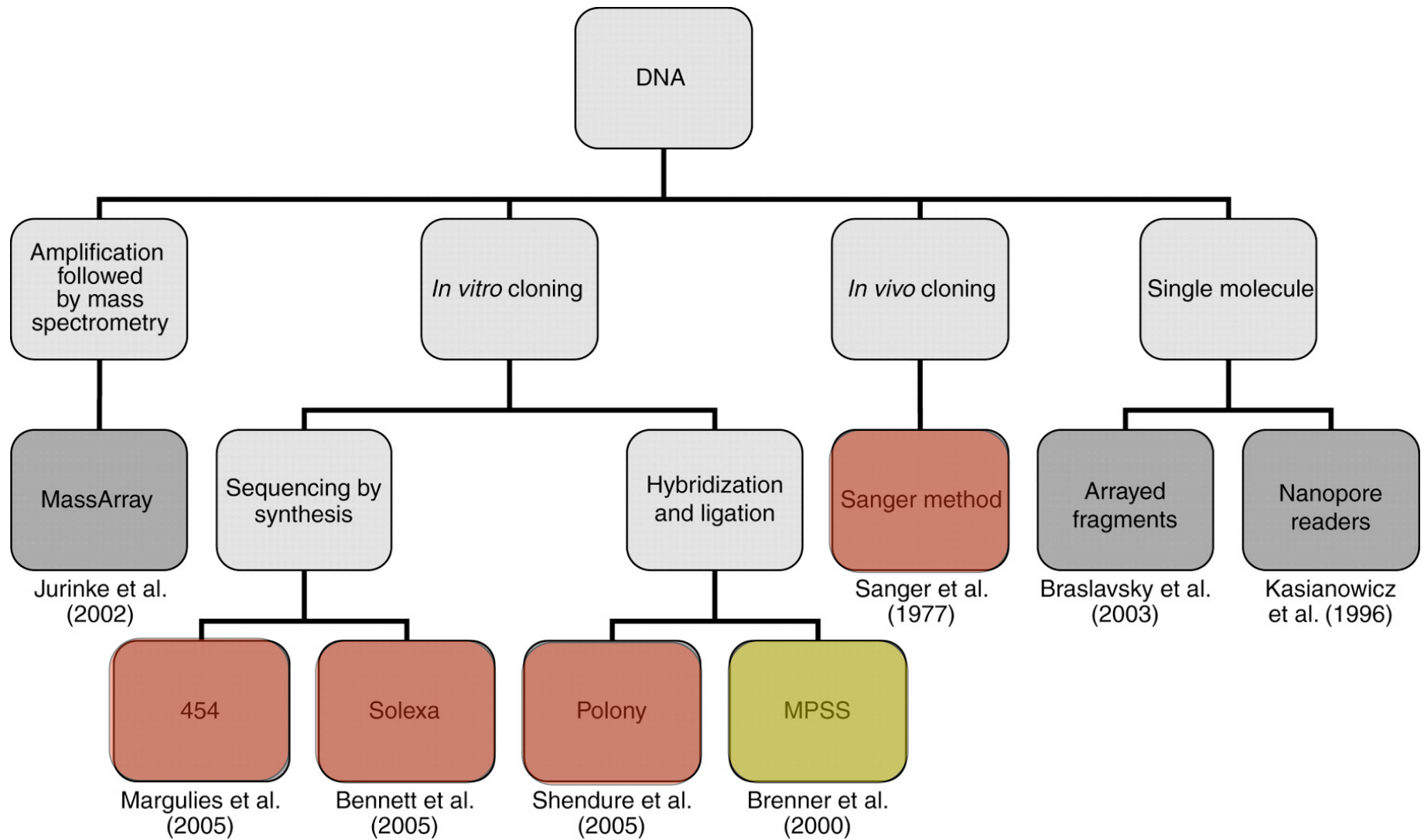
Chromatogram



A computer representation of the gel generates a “false color” image where each color corresponds to a base. The intensities are translated into peaks that represent the sequence.

Chemistry	Sanger Chain termination	Capillary	Dye terminator			
Read length	300		500-600			
Total/day	1-10kb		100kb- 1mB			
Samples	5-10		334			
Amplification	in vitro		in vitro			

An overview of current and emerging technologies for genomic sequencing.



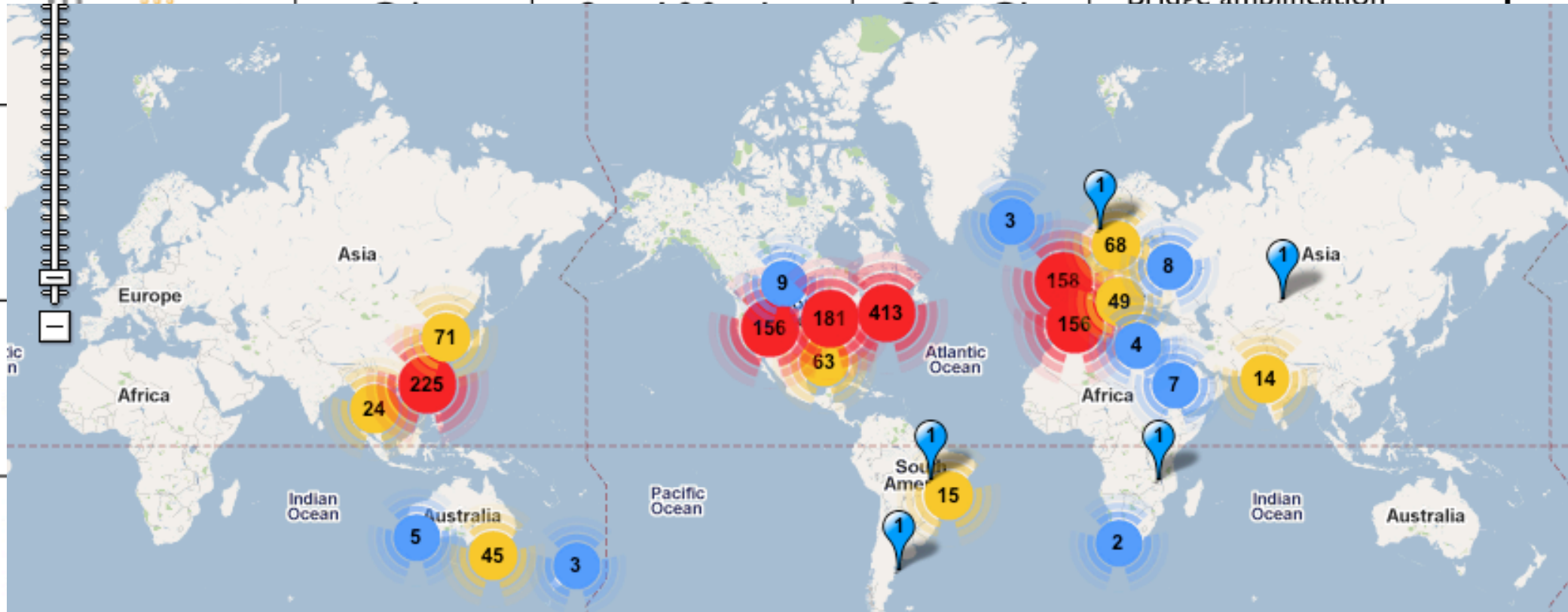
Hall N J Exp Biol 2007;210:1518-1525

Read Length

Gb/run

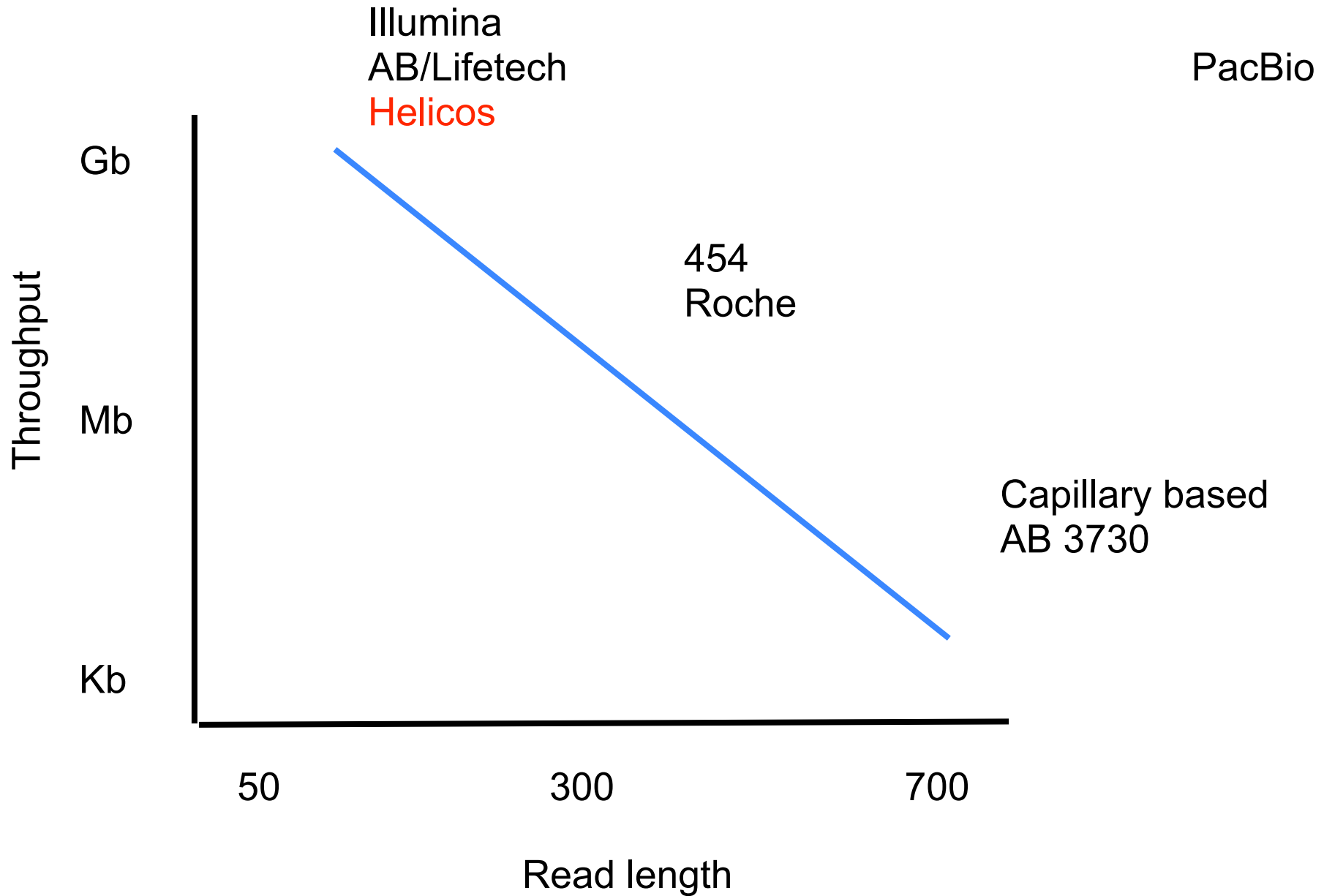
Technology

• Bridge amplification

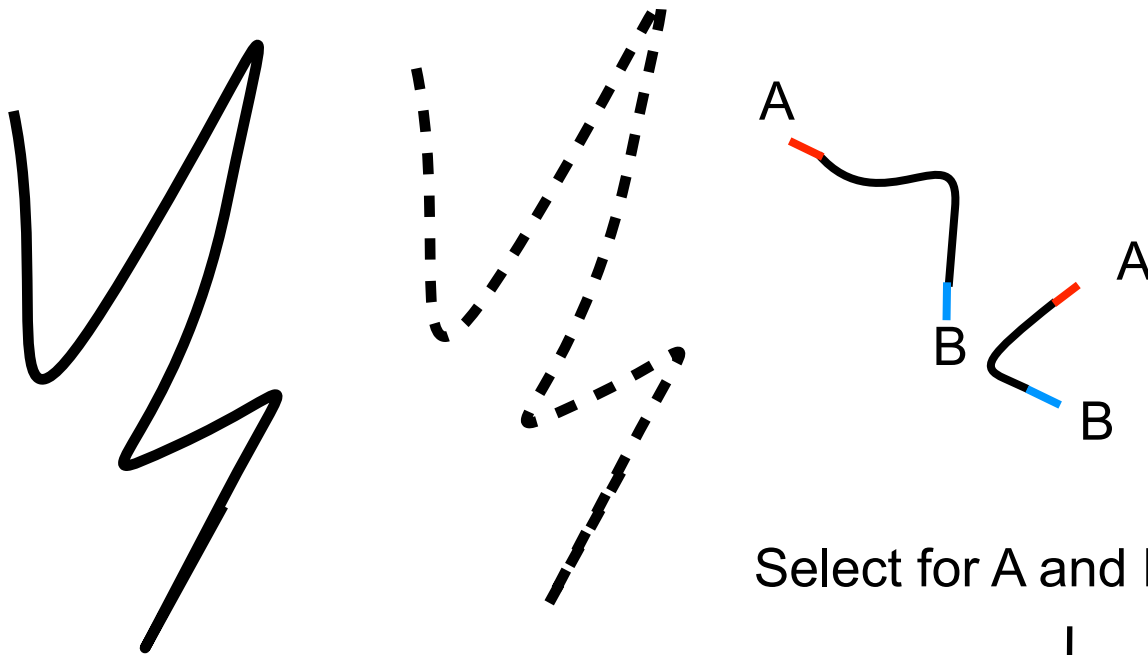


Sequencer

throughput = amt seq/unit time-cost



Template prep



Select for A and B adaptor fragments



Sequence



Attach to solid surface

Key NexGen attributes

- The library is not constructed by cloning
- Amplification is by a novel PCR
 - Fragments separated by
 - emulsion PCR
 - bridge PCR
- Very many fragments sequenced in parallel in a flow cell
- Imaged by microscope/CCD

Sequencing technologies — the next generation

Michael L. Metzker^{*‡}

Abstract | Demand has never been greater for revolutionary technologies that deliver fast, inexpensive and accurate genome information. This challenge has catalysed the development of next-generation sequencing (NGS) technologies. The inexpensive production of large volumes of sequence data is the primary advantage over conventional methods. Here, I present a technical review of template preparation, sequencing and imaging, genome alignment and assembly approaches, and recent advances in current

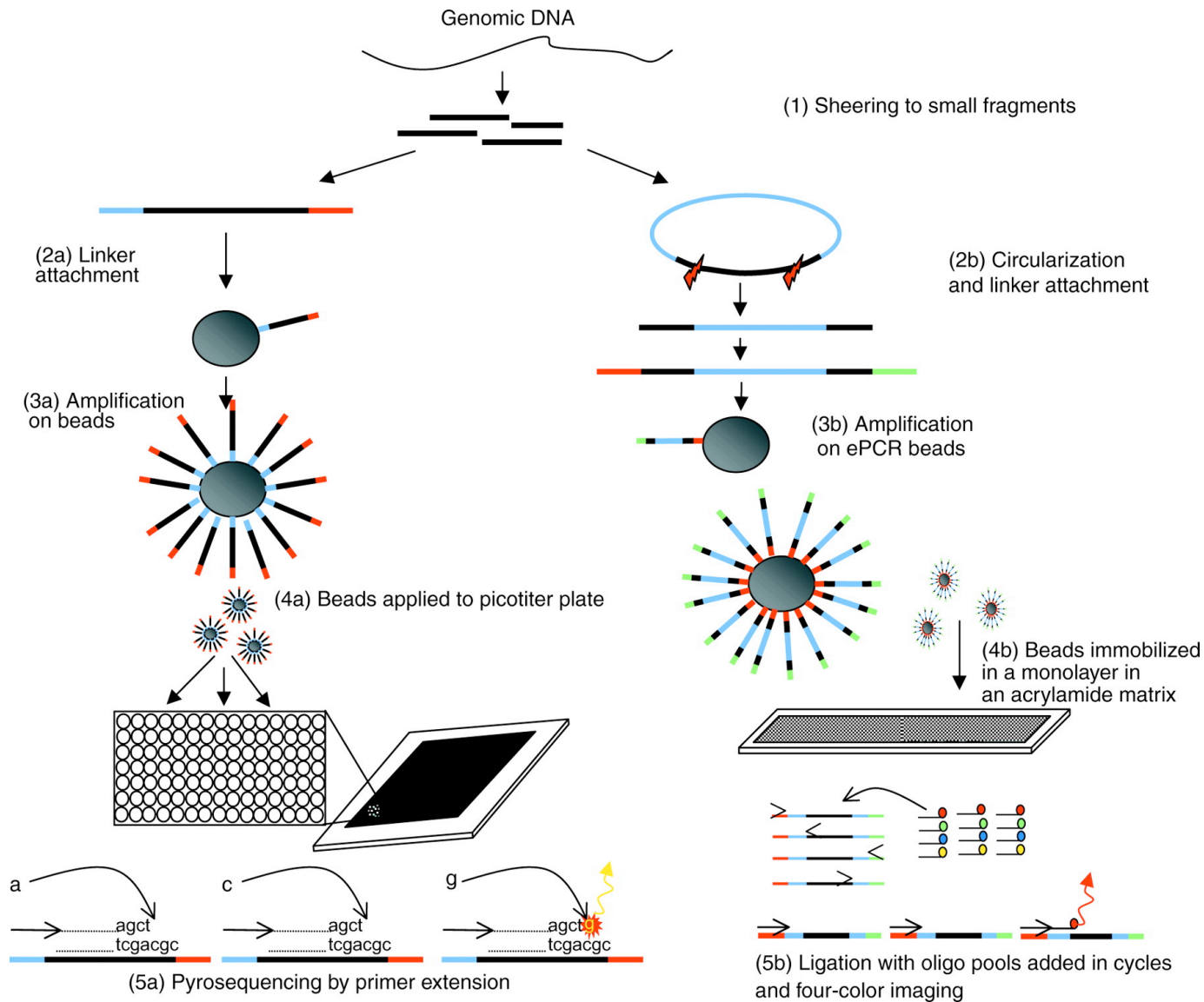
Amplification

- No single molecule available so...
- **Emulsion PCR** :isolated individual DNA molecules + primer-coated beads in aqueous droplets within an oil phase
- Polymerase chain reaction (PCR) coats each bead with clonal copies of the DNA molecule then an immobilization step for later sequencing

Polony sequencing

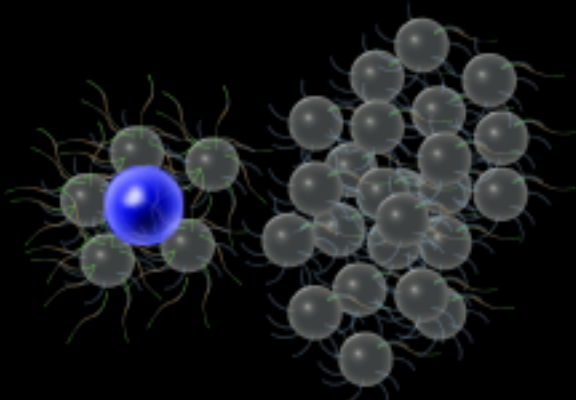
- Discrete clonal amplifications of a single DNA molecule, grown/immobilized in a gel matrix on a standard microscope slide

Outline of the 454 and polony sequencing process.

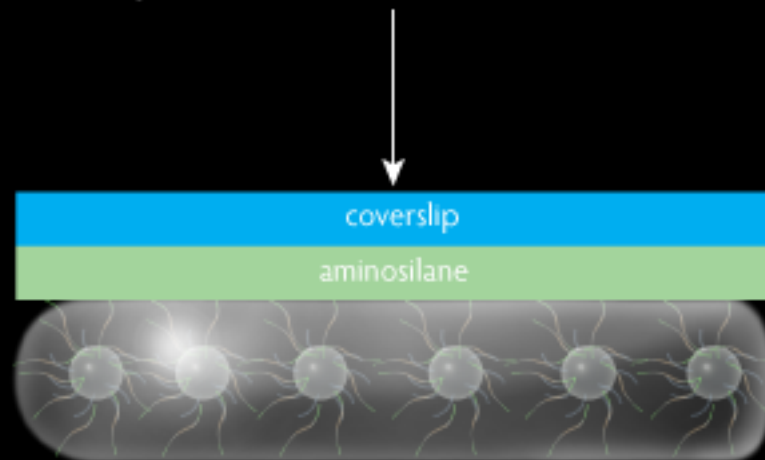


Hall N J Exp Biol 2007;210:1518-1525

The Journal of
**Experimental
Biology**



The beads are enriched by capture beads designed to bind to only those beads that have amplified DNA. The bead complex can then be isolated by centrifugation from the unamplified beads



The beads, binding to the aminosilane coating of the coverslip, spread out in a monolayer in an acrylamide gel, beneath which the sequencing reagents will flow



Degenerate nonamers are used for sequencing, each with a fluorophore attached. In the example here, T is detected at the 5th position. A computer program is then needed to assemble short reads from many beads

ARTICLES

Genome sequencing in microfabricated high-density picolitre reactors

Marcel Margulies^{1*}, Michael Egholm^{1*}, William E. Altman¹, Said Attiya¹, Joel S. Bader¹, Lisa A. Bemben¹, Jan Berka¹, Michael S. Braverman¹, Yi-Ju Chen¹, Zhoutao Chen¹, Scott B. Dewell¹, Lei Du¹, Joseph M. Fierro¹, Xavier V. Gomes¹, Brian C. Godwin¹, Wen He¹, Scott Helgesen¹, Chun He Ho¹, Gerard P. Irzyk¹, Szilveszter C. Jando¹, Maria L. I. Alenquer¹, Thomas P. Jarvie¹, Kshama B. Jirage¹, Jong-Bum Kim¹, James R. Knight¹, Janna R. Lanza¹, John H. Leamon¹, Steven M. Lefkowitz¹, Ming Lei¹, Jing Li¹, Kenton L. Lohman¹, Hong Lu¹, Vinod B. Makhijani¹, Keith E. McDade¹, Michael P. McKenna¹, Eugene W. Myers², Elizabeth Nickerson¹, John R. Nobile¹, Ramona Plant¹, Bernard P. Puc¹, Michael T. Ronan¹, George T. Roth¹, Gary J. Sarkis¹, Jan Fredrik Simons¹, John W. Simpson¹, Maithreyan Srinivasan¹, Karrie R. Tartaro¹, Alexander Tomasz³, Kari A. Vogt¹, Greg A. Volkmer¹, Shally H. Wang¹, Yong Wang¹, Michael P. Weiner⁴, Pengguang Yu¹, Richard F. Beigley¹ & Jonathan M. Rothberg¹

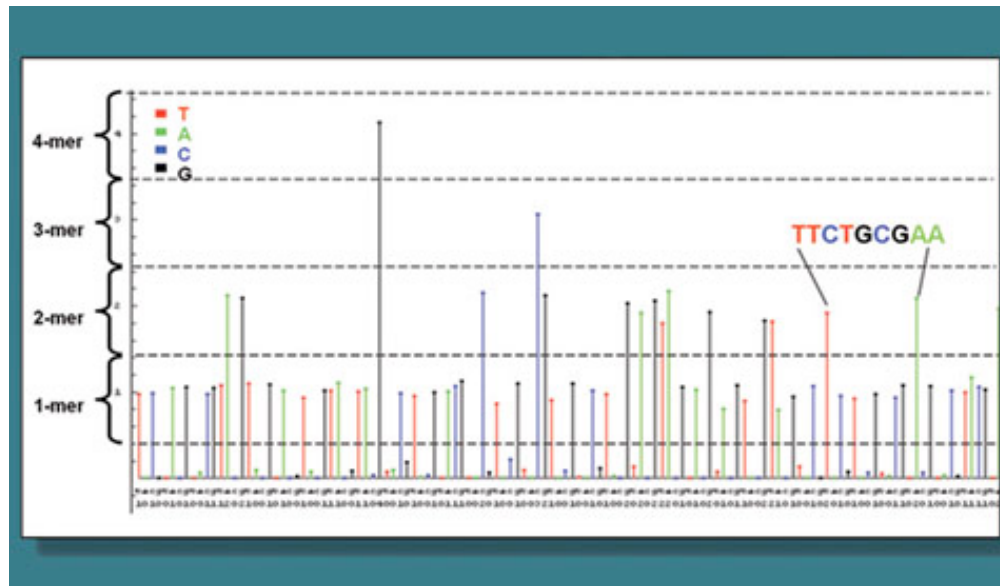
The proliferation of large-scale DNA-sequencing projects in recent years has driven a search for alternative methods to reduce time and cost. Here we describe a scalable, highly parallel sequencing system with raw throughput significantly greater than that of state-of-the-art capillary electrophoresis instruments. The apparatus uses a novel fibre-optic slide of individual wells and is able to sequence 25 million bases, at 99% or better accuracy, in one four-hour run. To achieve an approximately 100-fold increase in throughput over current Sanger sequencing technology, we have developed an emulsion method for DNA amplification and an instrument for sequencing by synthesis using a pyrosequencing protocol optimized for solid support and picolitre-scale volumes. Here we show the utility, throughput, accuracy and robustness of this system by shotgun sequencing and *de novo* assembly of the *Mycoplasma genitalium* genome with 96% coverage at 99.96% accuracy in one run of the machine.

DNA sequencing has markedly changed the nature of biomedical research and medicine. Reductions in the cost, complexity and time higher than that of Sanger sequencing by capillary electrophoresis, it is currently at the cost of substantially shorter reads and lower



454
SEQUENCING

Pyrosequencing flowgram



- presented 2005, first on market
- emulsion PCR
- pyrosequencing (polymerase-based)
- read length: 250 bp
- paired read separation: 3 kb
- 300 Mb per day
- \$60 per Mb
- error rate: around 5% per bp
- dominant type of error: indels, especially in homopolymers

Applications

RESEARCH ARTICLE

Sequencing and Neanderthal Ge

James P. Noonan,^{1,2} Graham Coop,³ S
Johannes Krause,⁴ Joe Alessi,¹ Feng C
Jonathan K. Pritchard,³ Edward M. Ru

Our knowledge of Neanderthals is based on a few fragments of DNA. To describe the characterization of these e development of a Neanderthal metagen analysis. Several lines of evidence indicate that the sequence identities between Neanderthal and modern human sequence is different. These results enable us to estimate the time based on multiple randomly distributed mutations in the Neanderthal genomic sequence we compared to a most recent common ancestor ~706,000 years ago. Ancestral populations split ~370,000 years ago.

nature

Vol 452 | 17 April 2008 | doi:10.1038/nature06884

LETTERS

The complete genome of an individual by massively parallel DNA sequencing

David A. Wheeler^{1*}, Maithreyan Srinivasan^{2*}, Michael Egholm^{2*}, Yufeng Shen^{1*}, Lei Chen¹, Amy McGuire³, Wen He², Yi-Ju Chen², Vinod Makhijani², G. Thomas Roth², Xavier Gomes², Karrie Tartaro^{2†}, Faheem Niazi², Cynthia L. Turcotte², Gerard P. Irzyk², James R. Lupski^{1,5,6}, Craig Chinault⁴, Xing-zhi Song¹, Yue Liu¹, Ye Yuan¹, Lynne Nazareth¹, Xiang Qin¹, Donna M. Muzny¹, Marcel Margulies², George M. Weinstock^{1,4}, Richard A. Gibbs^{1,4} & Jonathan M. Rothberg^{2†}

The association of genetic variation with disease and drug response, and improvements in nucleic acid technologies, have given great optimism for the impact of 'genomic medicine'. However, the formidable size of the diploid human genome¹, approximately 6 gigabases, has prevented the routine application of sequencing methods to deciphering complete individual human genomes. To realize the full potential of genomics for human health, this limitation must be overcome. Here we report the DNA sequence of a diploid genome of a single individual, James D. Watson, sequenced to 7.4-fold redundancy in two months using massively parallel sequencing in picolitre-size reaction vessels. This sequence was completed in two months at approximately one-hundredth the cost of traditional capillary electrophoresis methods. Comparison of the sequence to the reference genome led to the identification of 3.3 million single nucleotide polymorphisms, of which 10,654 were amino acid substitution within

subject's DNA, including single nucleotide polymorphisms (SNPs), small insertions and deletions (indels), and copy number variation (CNV).

The 454 base-calling software provides error estimates (Q values) for each base. We developed a three-step filtering process using the patterns of error and associated Q values from the 454 base-calling software to improve the accuracy of SNP discovery. An initial 14 million variant positions were filtered to 3.32 million putative SNPs (Table 1).

Comparison of these putative SNPs in the subject's genome with those in the dbSNP (dbSNP: <http://www.ncbi.nlm.nih.gov/projects/SNP/>) revealed 2.72 million in common ('known SNPs'). Approximately 99% of SNPs in dbSNP are bi-allelic. At only 10,425 positions did the subject's variant not match the variant found in dbSNP. Although some of these could represent a third allele in the population, or an error in the dbSNP nomenclature record, we

James Watson's Personal Genome Sequence



README: [How do I use the James Watson Genome Browser?](#)

Downloads: Download [bulk JW polymorphisms](#). For the complete data set, please go to the [NCBI Trace Archive](#) and search for `CENTER_NAME = 'CSHL'` and `CENTER_PROJECT = 'Project Jim'`.

Showing 34.46 kbp from chr7, positions 75,221,807 to 75,256,264

Instructions

Search using a sequence name, gene name, locus, or other landmark. The wildcard character * is allowed. To center on a location, click the ruler. Use the Scroll/Zoom buttons to change magnification and position.

Examples: [HTR2A](#), [macular degeneration](#), [rs726455](#), [DAOA](#), [chr22:20230140..20330139](#), [PARK3](#), [SNP:rs131693](#), [SPTB](#), [NM_001008496](#), [3q21.2](#), [ENM010](#).

[\[Hide banner\]](#) [\[Bookmark this\]](#) [\[Link to Image\]](#) [\[High-res Image\]](#) [\[Help\]](#) [\[Reset\]](#)

Search

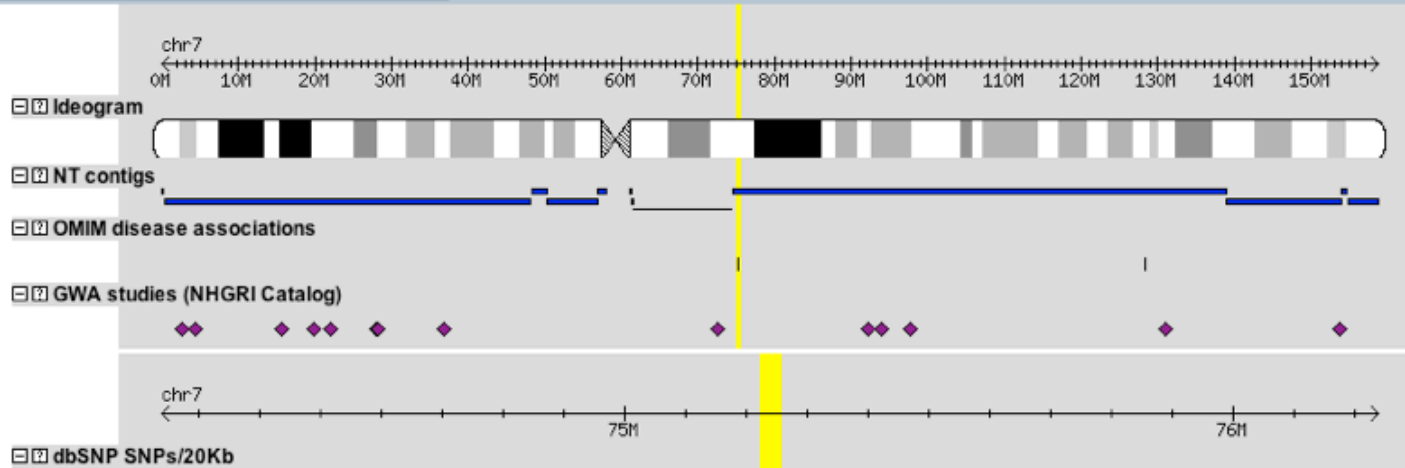
Landmark or Region:

Data Source

Reports & Analysis:

Scroll/Zoom: Flip

Overview

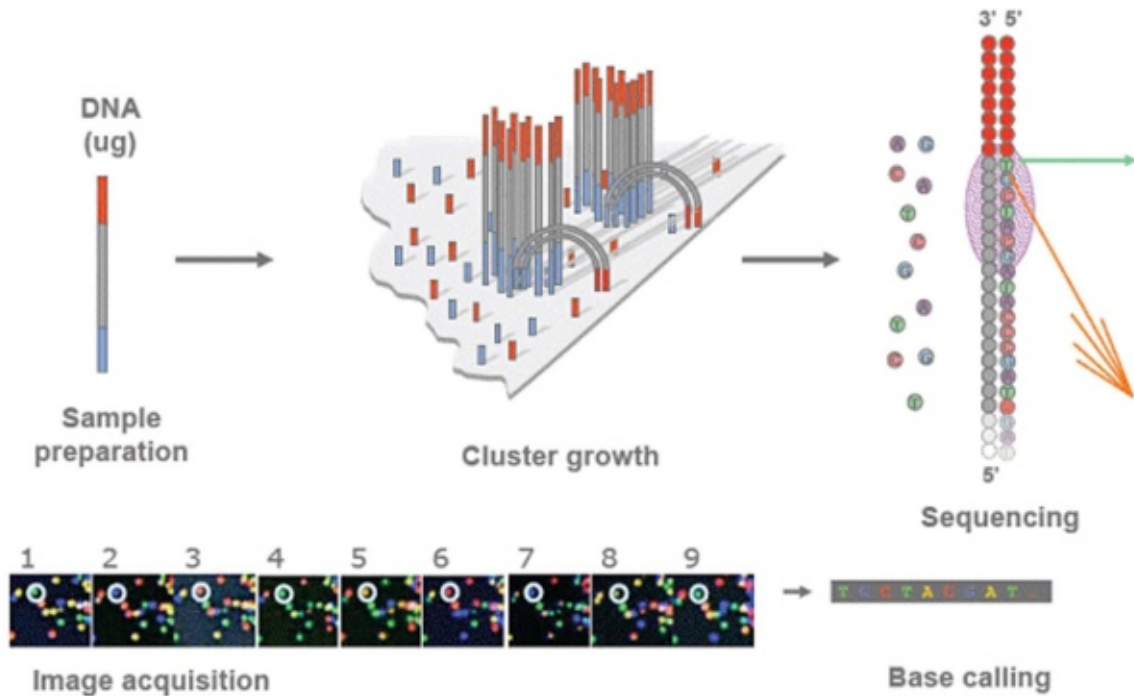


Illumina (Solexa)

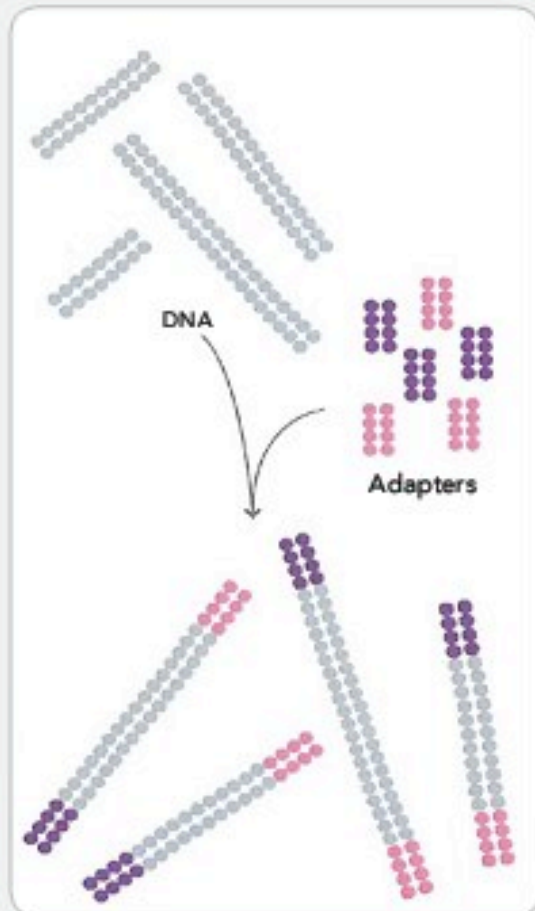
Systems / Genome Analyzer IIX



Illumina Sequencing Technology

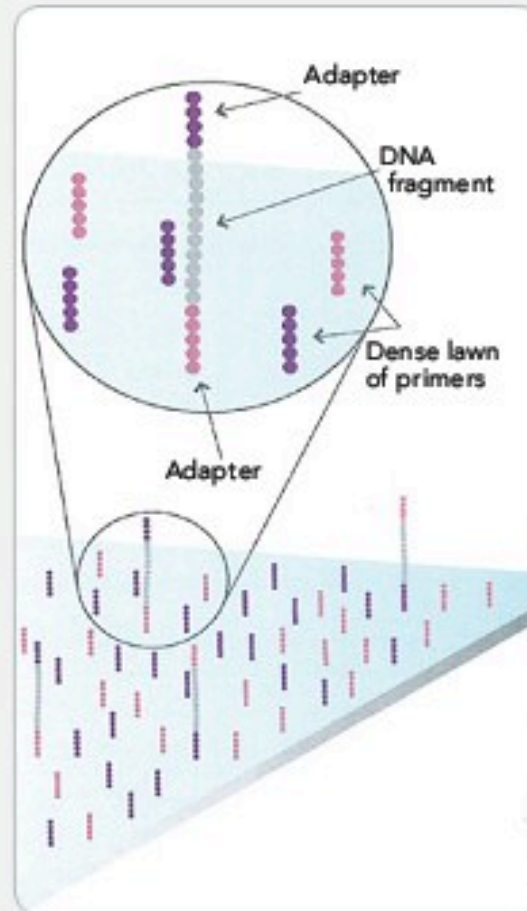


1. PREPARE GENOMIC DNA SAMPLE



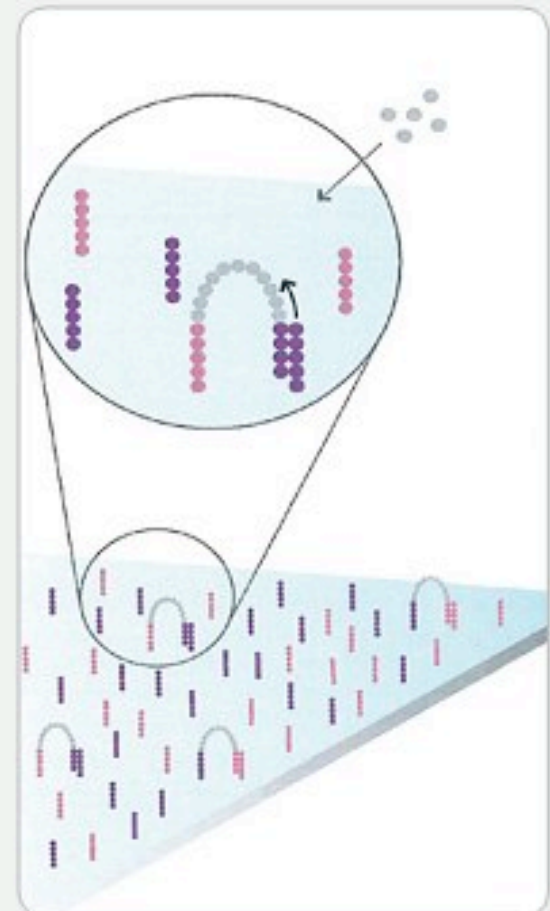
Randomly fragment genomic DNA and ligate adapters to both ends of the fragments.

2. ATTACH DNA TO SURFACE



Bind single-stranded fragments randomly to the inside surface of the flow cell channels.

3. BRIDGE AMPLIFICATION



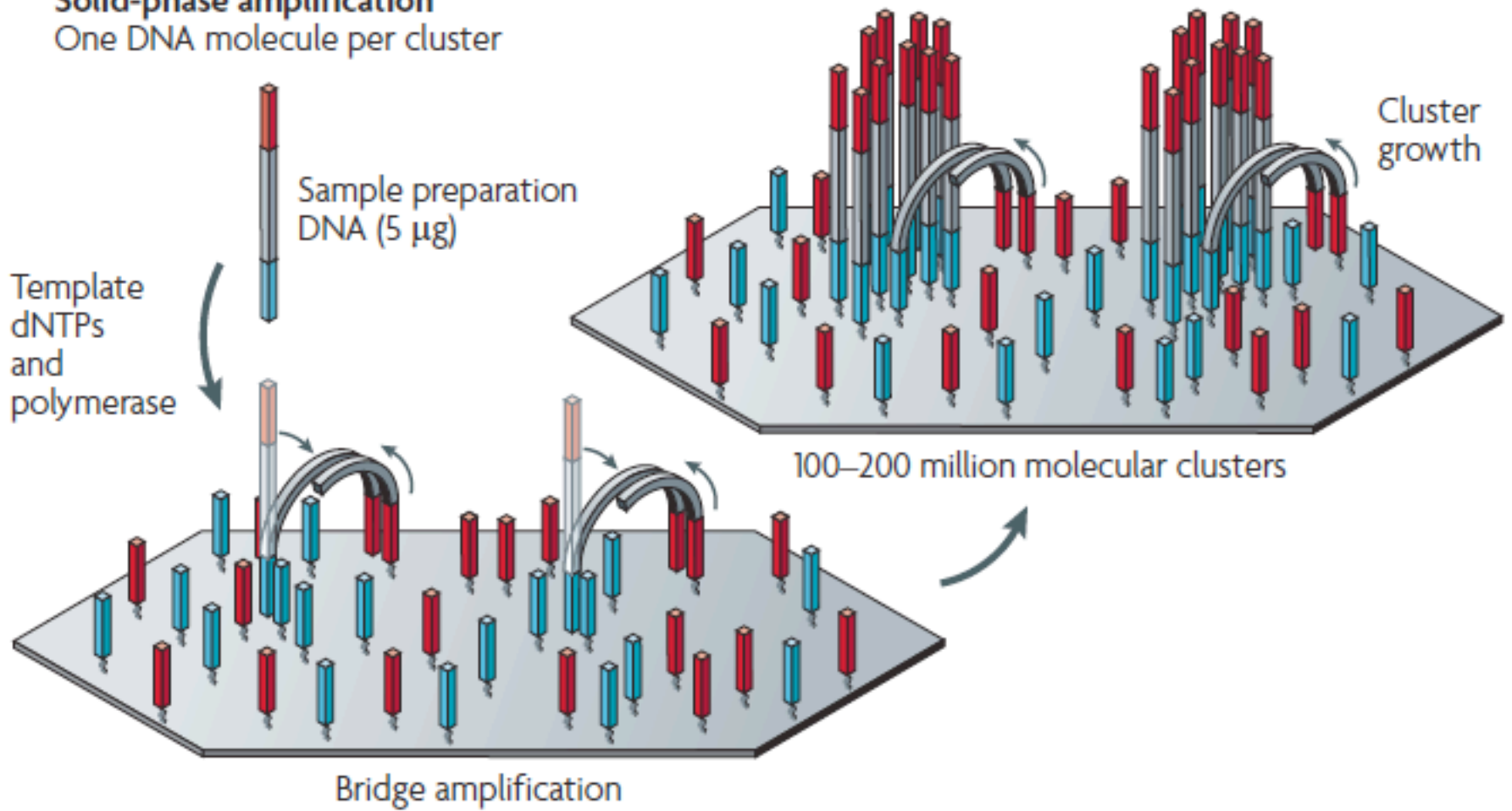
Add unlabeled nucleotides and enzyme to initiate solid-phase bridge amplification.

4. FRAGMENTS BECOME DOUBLE STRANDED

5. DENATURE THE DOUBLE-STRANDED MOLECULES

6. COMPLETE AMPLIFICATION

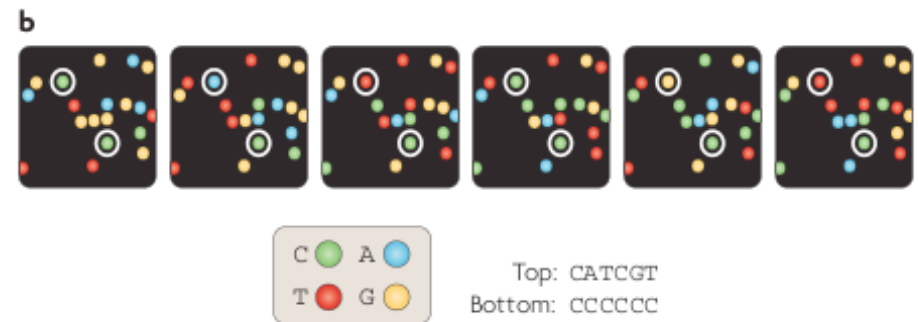
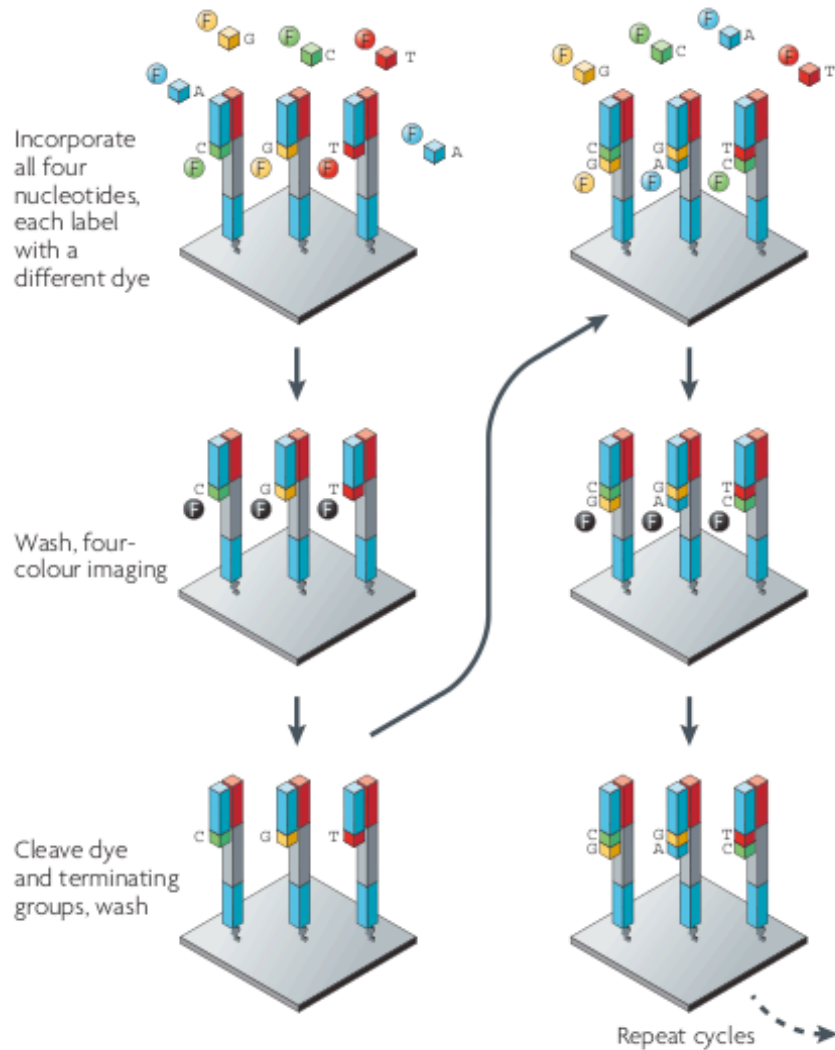
b Illumina/Solexa
Solid-phase amplification
One DNA molecule per cluster



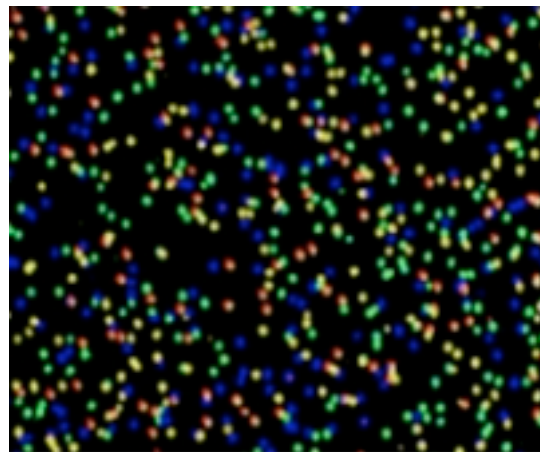
Single base extension - sequencing

Reversible terminator

a Illumina/Solexa — Reversible terminators



- massive parallelism
- 8 lane flow cells (microscope slides)
- ‘glorified PCR machine’ - cluster amplification
- 960X 4 images per cycle



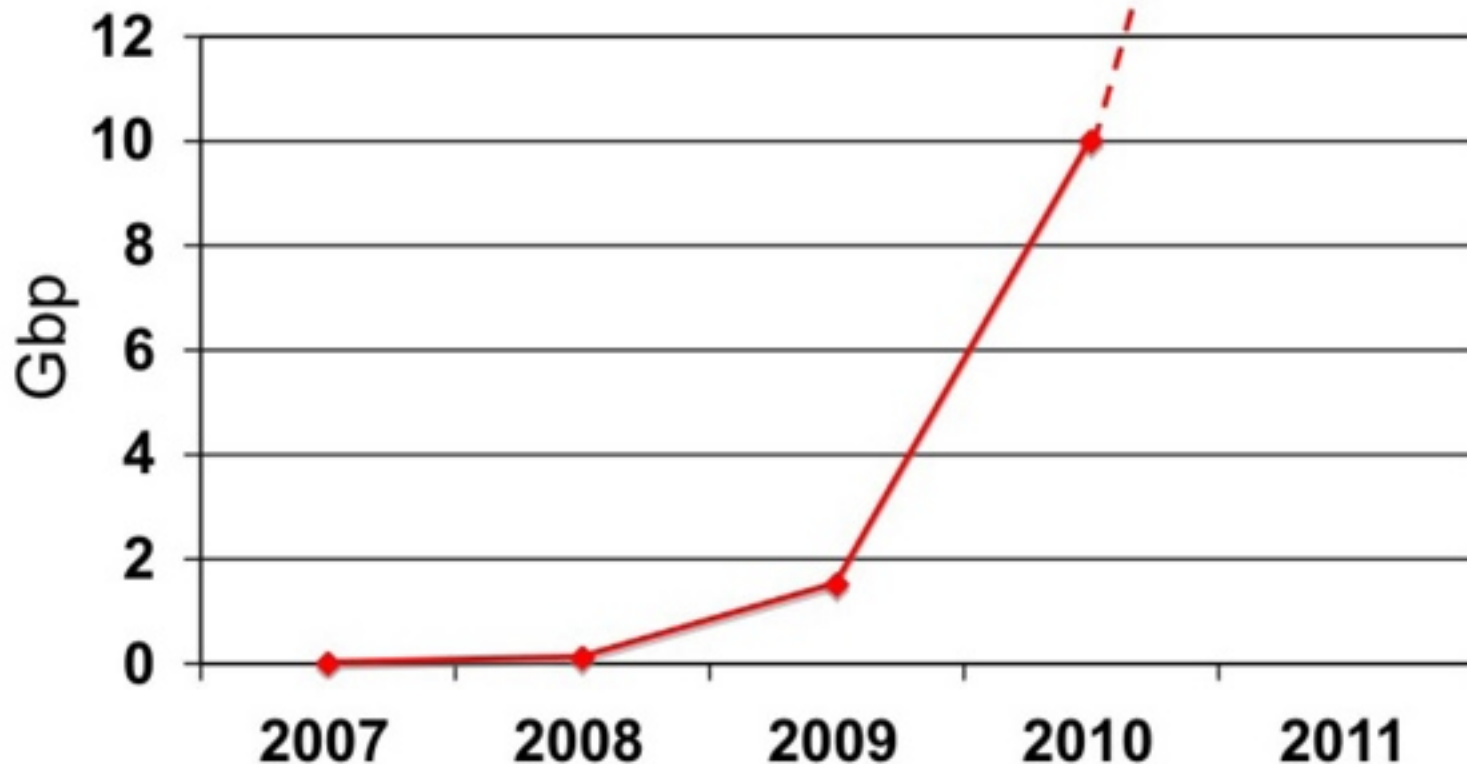
Stats

- second on the market
- bridge PCR
- polymerase-based sequencing-by-synthesis
- 32..40 bp (newest models: up to 100 bp)
- paired read separation: 200 bp - can use to generate longer reads
- 400 Mb per day (getting better)
- \$2 per Mb
- error rate: 1% per bp (good reads: 0.1%)
- dominant error type: substitutions

Hiseq N 000

- improved optics/imaging at more density
- 2 flow cells 8 day runtime

Exponential growth of Illumina mapped sequence / lane throughput



	Jan-07	Jan-08	Jan-09	Jan-10	Jan-11
◆ Gbp / lane	0.025	0.128	1.5	10	30-50 ?

Read type: 1x25 1x36 2x75 2x100 2x150

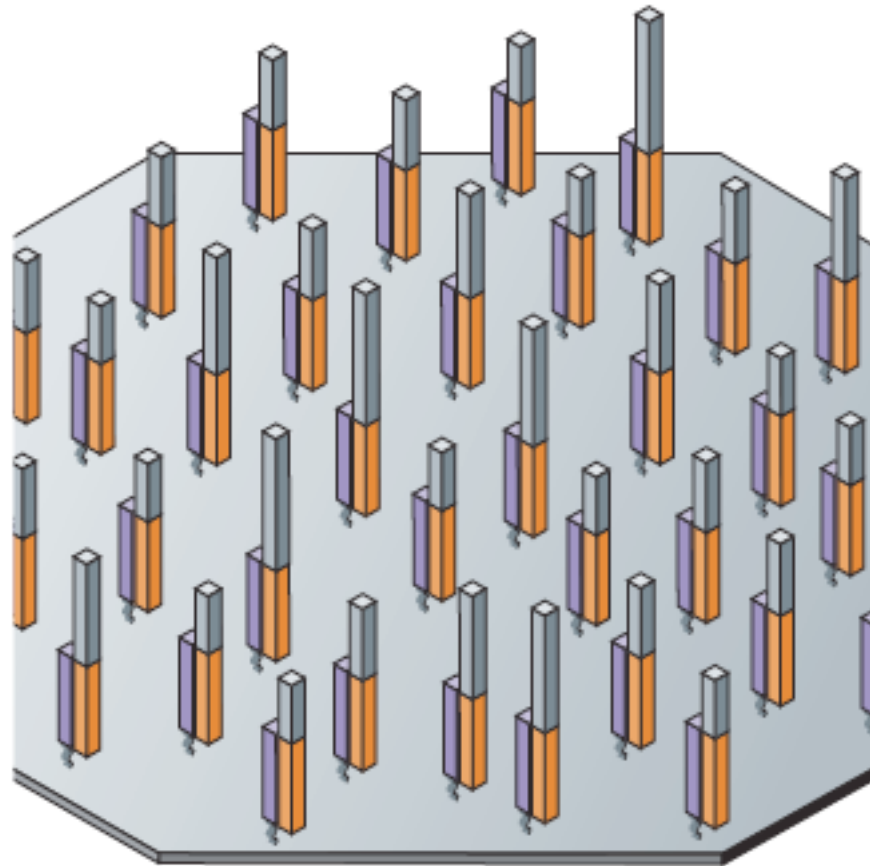
Cost/lane is relatively stable at \$600 to \$1,200

Single-molecule

- Bright fluorophores and laser excitation to detect base addition events from individual DNA molecules fixed to a surface (Helicos)

Immobilization by a primer

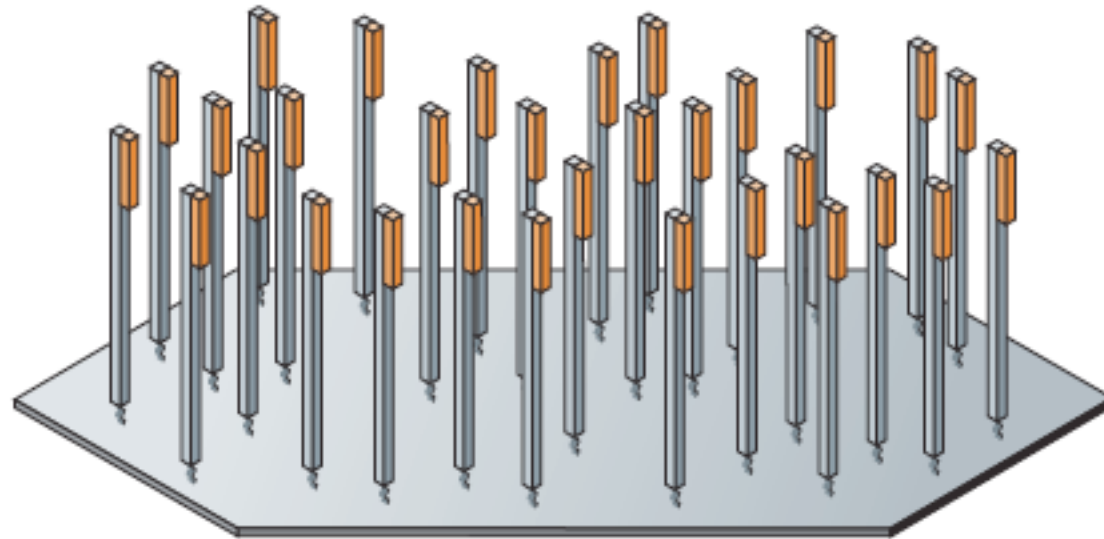
c Helicos BioSciences: one-pass sequencing
Single molecule: primer immobilized



Billions of primed, single-molecule templates

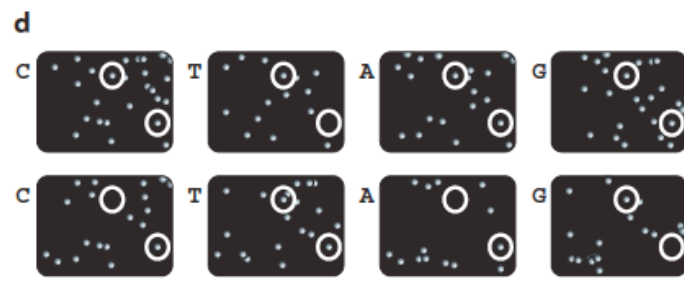
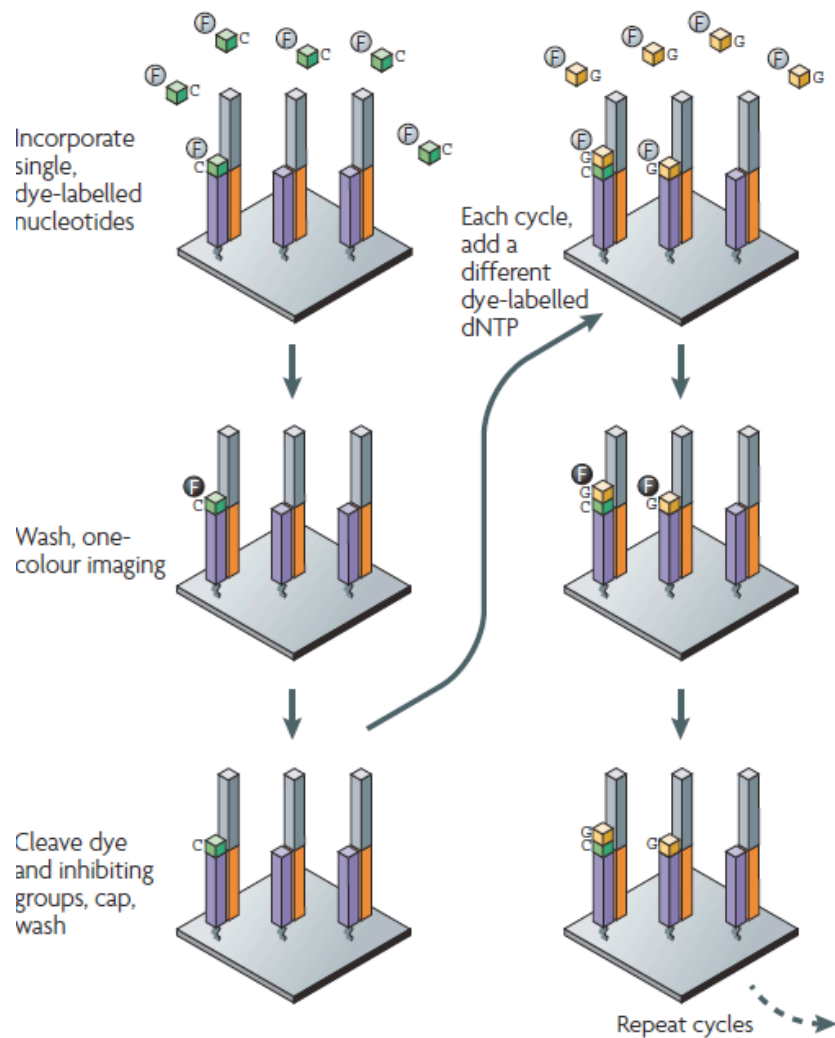
Immobilization by a template

d Helicos BioSciences: two-pass sequencing
Single molecule: template immobilized



Billions of primed, single-molecule templates

c Helicos BioSciences — Reversible terminators

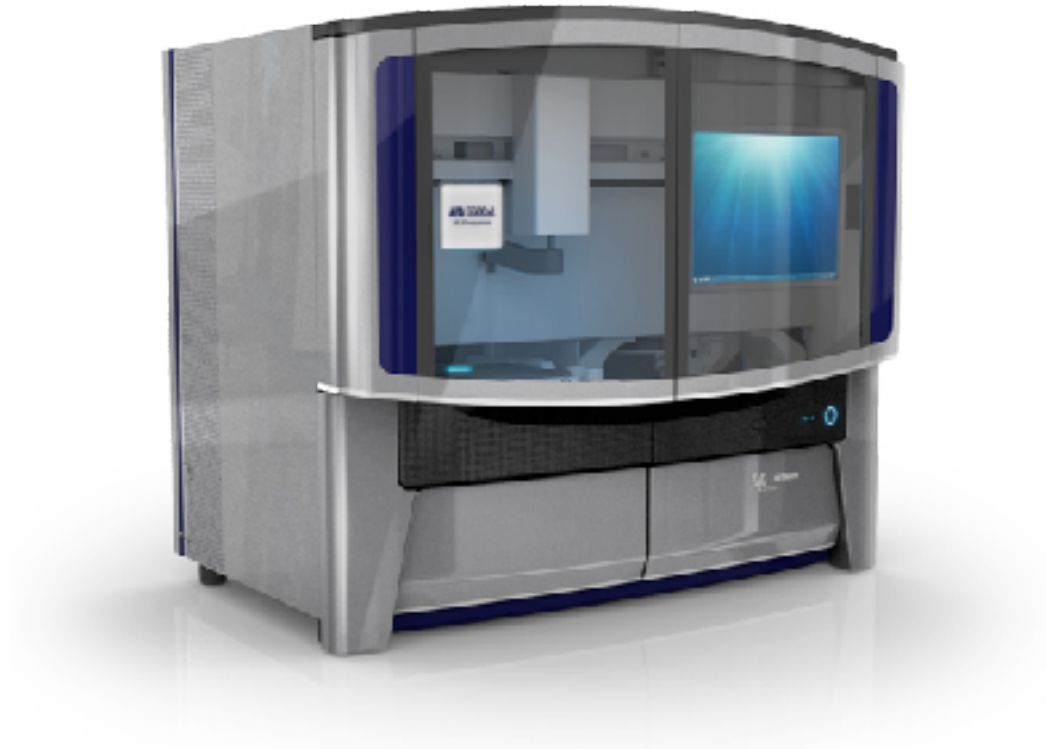


Top: CTAGTG
 Bottom: CAGCTA

Helicos

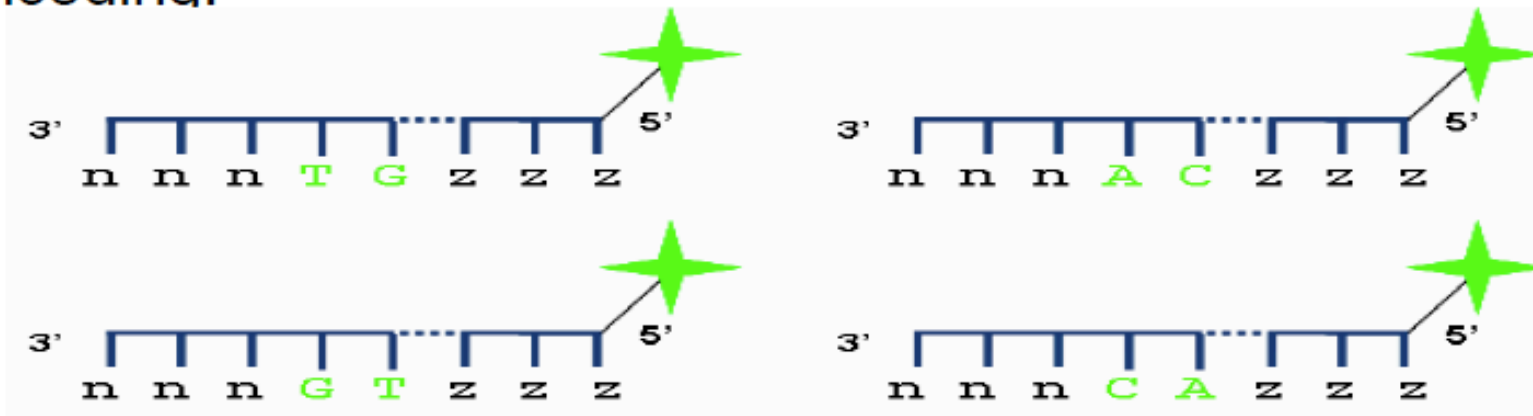
- on the market for a year
- no amplification
- single-molecule polymerase-based sequencing
- read length: 25..45 bp
- 1200 Mb per day
- \$1 per Mb
- error rate: <1% (manufacturer claim)

SOLiD ABI (Life Technologies)



Ligation sequencing

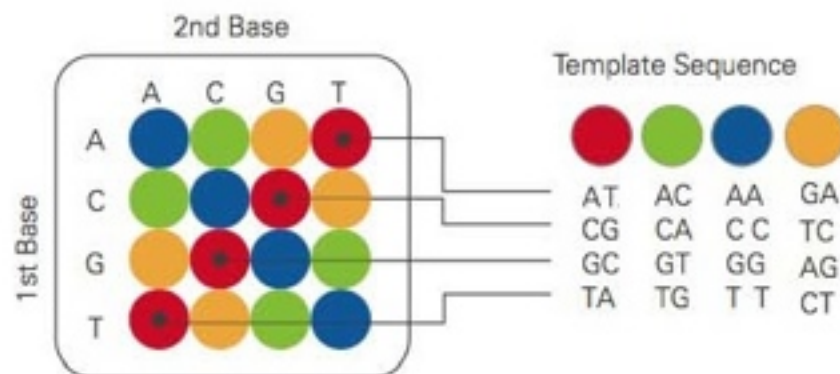
The SOLiD Sequencing System uses probes with dual base encoding.



SOLiD Sequencing - Color Space

- ▶ 4 fluorescent dyes for 16 possible 2-mers
- ▶ Reverse, complement and reverse complement are always of same color

Possible Dinucleotides Encoded By Each Color



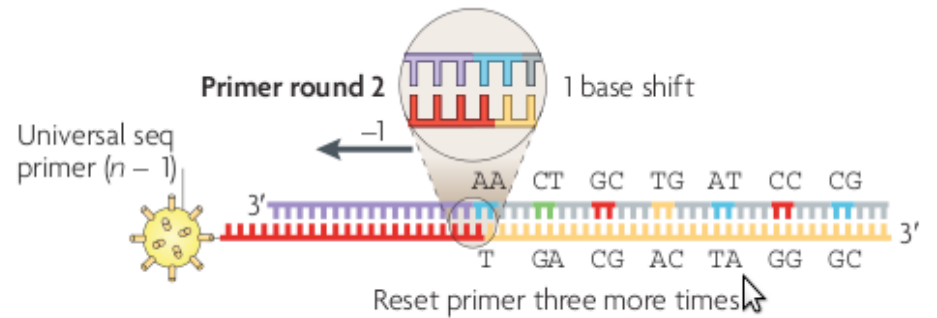
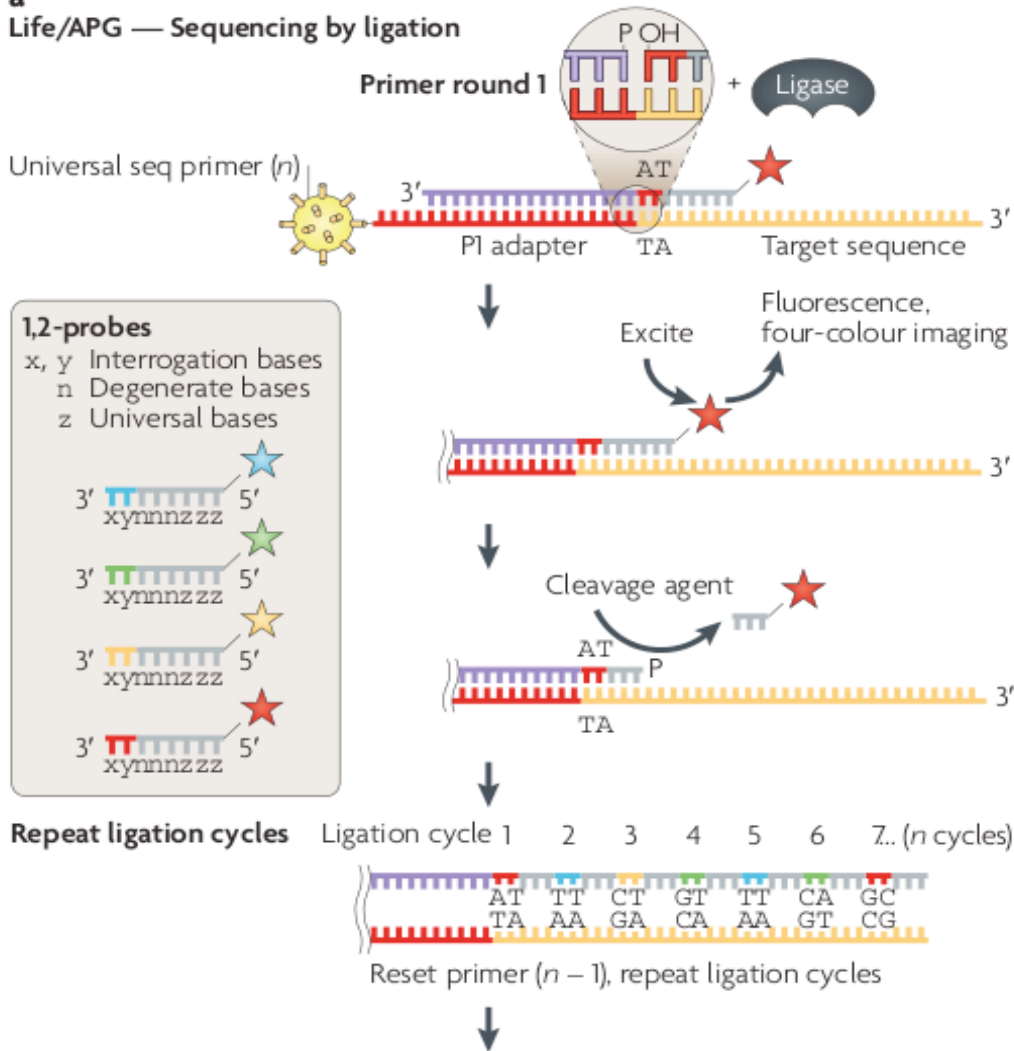
Double Interrogation

With 2 base encoding each base is defined twice



SOLiD

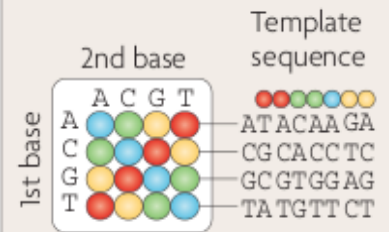
a Life/APG — Sequencing by ligation



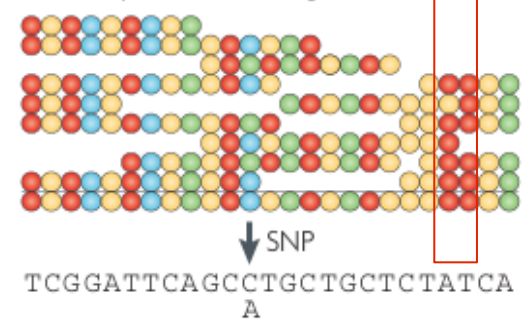
errors do not have adjacent compensatory changes

b

Two-base encoding: each target nucleotide is interrogated twice



Alignment of colour-space reads to colour-space reference genome



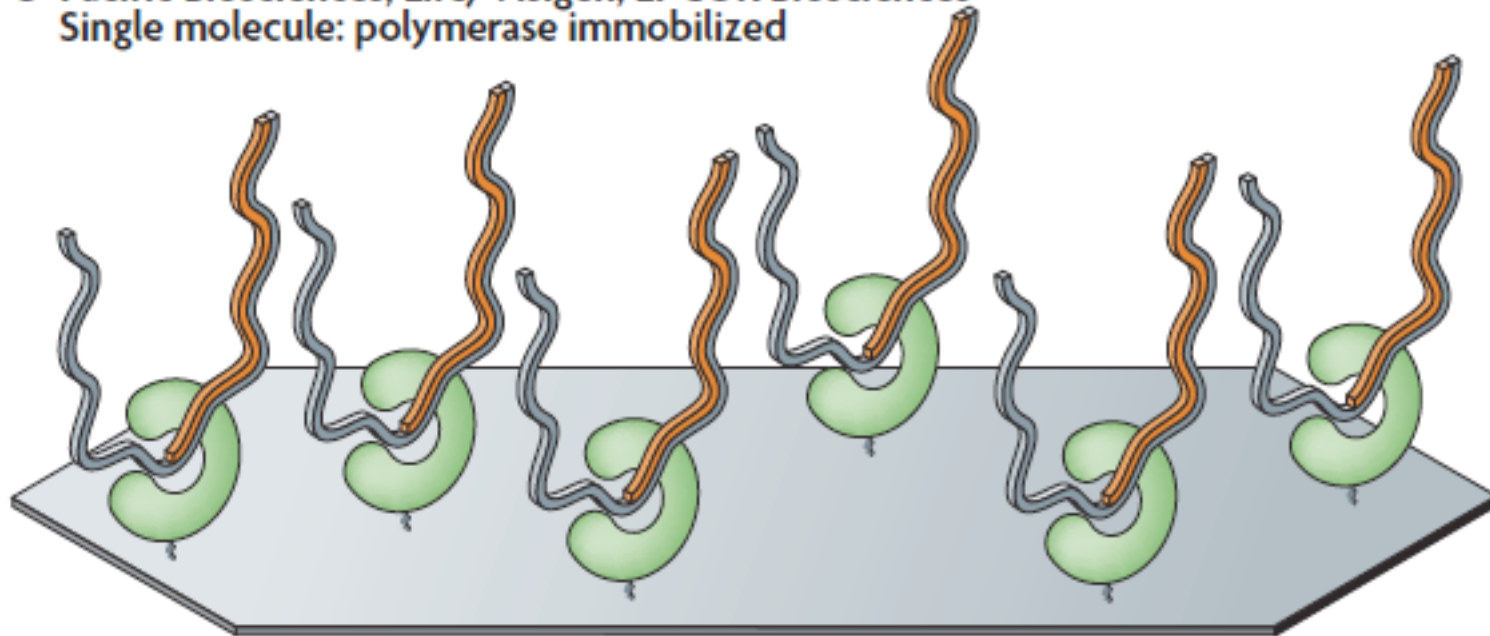
SNP requires adjacent valid colour change

- known pairs of bases within an 8mer
- read colours that represent 2 bases at a time
- decode from colour space - colour + identity of first base
- variant detection is colour change event specific

- third on market (since late 2007)
- emulsion PCR
- ligase-based sequencing
- read length: 50bp
- paired read separation: 3 kb
- 600 Mb per day (colour space)
- \$1 per Mb
- very low error rate: <0.1% per bp (still high compared to Sanger capillary sequencing: 0.001%)
- dominant error type: substitutions (colour shift)

immobilization of a polymerase

e Pacific Biosciences, Life/Visigen, LI-COR Biosciences
Single molecule: polymerase immobilized



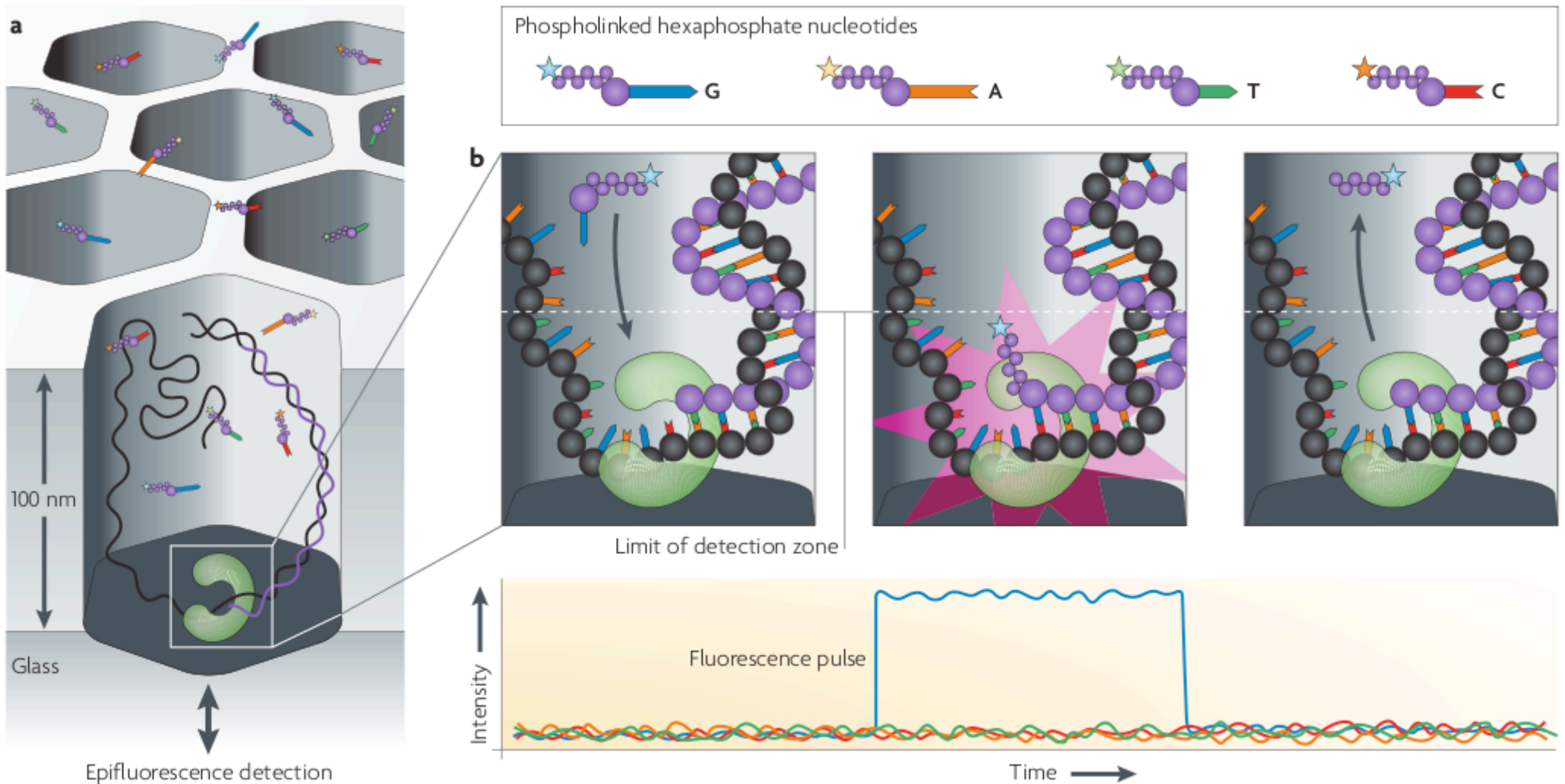
Thousands of primed, single-molecule templates

Pacific Biosciences

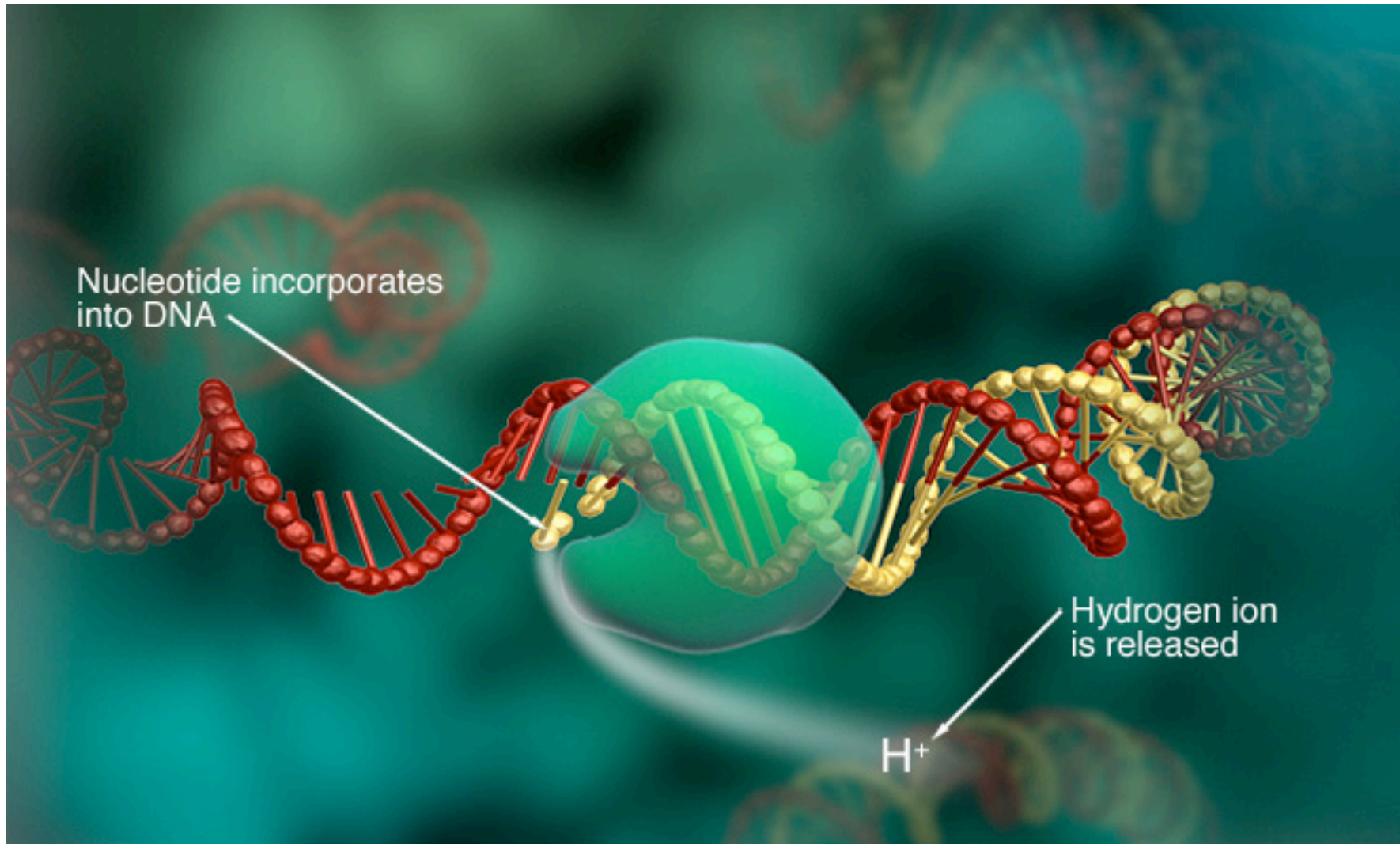
zero mode waveguide

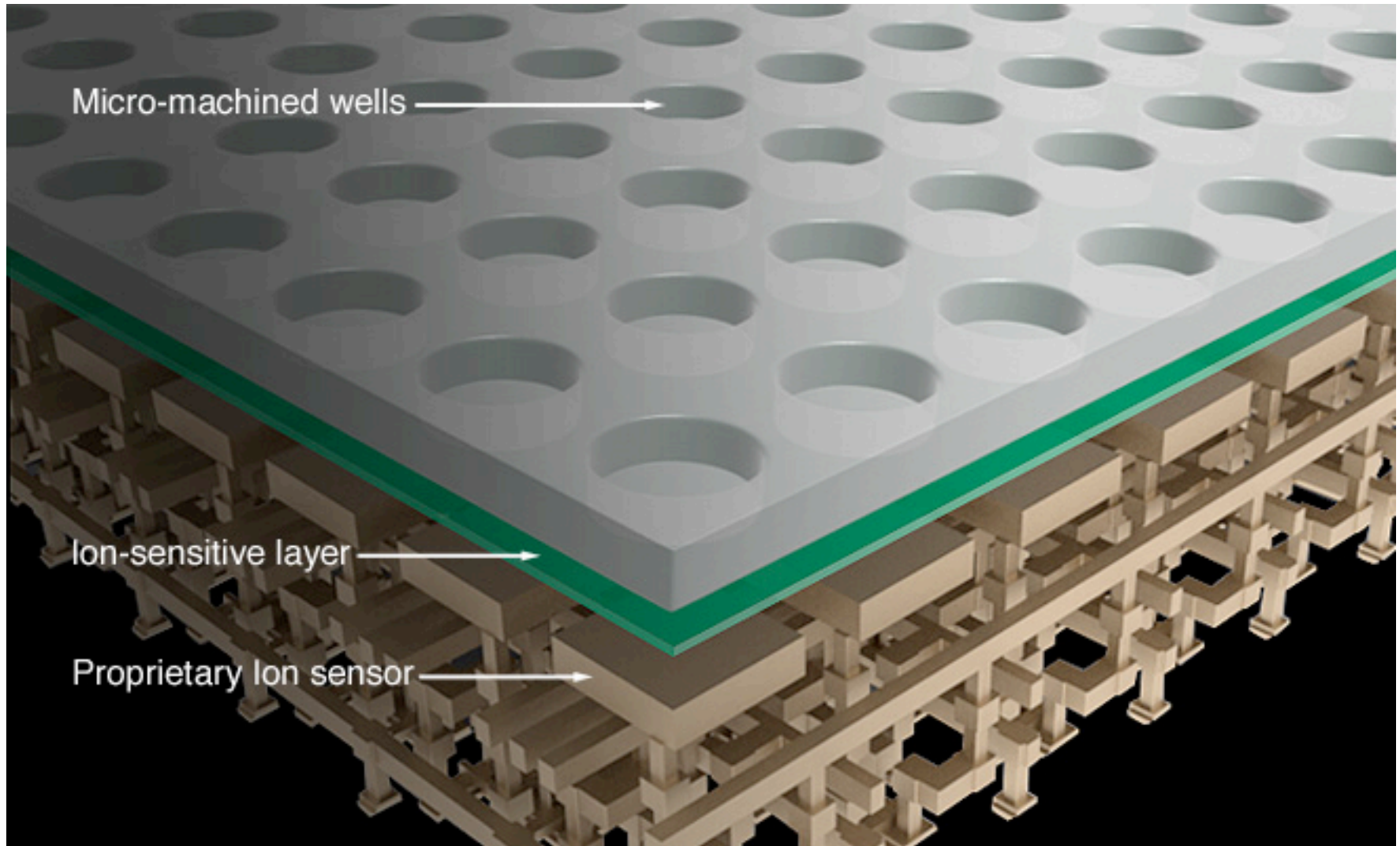
ZMRT technology → True single molecule

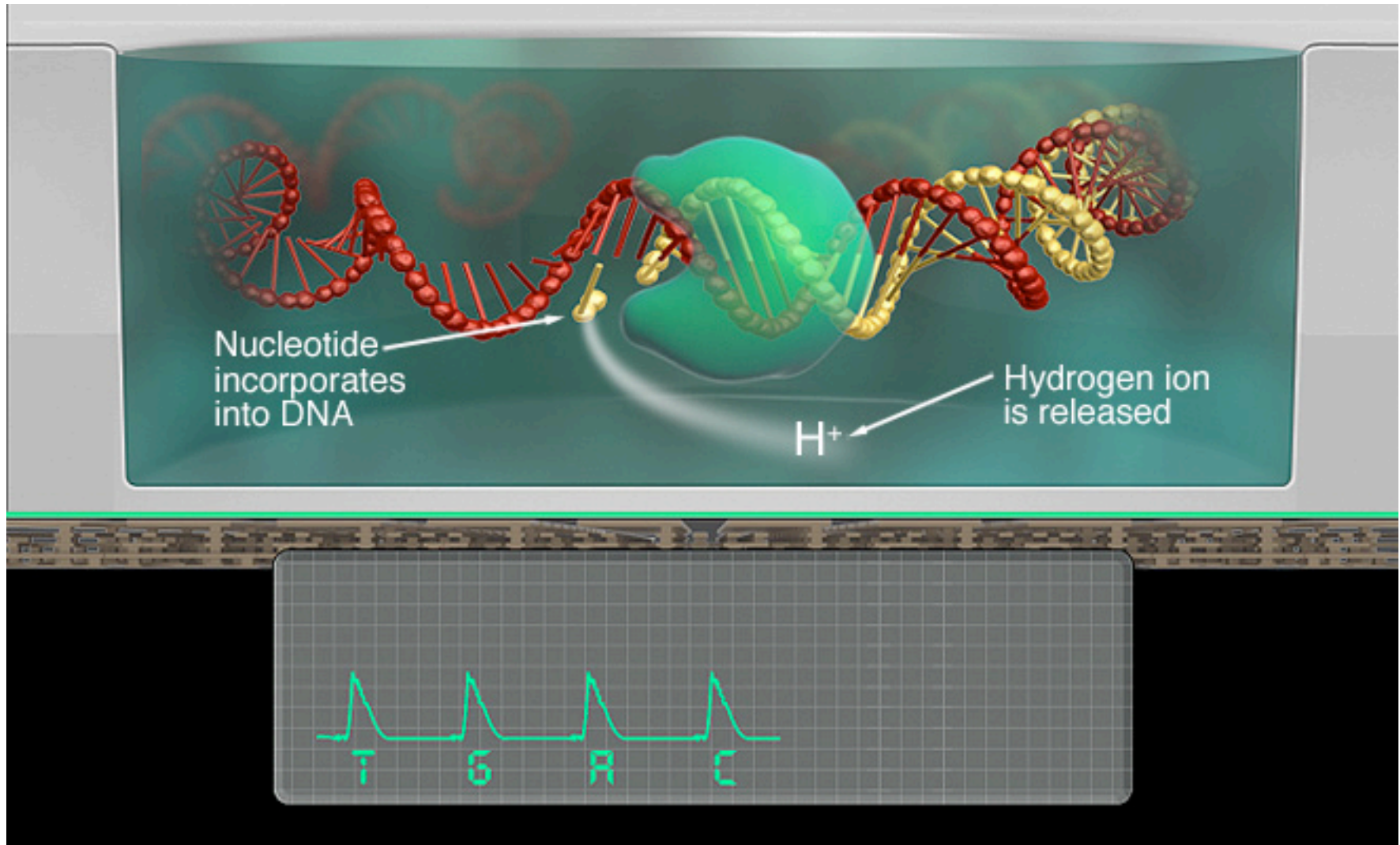
Pacific Biosciences — Real-time sequencing

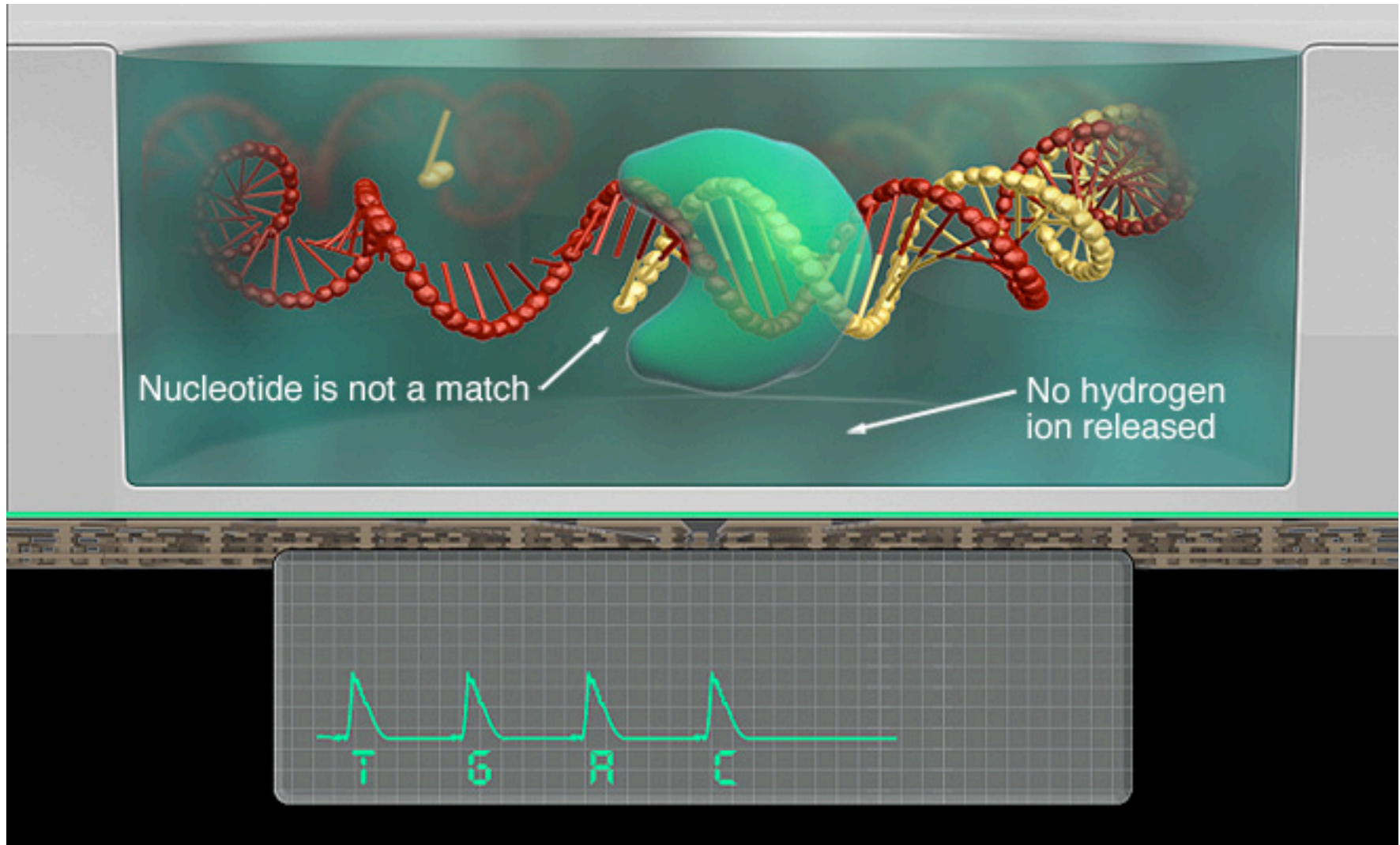


Ion Torrent



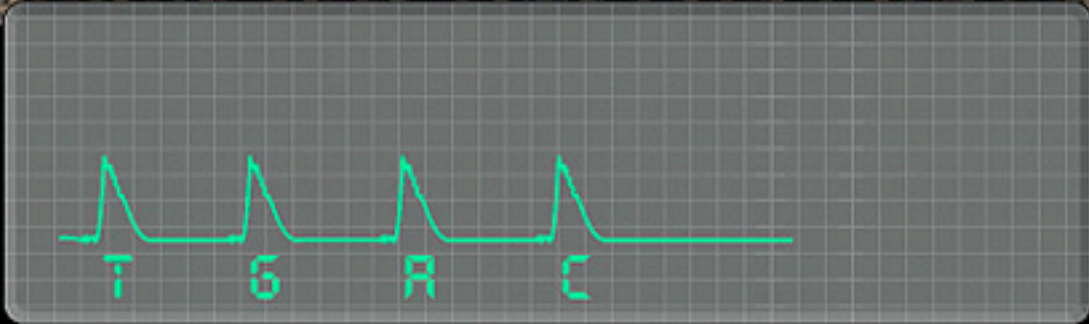


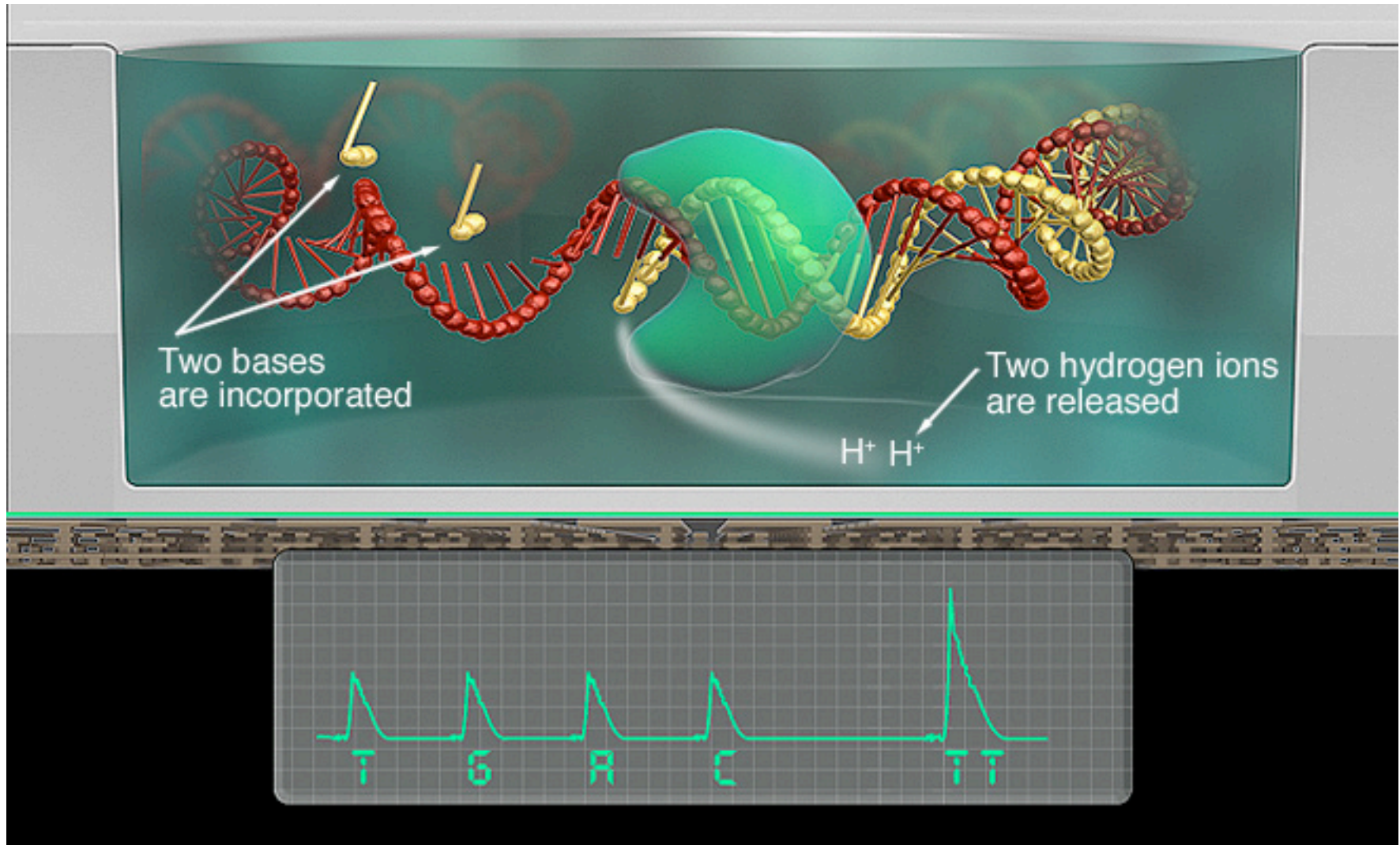


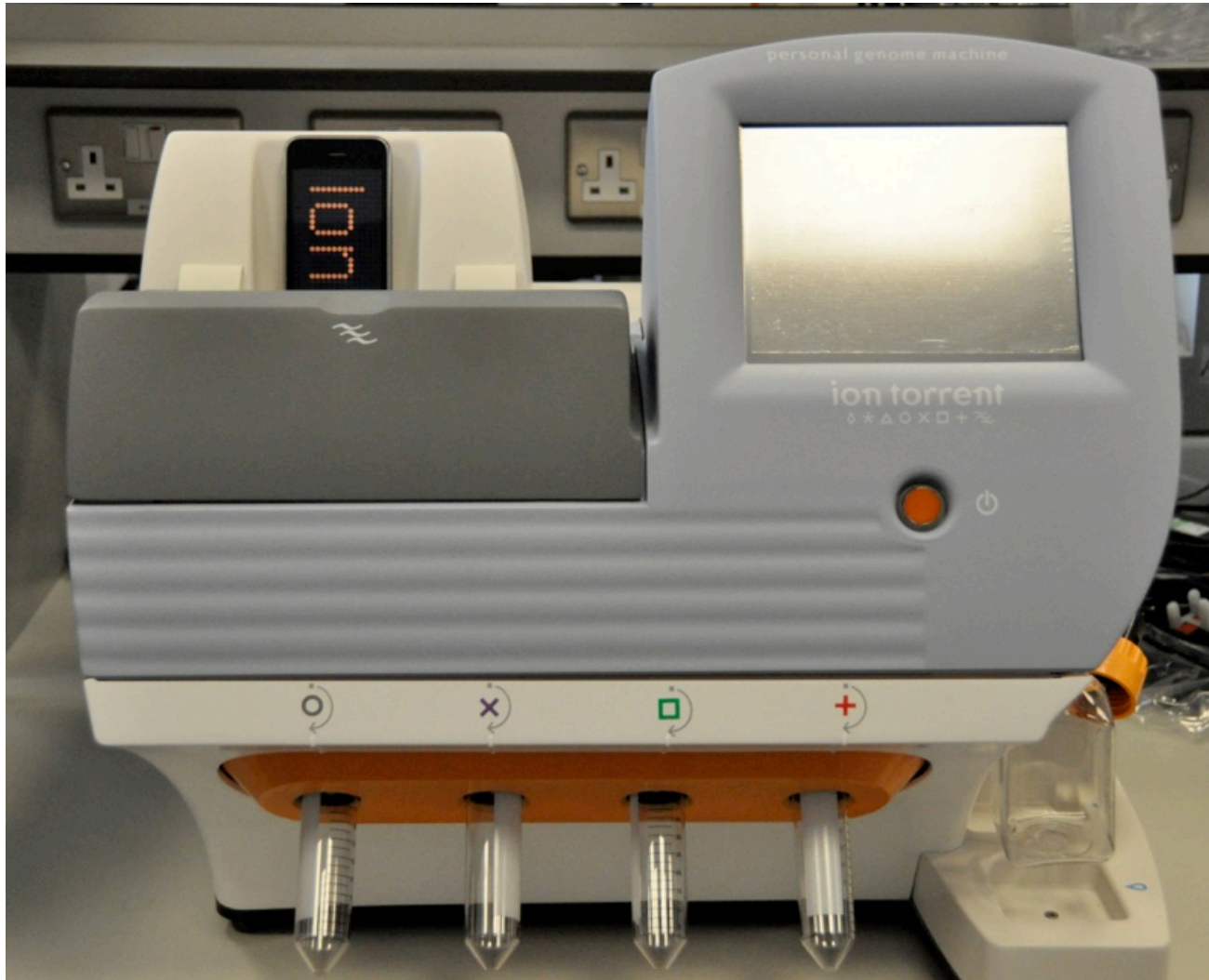


Nucleotide is not a match

No hydrogen ion released







The challenges of sequencing by synthesis

Carl W Fuller¹, Lyle R Middendorf², Steven A Benner³, George M Church⁴, Timothy Harris⁵, Xiaohua Huang⁶, Stevan B Jovanovich⁷, John R Nelson⁸, Jeffery A Schloss⁹, David C Schwartz¹⁰ & Dmitri V Vezenov¹¹

DNA sequencing-by-synthesis (SBS) technology, using a polymerase or ligase enzyme as its core biochemistry, has already been incorporated in several second-generation DNA sequencing systems with significant performance. Notwithstanding the substantial success of these SBS platforms, challenges continue to limit the ability to reduce the cost of sequencing a human genome to \$100,000 or less. Achieving dramatically reduced cost with enhanced throughput and quality will require the seamless integration of scientific and technological effort across disciplines within biochemistry, chemistry,

Table 1 Selected SBS platforms

Synthesis strategy	Company	Platform	Colors	Sequencing process	Amplification	Enzyme
Real-time	Pacific Biosciences	Zero-mode waveguide array	4	Continuous polymerization of labeled dNTPs	Single molecule	Polymerase
	Life Technologies; Visigen	Array of polymerase complexes	4	Continuous polymerization of labeled dNTPs	Single molecule	Polymerase
Synchronous-controlled	Life Technologies; ABI	SOLiD	4	Ligation of labeled 5 nt oligos	Yes, emulsion PCR	Ligase
	Danaher; Dover	Polonator	4	Ligation of fluorescently labeled 9 nt oligos	Yes, emulsion PCR	Ligase
	Illumina	Genome Analyzer	4	Polymerization using fluorescently labeled, reversibly terminating dNTPs	Yes, bridge amplification	Polymerase
	Roche; 454 Life Sciences	Genome Sequencer FLX	1	Polymerization of dNTPs added singly, luminescence detection of pyrophosphate	Yes, emulsion PCR	Polymerase
	Helicos	Heliscope	1	Polymerization of fluorescently labeled dNTPs added singly	Single molecule	Polymerase
Asynchronous with base-specific terminators	Various; requires high-resolution electrophoresis	Sanger dideoxy-sequencing	1, 4	Polymerization of fluorescently labeled ddNTPs with unlabeled dNTPs	Yes, clones or PCR	Polymerase

ddNTP, 2',3'-dideoxy-nucleotides.

- Simon Nicolas EMBL
- tutorial ISMB
 - Gunnar Ratsch and Ali Mortazavi
- Comparison data from:
 - - E Mardis, Trends in Genetics 24 (2008) 133
 - - R A Holt, S J M Jones, Genome Res 18 (2008) 839
 - - J Shendure, H Ji, Nature Biotech 26 (2008) 1135

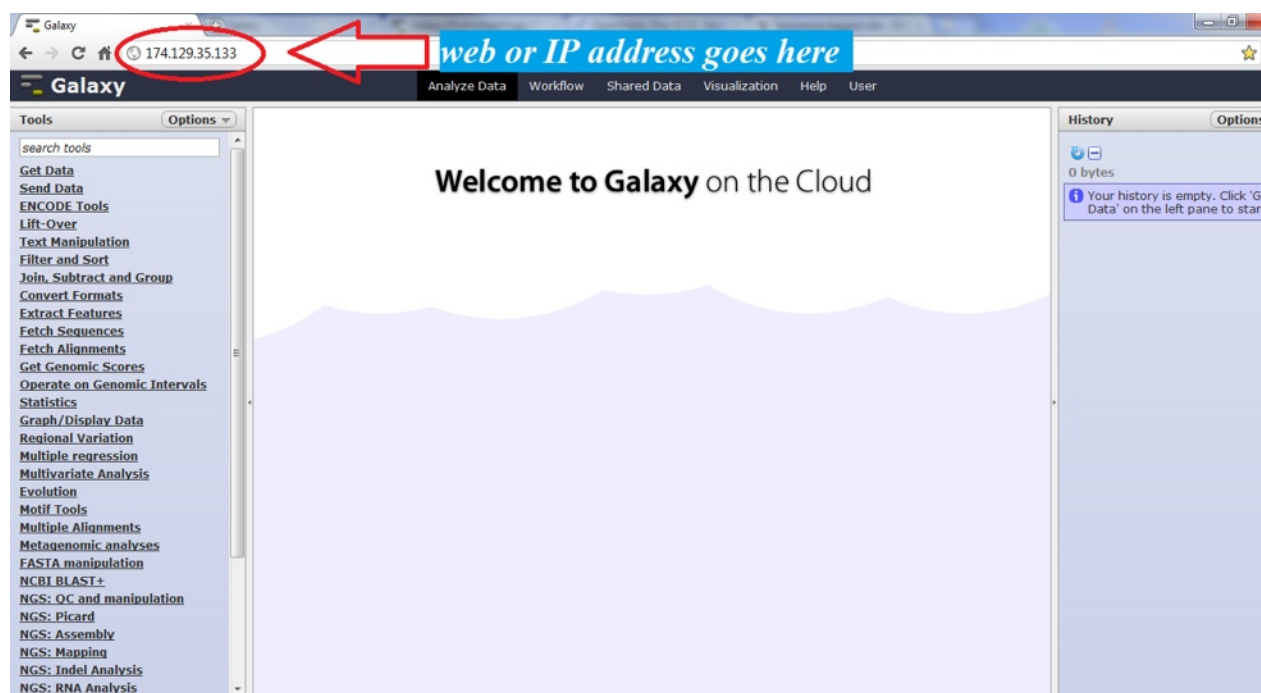
Introduction to Galaxy

Galaxy is an integrated tool management system with a user-friendly graphical user interface (GUI). It is designed for running multiple bioinformatics tools on genomic data in a single point-and-click environment. In this tutorial we are going to get acquainted with Galaxy. We'll see how to access Galaxy from your machine, get oriented and perform some basic tasks. Because genomic data is usually very large and the bioinformatics tools which we use tend to take a very long time to execute, it is a good idea to be running the data analysis on a dedicated computer system as opposed to on your own laptop or machine. This is why in this tutorial series we will be using Galaxy running on a remote machine ("on the Cloud") which has been set up just for you. This machine on the Cloud, together with a copy of Galaxy will be accessible to you for the duration of this course. We will be providing instructions towards the end of the course on how to set up your own system on the Cloud and get Galaxy running once this course is over.

In order for you to be able to access Galaxy on your assigned dedicated machine on the Cloud, you have been given a web or IP address in the form of A.B.C.D where A, B, C and D are numbers separated by dots. Please note it down! You will need it in order to access Galaxy from the web browser on your laptop.

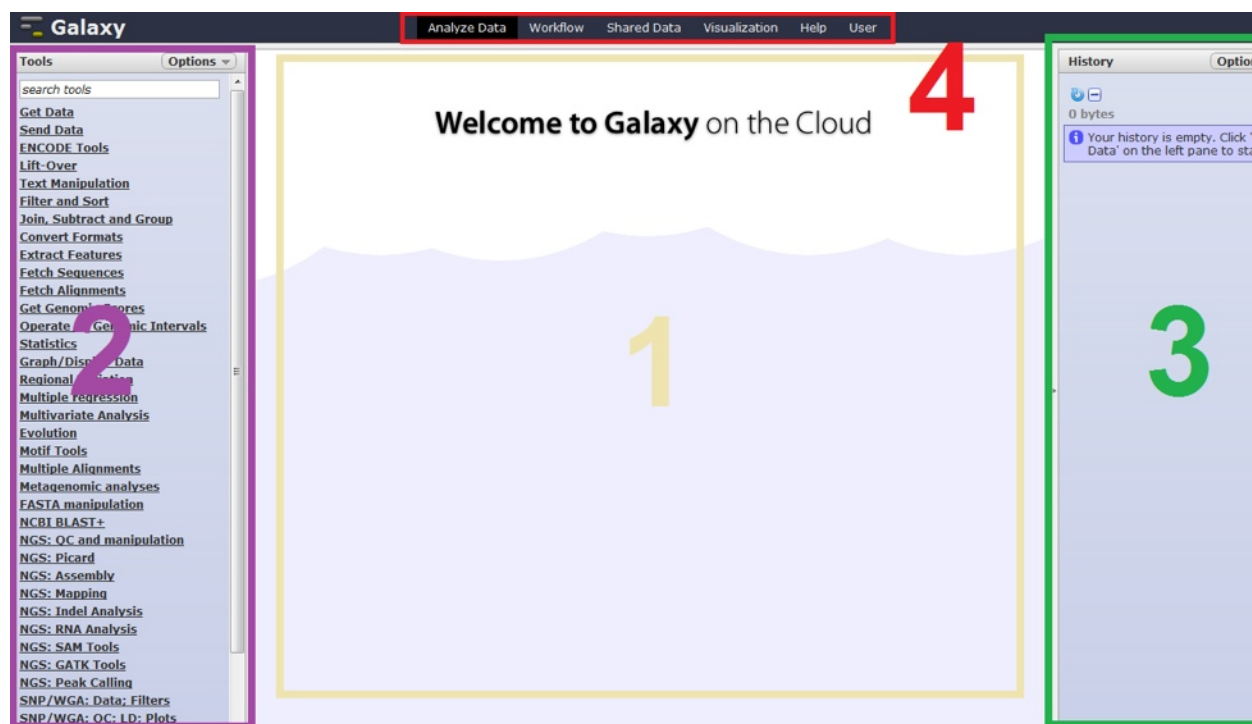
Getting Started with Galaxy

Open up a web browser of your choice, and enter the web or IP address given to you into the Address Bar and press ENTER. You should see the main Galaxy screen as shown below. This example was run using the Google Chrome web browser.



This is the main ANALYZE DATA window where data analysis is performed. Let's take a look at what this page consists of.

The main page is composed of 4 primary sections which are shown below:



1. The center page is where you will be viewing the data and entering parameter values when running tools. Now you don't see anything because we haven't loaded any data and we didn't run any tools yet.
2. The left column is where all of the tools and commands are located, grouped by major headings. You will be selecting tools from here when loading and analyzing data.
3. The right column is where a history of all the commands that were run are logged. The history is a very powerful feature of Galaxy which allows you to keep track of the steps that were undertaken from the very beginning of loading data, to the very last analysis step. We will see later on how to share your history with your collaborators and how to create workflows from them. Once again, the history is empty because we haven't run any tools yet.
4. The very top of the window contains several menus which allow you to move away from this main ANALYZE DATA page. You can move from one view to another without losing any of your data. We'll be using some of the other Galaxy functionality offered here in future tutorials.

You should always remember to log in to Galaxy by clicking on the USER menu item and LOG IN with your email address and password. Even though it is not required to run analyses, logging in with your account gives you much more power and functionality when it comes to managing histories and workflows. We'll see this later on. Now that we've seen what the various panes do, let's explore Galaxy by first importing some data!

Importing Data into Galaxy

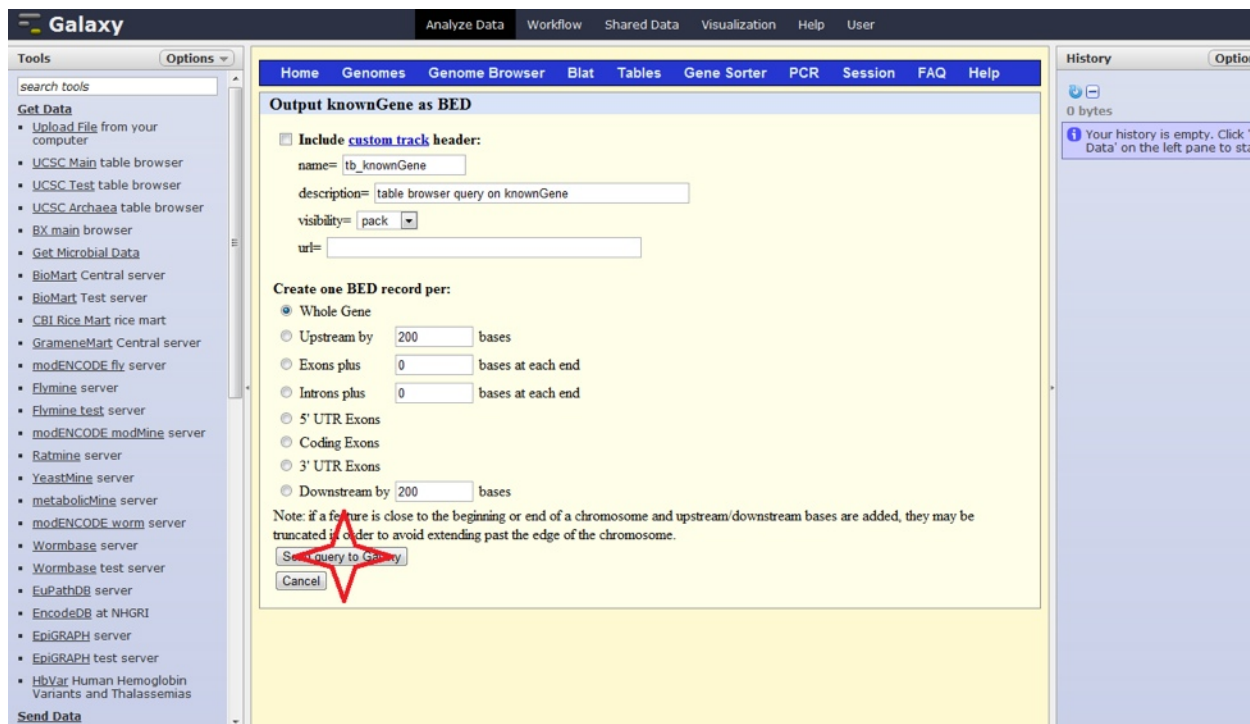
You can import data into Galaxy from a large variety of sources: from a local file on the machine, from shared data libraries and from numerous online data repositories. For the purposes of this tutorial, let's import some data from the UCSC [Table Browser](#). This [electronic resource](#) is based at the University of California at Santa Cruz, a major genomics research institution that has become a world-wide standard genomics data repository that is used by scientists in biology and bioinformatics all around the world. This online resource allows researchers to both visualize genomic data and obtain official, curated and published genomic sequence data. Let's say that we want to import UCSC [curated human gene coordinates on chromosome X](#) from the hg18 reference genome build:

The screenshot shows the Galaxy web interface with the UCSC Table Browser tool selected. The tool configuration is as follows:

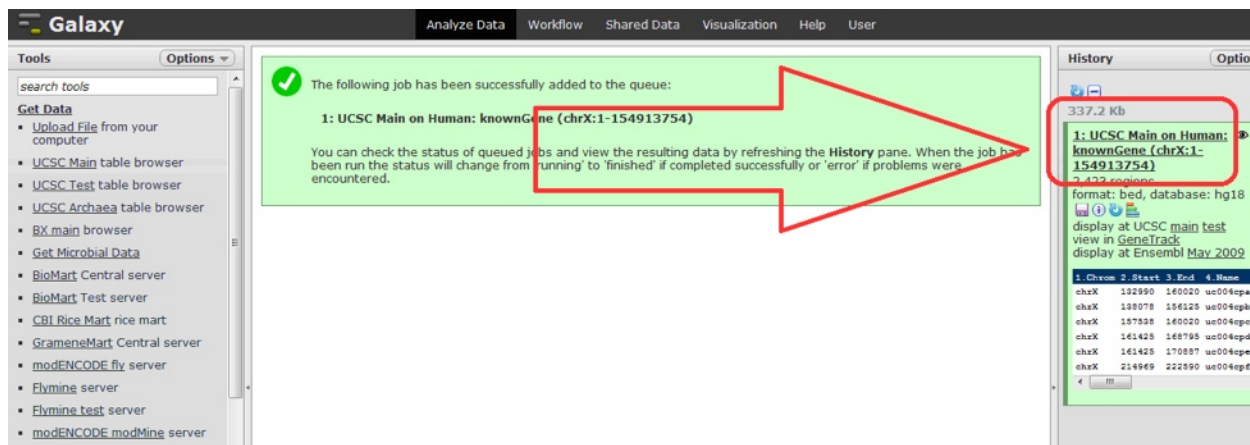
- Clade:** Mammal
- Genome:** Human
- Assembly:** Mar. 2006 (NCBI36/hg18)
- Group:** Genes and Gene Prediction Tracks
- Track:** UCSC Genes
- Table:** knownGene
- Region:** position chrX:1-154913754
- Output Format:** BED - browser extensible data
- Send output to:** Galaxy (checked), GREAT (unchecked)

The 'get output' button is highlighted with a red star.

1. Click on GET DATA in the tool menu in the left pane.
2. Click on UCSC Main table browser
3. Here in the center pane you will be selecting data parameters. In CLADE select Mammal, in GENOME select Human and in ASSEMBLY select Mar. 2006 (NCBI36/hg18).
4. In GROUP, select Genes and Gene Prediction Tracks, and in TRACK select UCSC Genes.
5. In TABLE select knownGene.
6. In REGION select the radio button for position and enter chrX in the data entry box. (Optional: If you press on the LOOKUP button after you type in 'chrX', it will fill in the genomic coordinates for chromosome X)
7. In OUTPUT FORMAT, select BED - browser extensible data.
8. Finally, make sure that there is a tick next to 'Send output to Galaxy', and then click GET OUTPUT.



On the next page, let's leave all the options at their default settings, and then click on the SEND QUERY TO GALAXY button. Galaxy will start running, and a new entry will appear in the right history pane. This was our first step data step. When it is first submitted, it will be colored grey meaning that it is scheduled to run, and as it runs it will be colored yellow. Once it finished running successfully, it will turn green and you will see the following screen. You can now click on the history name where you can see more details with a short summary showing that we've imported 2,423 regions together with a mini data view underneath.



Every history entry represents a unified collection of data (or a 'dataset'), and each will have a number and a name. The numbers will incrementally increase as you perform new operations on your data and the data at each stage will be saved and made available to you in this pane. You can click on the name to expand, or collapse the history entry view. Once expanded, there are several buttons that are available next to the name that allow you to view the data (eye), to edit the parameters or attributes of this data (pencil), or delete it (X). There are also some additional buttons underneath the title that allow you to download the data (floppy drive), get more information (i), rerun this step (arrow) or several buttons which allow you to visualize this data. In addition, at the

very bottom of this entry is a history mini-data view which only shows the first 6 data rows, and it is useful to get a general idea of how the data is structured. Here you can see that the 1st column contains the chromosome name, the 2nd column contains the START and the 3rd column contains the END genomic location coordinates for a particular gene, etc. Let's take a look at this data in greater detail. If you click on the little “eye” button, you'll be able to view the entire dataset in the center pane:

The screenshot shows the Galaxy interface with a table of genomic data. The table has columns for chromosome (chrX), start, end, and other genomic coordinates. The history panel on the right shows a dataset named "UCSC Main on Human knownGene (chrX:1-154913754)" with an eye icon highlighted by a red arrow.

chrX	Start	End	Score	Strand	Database/Build
chrX	132990	160020	0	+	Human Mar. 2006 (NCBI36/hg18)
chrX	138078	156125	0	+	Human Mar. 2006 (NCBI36/hg18)
chrX	157538	160020	0	+	Human Mar. 2006 (NCBI36/hg18)
chrX	161425	168795	0	-	Human Mar. 2006 (NCBI36/hg18)
chrX	161425	170887	0	-	Human Mar. 2006 (NCBI36/hg18)
chrX	161425	170887	0	-	Human Mar. 2006 (NCBI36/hg18)

Next to the “eye” button is a little “pencil” button. If you click on that, you can edit the attributes of this dataset:

The screenshot shows the Galaxy interface with the "Edit Attributes" dialog box open. The dialog box has fields for Name, Info, Database/Build, Number of comment lines, Chrom column, Start column, End column, Strand column, Name/Identifier column, and Score column for visualization. A red box highlights the Name field, which contains "Genes on chrX". A red arrow points to the "Save" button at the bottom of the dialog box.

Here let's change the name of this dataset to "Genes on chrX." You can also see that it is here where you can specify which columns contain which data, e.g. that column 1 contains chromosome information, etc. We don't really want to change anything else here, so just click on SAVE. You will then see that the name of this dataset was changed.

Note: next to the "pencil" button is an "X". If you click on that, this will delete a dataset from your history. This is useful if you make a mistake along the way and you end up with data that you don't need for subsequent steps. Otherwise, it is not recommended to delete data from your history since this will make it much harder (if not impossible) to retrace all of your steps from start to finish, and this defeats the whole purpose of the history!

Preparing and Manipulating Data for Analysis

Once you have a dataset imported into Galaxy, you may need to manipulate the data by sorting, filtering or cutting columns out of it before you can continue. This may be necessary because some tools may require that the data be presented in a certain format, but usually we may want to organize the data anyway so that it only contains the information we really need, in as a clean format as possible. This is recommended because usually most raw genomic datasets come with a lot of information that might not always be necessary for your analysis. In this section we shall be performing a few basic operations, but keep in mind that many more are possible.

We already have a chromosome X gene dataset in Galaxy. Let's say that we are only interested in the UCSC gene names, their chromosomal locations, the strand where they are located and the exon counts on this chromosome. To keep only the data we need, we'll cut out the columns with this information and discard the rest.

The screenshot shows the Galaxy interface with the 'Cut' tool selected. In the 'Tools' pane on the left, 'Text Manipulation' and 'Cut columns from a table' are highlighted with red boxes. In the 'Cut' tool configuration, 'Cut columns:' is set to 'c1,c2,c3,c6' and 'From:' is set to '1: Genes on chrX'. A warning message is displayed: 'WARNING: This tool breaks column assignments. To re-establish column assignments run the tools and click on the pencil icon in the latest history item.' Below the warning, an example shows how columns 1 and 3 are cut from a dataset, resulting in a tabular format with columns 'apple, is, good' and 'windows, is, bad'.

1. Click on TEXT MANIPULATION in the tool pane, and then select the CUT tool. This tool will extract the columns we want and create a new dataset from them.

- In the tool parameters, in CUT COLUMNS, type in "c1,c2,c3,c6,c4,c10". Notice here that we can order the columns any way we like. In this case we are placing the strand (column 6) before the gene ID (column 4).
- Make sure that FROM contains our initial dataset. Here it is the only one present, however when you work with the CUT tool in the future, always make sure you select which dataset to work on.
- Press EXECUTE. You should get a now have a new dataset in your history numbered as 2.

We have now created a new dataset in which we have eliminated unwanted information and kept only those pieces of information that we'll be using in subsequent steps. Go ahead and rename this dataset to something you like, and then click SAVE. It is always a good idea to rename your datasets as they are created to make sure you can recognize it. It's very easy to get lost in many hundreds of data steps and datasets in Galaxy when performing complicated analyses!

For the purposes of this tutorial, we'll be needing this dataset in INTERVAL format. If you remember we imported this dataset from UCSC in BED format. You can perform this kind of conversation of data formats right here in the pencil tool. Explore the options available. If the CUT tool did not already convert the dataset to INTERVAL, then do so now and click SAVE. Interval format is a simple format in which Galaxy is told that the dataset contains a "start" and "end" column which defines a genomic interval for a feature, in this case, for genes. Usually Galaxy is good at guessing the format for you and you do not have to do this step. When a file is in interval format, there are other attributes that can be set under the pencil tool, where you tell Galaxy which columns contain which pieces of data, such as strand information etc. In case you have to convert your dataset to interval, make sure the columns are properly set. Column 1: chromosome, Column 2: Start, Column 3: End, Column 4: Strand, Column 5: Name.

The screenshot shows the Galaxy web interface. On the left, the 'Tools' panel is visible, with 'Filter and Sort' highlighted under 'Text Manipulation'. The main area displays the 'Filter' tool configuration. The 'Filter' dropdown is set to '4: Genes chrX concise'. Below it, the 'With following condition' field contains 'c6<=2'. A red box highlights the dropdown and the condition field. A red arrow points to the 'Execute' button. The right panel shows the 'History' tab with two datasets: '4: Genes chrX concise' and '1: Genes on chrX'. The '4: Genes chrX concise' dataset is selected and shows a table of genomic data with columns for chromosome, start, end, strand, and name.

Filter:
4: Genes chrX concise
Dataset missing? See TIP below.
With following condition:
c6<=2
Double equal signs, ==, must be used as shown above. To filter for an arbitrary string, use the Select tool.

TIP: Double equal signs, ==, must be used as "equal to" (e.g., c1 == 'chr22')

TIP: Attempting to apply a filtering condition may throw exceptions if the data type (e.g., string, integer) in every line of the columns being filtered is not appropriate for the condition (e.g., attempting certain numerical calculations on strings). If an exception is thrown when applying the condition to a line, that line is skipped as invalid for the filter condition. The number of invalid skipped lines is documented in the resulting history item as a "Condition/data issue".

TIP: If your data is not TAB delimited, use Text Manipulation->Convert

Syntax
The filter tool allows you to restrict the dataset using simple conditional statements.

- Columns are referenced with c and a number. For example, c1 refers to the first column of a tab-delimited file
- Make sure that multi-character operators contain no white space (e.g., <= is valid while < = is not valid)
- When using 'equal-to' operator double equal sign '=' must be used (e.g., c1=='chr1')
- Non-numerical values must be included in single or double quotes (e.g., c6=='*')
- Filtering condition can include logical operators, but make sure operators are all lower case (e.g., (c1=='chrX' and c1!='chrY') or not c6=='*')

Example

- c1=='chr1' selects lines in which the first column is chr1
- c3-c2<100*c4 selects lines where subtracting column 3 from column 2 is less than the value of column 4 times 100
- len(c2.split(',')) < 4 will select lines where the second column has less than four comma separated elements
- c2>= 1 selects lines in which the value of column 2 is greater than or equal to 1
- Numbers should not contain commas - c2<=44,554,350 will not work, but c2<=44554350 will
- Some words in the data can be used, but must be single or double quoted (e.g., c3=='exon')

Now let's filter our data to only contain those genes that have a certain number of exons. Let's say we only want to see those genes that have at most 2 exons.

1. Click on the FILTER AND SORT tool heading, and select the FILTER tool.
2. Select the appropriate dataset (the latest concise, cut one, whatever you called it) in FILTER:
3. Provide a condition by typing in "c6<=2". This condition means, select only those rows (genes) in our dataset whose column 6 (exon count) is less than or equal to 2.
4. Click on EXECUTE. This will create a new dataset which should contain only 436 genes. In the data mini-summary Galaxy also tells you that this dataset is only 17.99% the size of our initial one.

Click on the eye tool on this new dataset, and you'll see that column 6 (exon count) in this file has values of 1 or 2, which means it worked! Once again, rename this dataset and give it a useful name, such as "chrX Genes <= 2 exons".

In our next step, we're going to see how to join two datasets together. Before we go on, we need to import a new dataset that we'll be joining with this one. Let's say we are interested in looking at the predicted DNase-I hypersensitive sites on chromosome X and joining this dataset with our previous one.

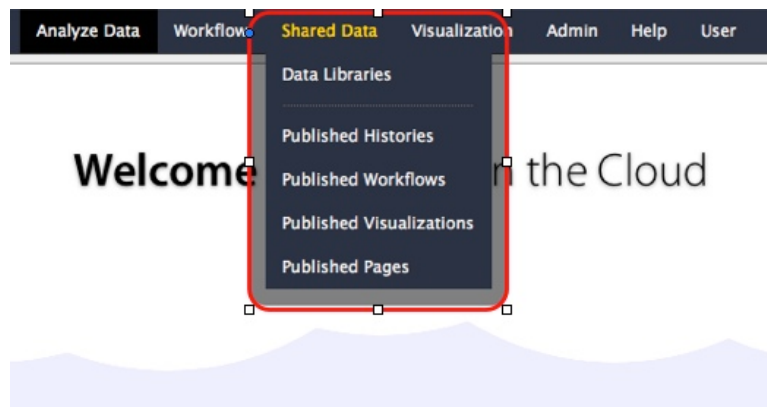
The screenshot shows the Galaxy interface with the UCSC Main Table Browser tool selected. The tool configuration is as follows:

- clade: Mammal
- genome: Human
- assembly: July 2003 (NCBI34/hg16)
- group: Regulation
- track: NHGRI DNaseI-HS
- table: nhgriDnaseHs
- region: position chrX:1-153692391
- identifiers (names/accessions): paste list, upload list
- filter: create
- intersection: create
- correlation: create
- output format: BED - browser extensible data
- Send output to: Galaxy GREAT
- output file: (leave blank to keep output in browser)
- file type returned: plain text gzip compressed
- get output: summary/statistics

1. Select the GET DATA tool heading, and select the UCSC Main Table Browser tool.
2. For CLADE select Mammal, for GENOME select Human, for ASSEMBLY select July 2003 (NCBI34/hg16).
3. For GROUP select Regulation, for TRACK select NHGRI DNaseI-HS

4. For TABLE select nhgriDnaseHs.
5. Under REGION make sure you select the position radio button and type in “chrX”.
6. Click on GET OUTPUT.
7. On the next page, click on SEND QUERY TO GALAXY.

You should notice that we have selected data from a previous genome assembly build (version), ie. hg16 and not hg18. This would be a good time to once again rename this dataset with a name of your choosing, but make sure that “hg16” is in the name to make sure you can identify it. Every build has different genome coordinates, and so the coordinates will not match with our first dataset. This could happen by accident, or in reality it could be that you have a dataset with genomic coordinates from a previous version of the genome assembly. In our case, DNase-I hypersensitive sites data is only available under build hg16. You can use Galaxy to convert this build to the newer hg18 by using the Convert Genome Coordinates tool under the LIFT-OVER tool heading. To save time and learn how to upload your own files to Galaxy, we’ve done the conversion to hg18 already. We provide you with the file “DNase HS hg18.bed” through the Galaxy “Shared Data” system:



1. Navigate to Data Libraries, MitoSequences, Intro and find the “DNase_HS_hg18.bed” file

Name	Message	Uploaded By	Date	File Size
chr22				
chrM				
Exome Information	Bait and target coordinates			
Intro				
DNase_HS_hg18.bed		ohofmann@hsph.harvard.edu	2011-10-24	5.9 Kb
gene1.fa		ohofmann@hsph.harvard.edu	2011-06-28	4.0 Kb
gene2.fa		ohofmann@hsph.harvard.edu	2011-06-28	4.5 Kb
Reference Data				
Visualization	Tracks for UCSC and IGV			
contaminants.fa	Potential contaminants identified in a sequencing data set	ohofmann@hsph.harvard.edu	2011-06-27	120 bytes

2. Import it into your current user history by clicking on the arrow next to it, “Import dataset into selected histories”, and the “Import library datasets” button

The screenshot shows the Galaxy Data Library interface for 'MitoSequences'. A table lists datasets with columns for Name, Message, Uploaded By, Date, and File Size. A context menu is open over one dataset, listing actions such as 'Edit information', 'Move this dataset', 'Use template', 'Make public', 'Edit permissions', 'Upload a new version of this dataset', 'Import this dataset into selected histories', 'Download this dataset', and 'Delete this dataset'.

Name	Message	Uploaded By	Date	File Size
chr22				
chrM				
Exome Information	Bait and target coordinates			
Intro				
		ohofmann@hsph.harvard.edu	2011-10-24	5.9 Kb
		ohofmann@hsph.harvard.edu	2011-06-28	4.0 Kb
		ohofmann@hsph.harvard.edu	2011-06-28	4.5 Kb
	for UCSC and IGV			
	contaminants identified in a sequencing data set	ohofmann@hsph.harvard.edu	2011-06-27	120 bytes

- Go back to the main analysis page and you will find the BED file in your current history, ready to use

Alternatively, you can download the file from our server and manually upload it into the current history:

The screenshot shows the 'Upload File' tool in Galaxy. The 'Get Data' menu is open, and 'Upload File from your computer' is selected. The 'File' section shows 'Choose File' selected, with 'DNase HS hg18.bed' listed. The 'Genome' section shows 'Human Mar. 2006 (NCBI36/hg18)' selected. The 'Execute' button is highlighted with a red star.

- Select the GET DATA tool heading, and select Upload File from your Computer.
- Under FILE, select Choose File and browse to the directory where this file is located.
- Under GENOME, type "hg18" in the search bar and select it.
- Press EXECUTE.

Now that we have our converted file in hg18, we can do some more involved analysis.

Analyzing Data

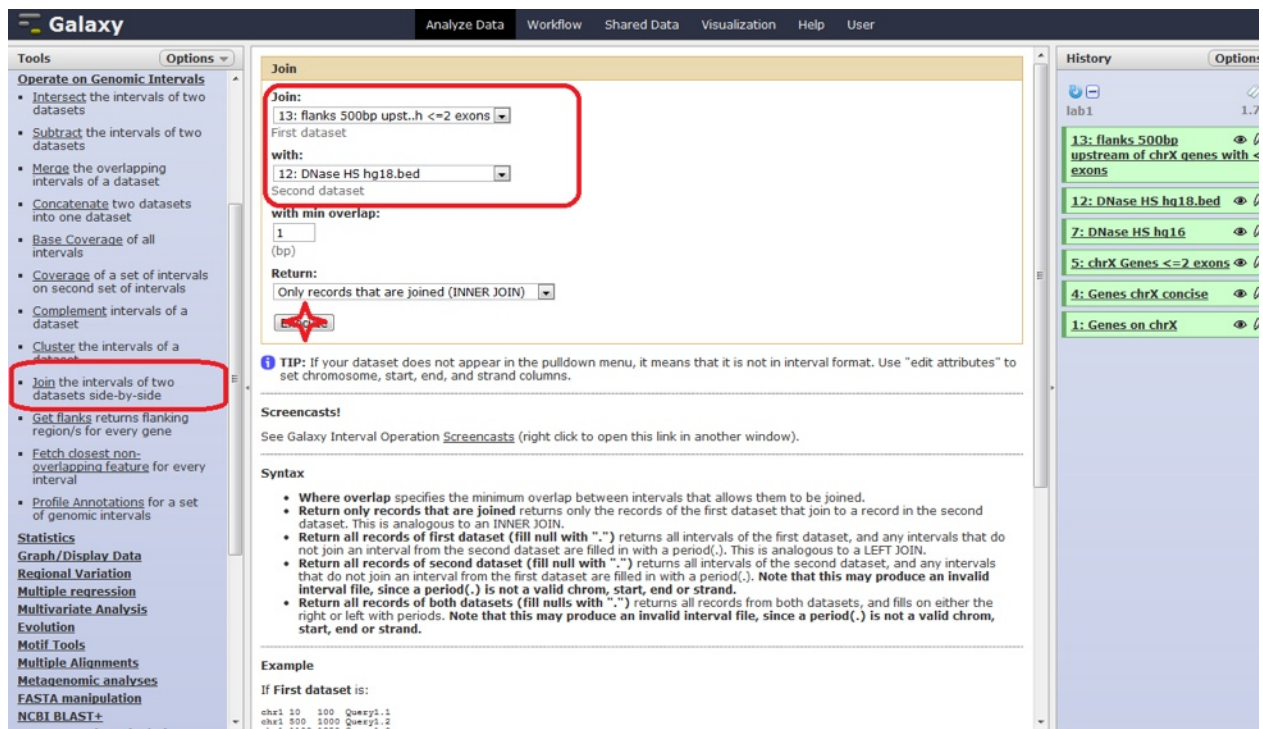
Now that we have a few datasets ready for analysis after we have performed some pre-processing, we can do some analysis. There are very many tools that can be used for analysis in Galaxy, but here we will just show you some examples of what can be done.

Let's say we are interested in determining if there are any putative DNase hypersensitive sites in the upstream flanking regions from our gene locations on chromosome X. To check this, we will need to obtain the coordinates of those flanking regions first. Then we shall join the two datasets to see whether they have an intersection and where those intersections lie.

The screenshot shows the Galaxy web interface with the 'Get flanks' tool configuration. The 'Select data' dropdown is set to '5: chrX Genes <=2 exons'. The 'Region' is set to 'Whole feature' and 'Location of the flanking region/s:' is set to 'Upstream'. The 'Offset' is 0 and the 'Length of the flanking region(s):' is 500. The 'Execute' button is highlighted with a red star. The 'Tools' sidebar on the left has 'Get flanks returns flanking region/s for every gene' highlighted with a red box. The 'History' panel on the right shows a list of datasets including '12: DNase HS hg18.bed', '7: DNase HS hg16', '5: chrX Genes <=2 exons', '4: Genes chrX concise', and '1: Genes on chrX'.

1. Select the OPERATE ON GENOMIC INTERVALS tool heading and click on the Get Flanks tool.
2. Under SELECT DATA, select the dataset which contains chromosome X genes with 2 or fewer exons.
3. Under REGION, keep the option at Whole Feature and keep the option of Upstream under LOCATION OF FLANKING REGIONS.
4. Keep an OFFSET of 0, but increase the Length of flanking region to 500.
5. Click Execute.

You can now rename this new dataset something like "flanks 500bp upstream of chrX genes with <= 2 exons". Now we can join this dataset with our DNase hypersensitive sites dataset.



1. Under the OPERATE ON GENOMIC INTERVALS tool heading, select the Join tool.
2. For the FIRST dataset choose the dataset that contains the upstream flanking regions.
3. For the SECOND dataset choose the dataset that contains the DNase hypersensitive sites.
4. You can choose a Minimum Overlap, but keep it at the default of 1.
5. Under RETURN, keep the default of Only Records that are joined (INNER JOIN). This will only return the regions which are overlapping and nothing else.
6. Click EXECUTE.

Remember to rename the new dataset if you like. Let's click on the eye button and see what this data tells us. If the data does not fit the center pane very well, you can always resize the side panes by dragging your mouse over the pane edges. You can also hide panes by clicking on the pane edges. Let's hide the right history pane so that we view the data clearly.

Tools	Options	chrX	38545128	38545628	+	uc010ngx.1	2	chrX	38545601	38545674	34_2	500
Operate on Genomic Intervals	Intersect the intervals of two datasets	chrX	49530584	49531084	-	uc004doc.2	2	chrX	49530500	49531125	79_3	750
		chrX	53127786	53128286	+	uc004drv.1	2	chrX	53128065	53128441	83_2	500
Subtract the intervals of two datasets	Merge the overlapping intervals of a dataset	chrX	70629024	70629524	-	uc004dzz.1	1	chrX	70629391	70629554	101_2	500
		chrX	100549346	100549846	+	uc004ehm.1	2	chrX	100549292	100549706	110_3	750
Concatenate two datasets into one dataset	Base Coverage of all intervals	chrX	100549346	100549846	+	uc004ehh.1	2	chrX	100549292	100549706	110_3	750
		chrX	118776103	118776603	+	uc010nql.1	1	chrX	118776496	118776766	125_2	500
Coverage of a set of intervals on second set of intervals	Complement intervals of a dataset	chrX	118889818	118890318	-	uc004esb.1	1	chrX	118889822	118889905	126_2	500
		chrX	134013887	134014387	-	uc004eyf.1	1	chrX	134013907	134014564	144_3	750
Cluster the intervals of a dataset	Join the intervals of two datasets side-by-side	chrX	134013798	134014298	-	uc004eyr.2	2	chrX	134013907	134014564	144_3	750
		chrX	134305886	134306386	+	uc004eyr.2	2	chrX	134306304	134306879	147_3	750
Get flanks returns flanking region/s for every gene	Fetch closest non-overlapping feature for every interval	chrX	154147046	154147546	+	uc004fme.1	2	chrX	154146984	154147139	181_3	750
		chrX	154341470	154341970	-	uc004fnj.1	1	chrX	154341170	154341526	182_2	500

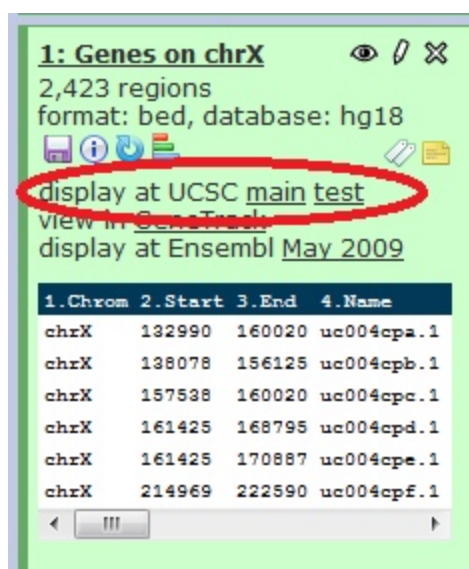
We can see that we have 13 upstream flanking regions of genes on chromosome X with 2 or fewer exons that have predicted DNase hypersensitive sites. The two datasets were joined together and are now placed side by side, with the only rows present that show an overlap. Column 1 shows the

chromosome, column 2 shows the start position, column 3 shows the end position of the upstream flanking regions, column 4 shows the strand, column 5 shows the USCS gene ID, column 6 shows the number of exons in that gene, column 7 is the chromosome again (from our 2nd joined dataset), column 8 shows the start position, column 9 the end position of the DNase hypersensitive region, column 10 shows the name of this DNase HS site and column 11 shows the hypersensitivity score.

Visualizing your Data

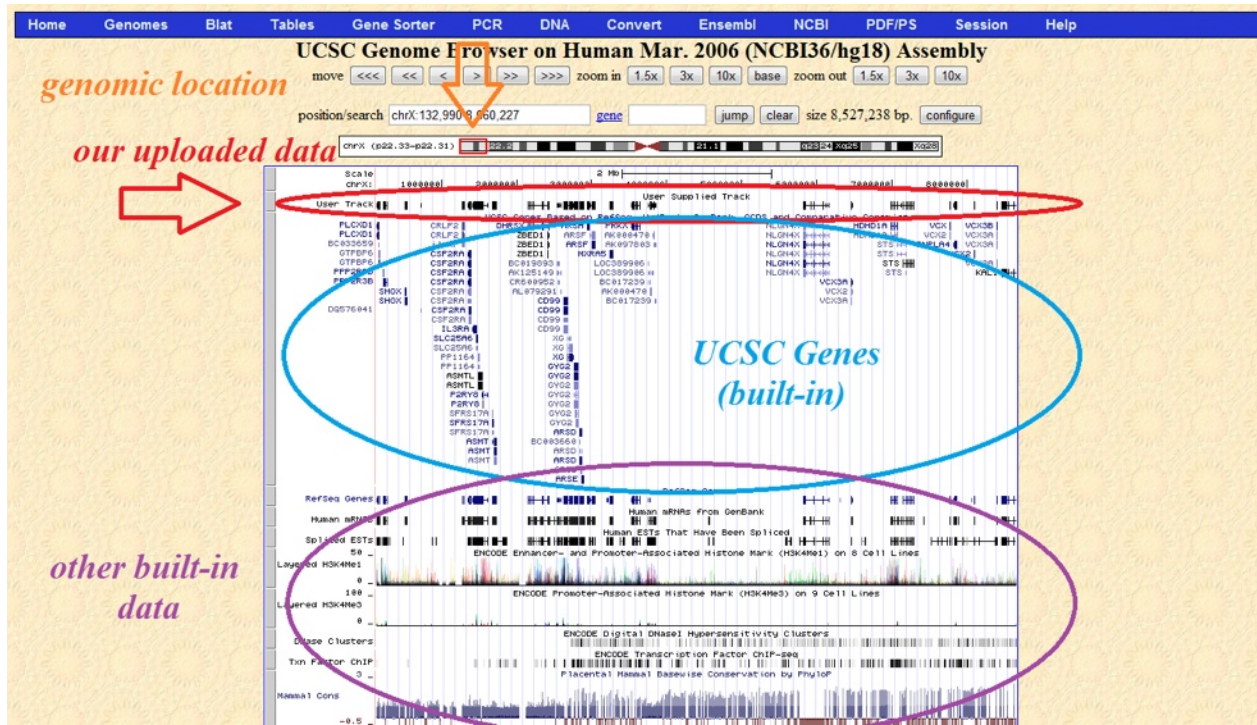
A picture is worth a thousand words. With Gigabytes of genomic data, the only way to get a good grasp of what it represents and what it means is by displaying it appropriately. We shall revisit the topic of visualization in a future tutorial, but let's perform some simple visualizations in Galaxy with the data that we already have.

Any time that you have a file in BED format (this is the format that we downloaded the data from the UCSC Table Browser), you can instantly visualize it using the UCSC Genome Browser, and this functionality is integrated into Galaxy. Our first dataset that we obtained from the UCSC Table Browser was the set of coordinates for UCSC genes from chromosome X.



Go back down in your history until you find our first dataset. You can see right under the title that the format is “bed”, so it can be easily visualized from this box. You will see a line which says “Display at UCSC...”. Click on MAIN. This will open up a new internet browser window or tab which will take you directly to the UCSC Genome Browser where you can view the data that we have just uploaded. The UCSC Genome Browser will be covered in more depth in the Visualization Tutorial later on in the course, but it is a relatively straightforward online resource that allows you to navigate the genome of your choosing with ease. The Genome Browser has the capacity to display a lot of data! Many tracks are built in, and can be turned on or off by using the controls at the bottom of the browser page. The data that we have just uploaded however (which is not built-in), is defined as a “custom track”. You can see that our data is titled “User Supplied Track”.

We can use this approach to visualize any BED files you may have, but what do we do about the other datasets which have that are not in BED format? We will have to create custom tracks for them by using special Galaxy tools.

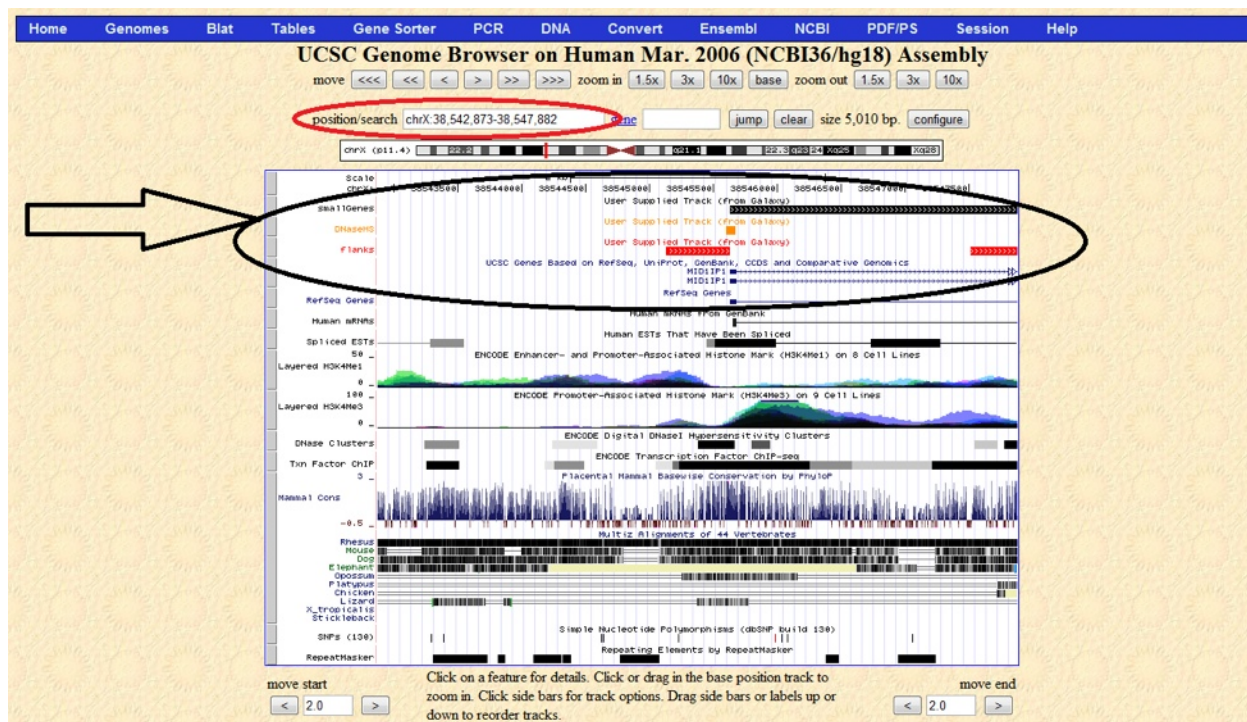


Let's say that we also wish to view the genes that have at most 2 exons, the DNase hypersensitive sites and the upstream flanking regions of those genes.

The screenshot shows the Galaxy 'Build custom track' tool interface. On the left, the 'Tools' sidebar is visible, with 'Build custom track for UCSC genome browser' circled in red. The main panel shows the 'Build custom track' tool with an 'Add new track' button and an 'Execute' button. A red starburst highlights the 'Add new track' button. Below the tool name, there is a description: 'This tool allows you to build custom tracks using datasets in your history for the UCSC genome browser. You can view these custom tracks on the UCSC genome browser by clicking on **display at UCSC main/test** link in the history panel of the output dataset.' A warning note states: 'Please note that this tool requires all input datasets(tracks) to have the same genome build. The tool throws an error when this requirement is not met. You may then have to choose a valid dataset or remove invalid tracks.'

1. Select the GRAPH/DISPLAY DATA tool heading and select the Build Custom Track for UCSC genome browser tool.

2. Click the Add new Track button. Select the datasets we wish to display together and keep pressing Add new Track until you have entered all of the datasets we need to be visualized. At each step give them a meaningful name for visualization (all of them! if you leave at least 2 tracks with the default “unnamed” name, it won’t display properly). Also select a color for this track of your choosing. Remember to include the “chrX genes <= 2 exons” dataset, “DNase HS hgr8.bed”, and “flanks 500bp upstream”.
3. When you’re done, click EXECUTE.
4. Once the job finishes running, click on DISPLAY AT UCSC MAIN as we did in the previous example on the newly created dataset history entry.



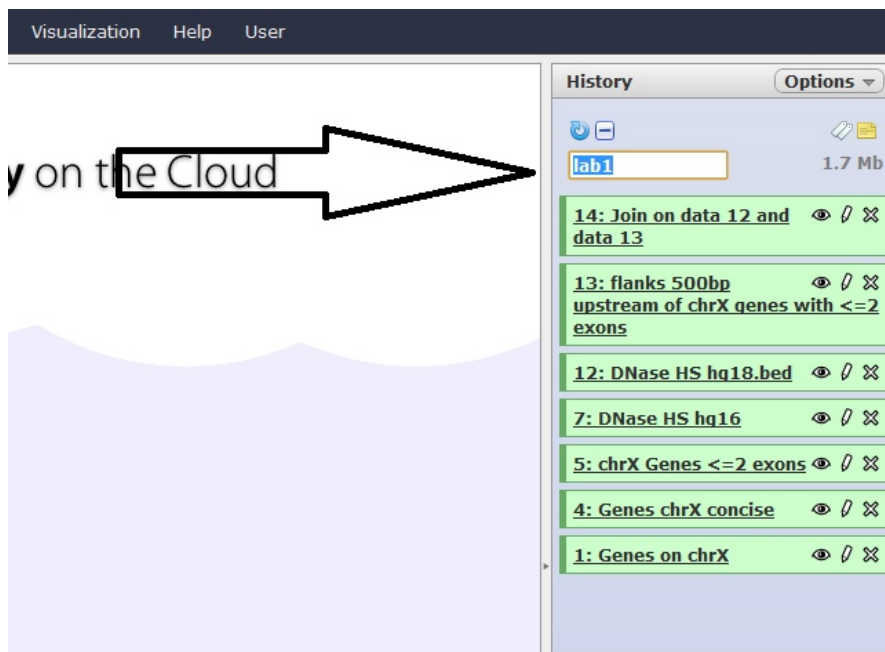
You’ll end up at some random position on chromosome X, but probably won’t see any of our displayed features. Enter the following coordinates to view under POSITION/SEARCH: chrX:38,542,873-38,547,882 and click JUMP.

Now you should see the view as in above. You can now see a track showing smallGenes, which shows where genes are present with at most 2 exons, and you can also see the location of DNase hypersensitive sites and the upstream regions of our small genes.

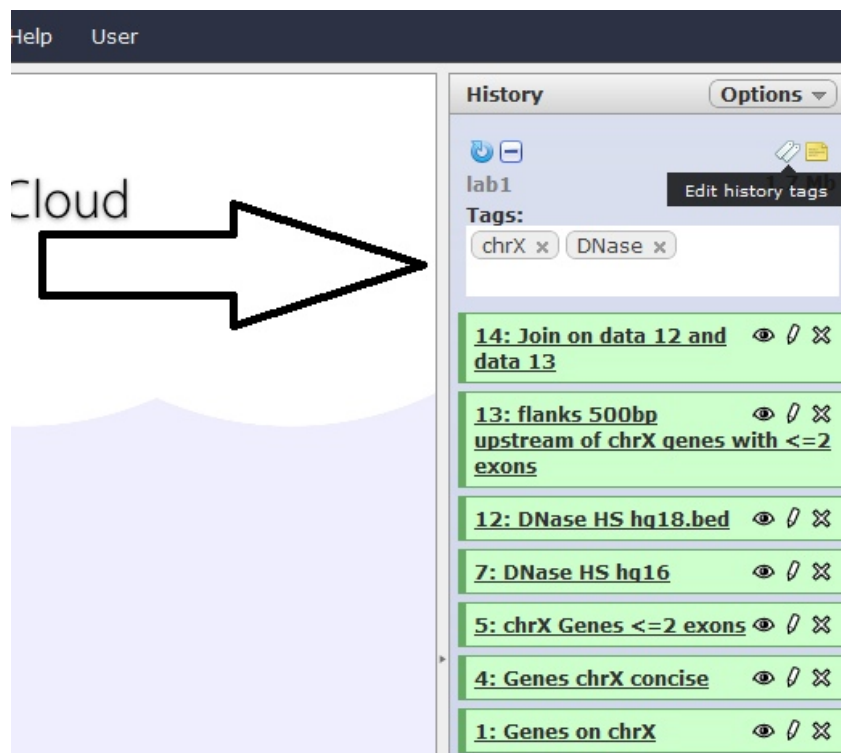
Go ahead an experiment with the genome browser and see what it has to offer. You can zoom in and zoom out with the buttons on top. Try right-clicking on the grey area to the left of the smallGenes track (arrow in image above) and select FULL. This will display all of the genes and their variants on separate lines. You’ll notice that previously, this track was in DENSE mode which means that it displayed all of the information on a single line without showing the subtracks. Try the other display options and see what they look like. Again, we’ll be covering visualization in greater detail in future tutorials.

Working with Histories

Let's say we have finished our analysis. It is always a good idea to give your history a meaningful name.

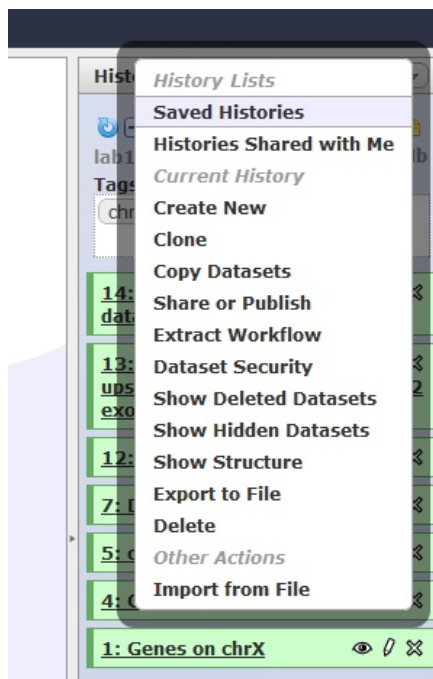


All you have to do is click above the history where it might say “Unnamed History” and type in something you like. You can also add tags by clicking on the white “tags” button to the right. Tags are short keywords that allow you to easily find histories in the future once you start getting swamped in your own histories. You can include something like “chrX”, “DNase”.



Remember how we said it's good to log in? Well if you click on OPTIONS above the history pane, you will see a long list of really useful options that you wouldn't have access to if you hadn't logged in. Here you can select to view all of your saved histories, convert a history to a workflow,

share your history with others, see deleted datasets and many other useful things. Click on **SAVED HISTORIES**.



This should bring you to a list that looks like this:

Saved Histories

search history names and tags
[Advanced Search](#)

<input type="checkbox"/>	Name	Datasets	Tags	Sharing	Size on Disk	Created	Last Updated ↑
<input type="checkbox"/>	lab1 ▾	7	2 Tags		1.7 Mb	~ 8 hours ago	37 minutes ago
For 0 selected histories: <input type="button" value="Rename"/> <input type="button" value="Delete"/> <input type="button" value="Delete and remove datasets from disk"/> <input type="button" value="Undelete"/>							

This will show you all of the histories that you have under your user account, the number of datasets per history, their tags and how large they are on disk together with other time stamp information. You can manage your histories here. Next to our only history name “labr”, there is a small arrow. If you click on it, you will get a menu with several options: Select **SHARE OR PUBLISH**.

Saved Histories

search history names and tags 🔍

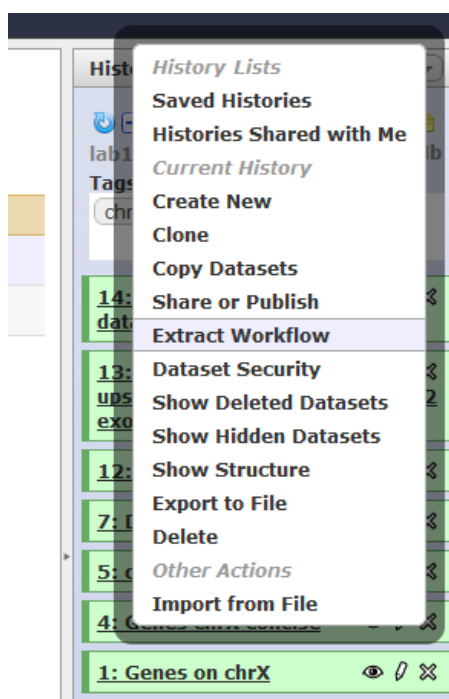
Advanced Search

Name	Datasets	Tags	Sharing	Size on Disk	Created	Last Updated ↑
Switch				1.7 Mb	~ 8 hours ago	37 minutes ago
<div style="border: 1px solid gray; padding: 5px; width: fit-content;"> <ul style="list-style-type: none"> Switch Share or Publish Rename Delete Delete and remove datasets from disk </div>						
			delete	Delete and remove datasets from disk	Undelete	

On the next screen you have a choice to publish this history via an internet link. You can make this link available to your colleagues and they'll be able to view it and use it. You can also choose to share your history via email, where you have to type in the email of your colleagues. They need to have Galaxy accounts under those emails for this to work. Once you do this, they will have these histories accessible through their own accounts. A great and transparent way to share your analysis pipeline with others, and this vastly promotes analysis reproducibility!

Working with Workflows (Optional)

Let's say you have refined your analysis pipeline and you plan on using this sequence of analysis tools and steps many times again on different datasets that you have. You can convert this history into a workflow easily where all you would have to do is ask the user for an input dataset and a set of input parameters, and he/she could run the history over and over again for different datasets without manually going through all the history steps. Let's make a workflow out of this history that we have on hand.



Click on **OPTIONS** in the history pane, and then click on **Extract Workflow**. You should see the screen on the next page. Here you can select which components of the history you want included in the workflow. Interactive components such as the **UCSC Table Browser** and manual file uploading are not supported in workflows, so we have to create special data input modules to replace them.

The following list contains each tool that was run to create the datasets in your current history. Please select those that you wish to include in the workflow. Tools which cannot be run interactively and thus cannot be incorporated into a workflow will be shown in gray.

Workflow name

Tool	History items created
UCSC Main <i>This tool cannot be used in workflows</i>	1: Genes on chrX <input checked="" type="checkbox"/> Treat as input dataset
Cut <input checked="" type="checkbox"/> Include "Cut" in workflow	4: Genes chrX concise
Filter <input checked="" type="checkbox"/> Include "Filter" in workflow	5: chrX Genes <=2 exons
UCSC Main <i>This tool cannot be used in workflows</i>	7: DNase HS hg16 <input checked="" type="checkbox"/> Treat as input dataset
Upload File <i>This tool cannot be used in workflows</i>	12: DNase HS hg18.bed <input checked="" type="checkbox"/> Treat as input dataset
Get flanks <input checked="" type="checkbox"/> Include "Get flanks" in workflow	13: flanks 500bp upstream of chrX genes with <=2 exons
Join <input checked="" type="checkbox"/> Include "Join" in workflow	14: Join on data 12 and data 13

Once you give this workflow a name and click on **CREATE WORKFLOW**, it should succeed and show you:



Workflow 'Workflow constructed from history 'lab1'' created from current history.

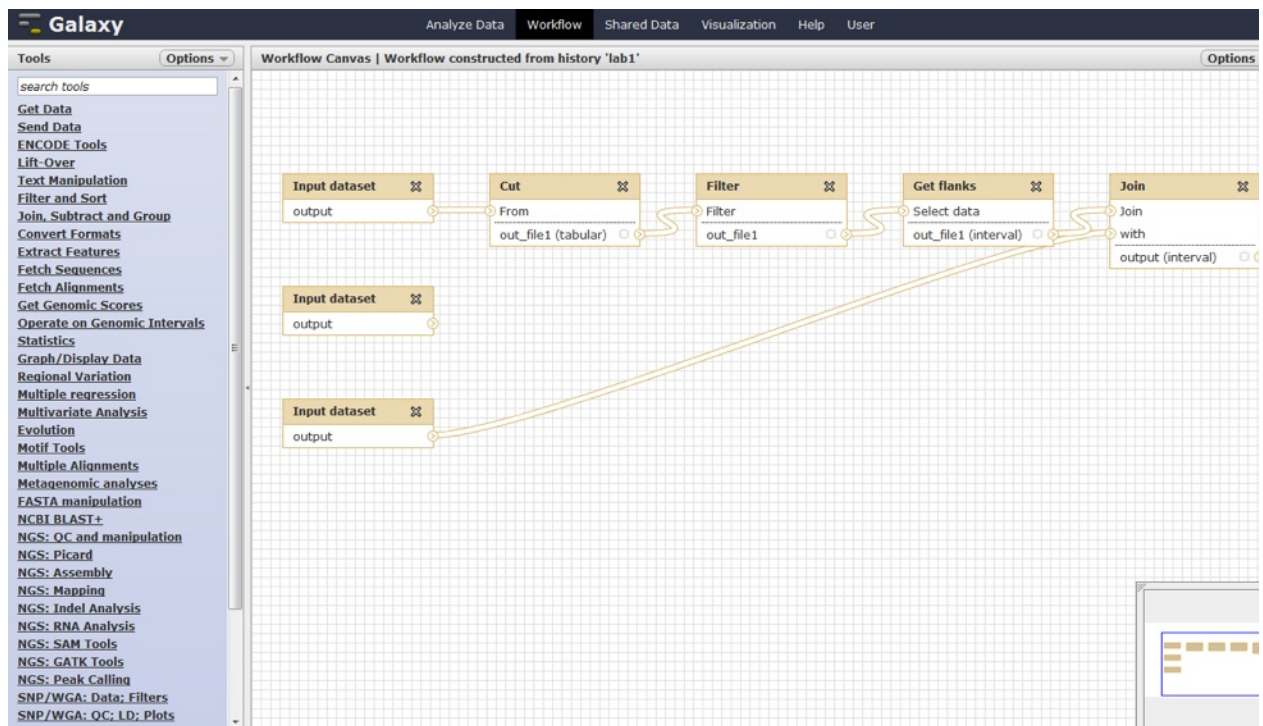
To find your workflows, click on **WORKFLOW** on the top menu in Galaxy. This will bring you to the workflow management page.

The screenshot shows the Galaxy web interface. At the top, there is a navigation bar with the Galaxy logo and links for 'Analyze Data', 'Workflow', 'Shared Data', 'Visualization', 'Help', and 'User'. Below this, the 'Your workflows' section is displayed. A table lists workflows, with the first entry being 'Workflow constructed from history 'lab1'' with 7 steps. A red box highlights the 'Name' column and a small downward arrow next to the workflow name. Below the table, there are sections for 'Workflows shared with you by others' (with a message 'No workflows have been shared with you.') and 'Other options' (with a button 'Configure your workflow menu').

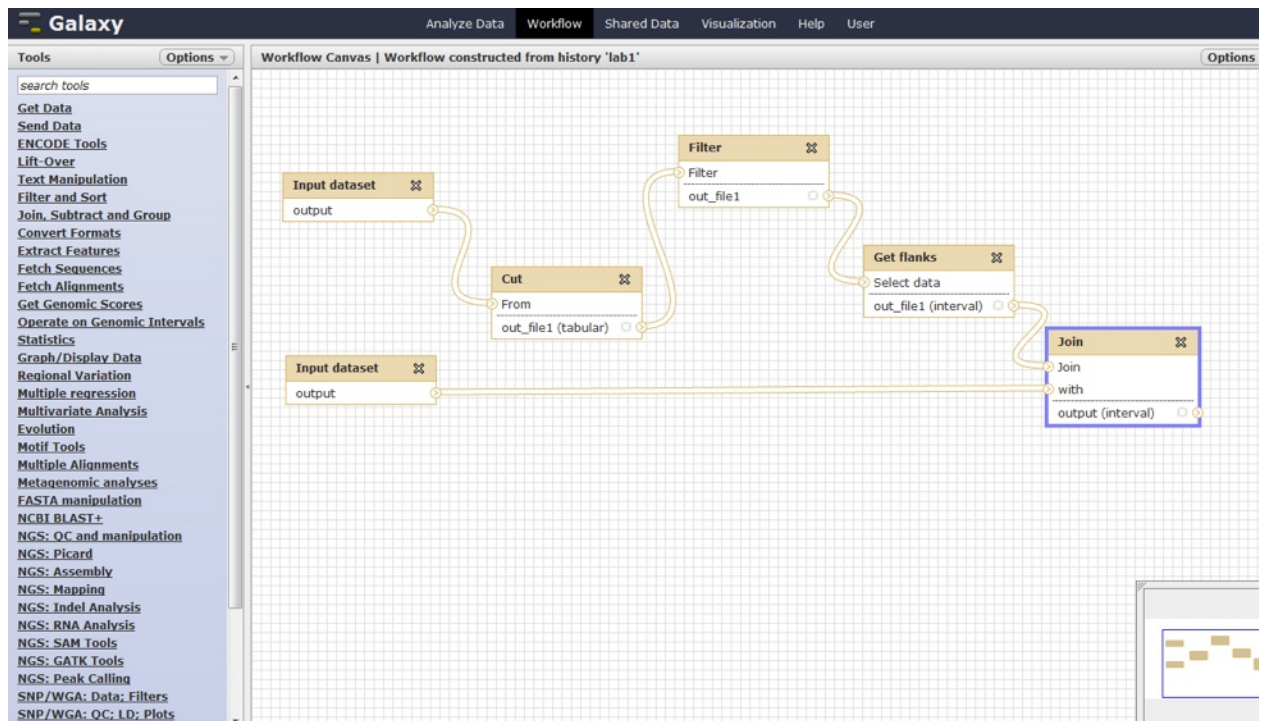
Let's take a look at our workflow. Click on the small arrow next to its name and click on EDIT.

This screenshot shows the same Galaxy interface as the previous one, but with a context menu open over the workflow name. The menu options are: Edit, Run, Share or Publish, Download or Export, Clone, Rename, and Delete. The 'Edit' option is highlighted in blue. The background shows the 'Your workflows' section with the workflow name 'Workflow constructed from history 'lab1'' and the 'Other options' section with the 'Configure your workflow menu' button.

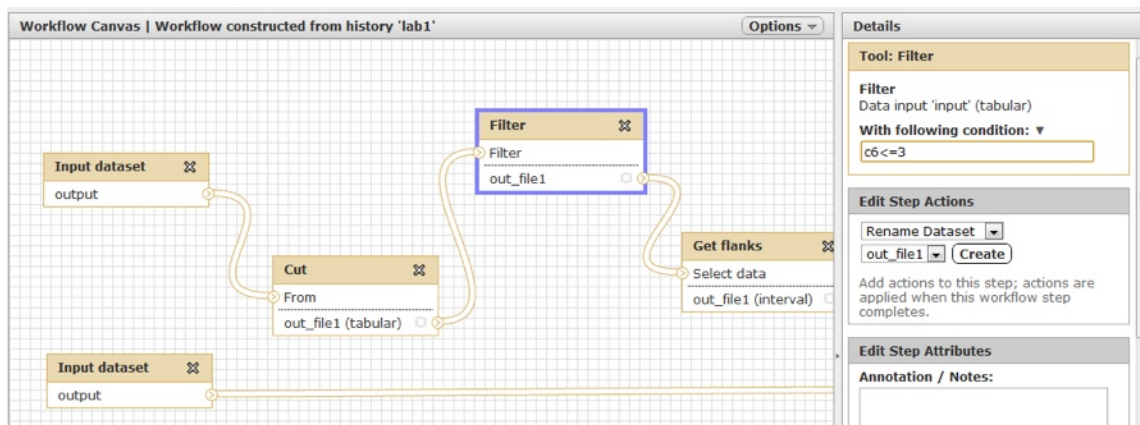
You can see the workflow design canvas with the workflow that we've just created from our history. Here you can design a workflow from scratch or edit an existing workflow. You can select any tool from the left pane and it will appear on the canvas. You can drag tools around by clicking and dragging your mouse over them. On the bottom right corner there is a "map" view which allows you to navigate the workflow. By clicking on the canvas itself or on the map, you can pan the view. You can click on tool boxes and modify their options.



Let's modify our workflow slightly to make it look better. First of all let's delete the INPUT DATASET box that isn't connected to anything. If you remember this is the dataset that we inputted from UCSC that was in the hg16 build which we did not use. To delete the box, click on the X in top right corner of the tool box. Then you can reorganize the other boxes so that their connections are more easily seen.



Now let's say that we want to select genes on chromosome X, but not those that have at most 2 exons, but those with at most 3 exons. Click on the FILTER box to bring up its options. We can then change the condition to "c6<=3".



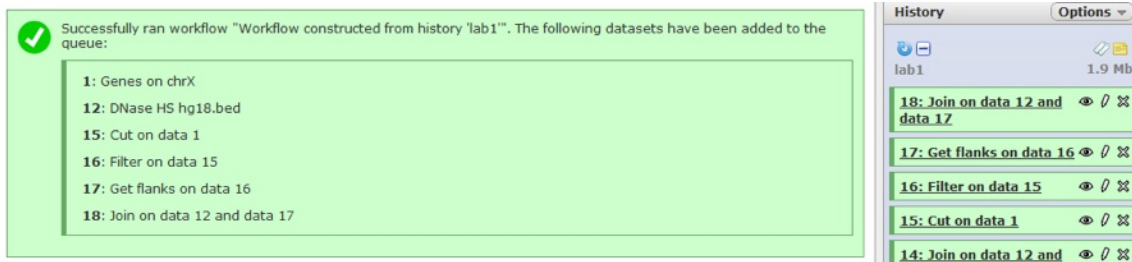
Also note that you can connect/disconnect the arrows between tool boxes. The arrows show data flow; data is fed out of modules as outputs and then fed into other modules as inputs. You could in theory build any workflow like this from scratch if you know what steps are required of the analysis to be performed. Let's rename the input modules to make it easier for the user to know what kind of input is expected there. Click on the top input dataset box, and under NAME call it "Input Genes" and call the bottom input dataset as "Input DNase sites". You can also type more information under ANNOTATION to make it clearer for the user what is required of them at this step. Now click on OPTIONS on top and SAVE.

Now let's run this workflow. If we were to run this workflow from scratch, we would first need to have all the required input datasets in our history. We already have them, so we can just run the workflow directly. Click on WORKFLOWS on the top menu. Then from the list of saved workflows, click on RUN next to the workflow name we have just created.

The screenshot shows the Galaxy interface with the following elements:

- Tools Panel (Left):** A list of available tools including Get Data, Send Data, ENCODE Tools, Lift-Over, Text Manipulation, Filter and Sort, Join, Subtract and Group, Convert Formats, Extract Features, Fetch Sequences, Fetch Alignments, Get Genomic Scores, Operate on Genomic Intervals, Statistics, Graph/Display Data, Regional Variation, Multiple regression, Multivariate Analysis, Evolution, Motif Tools, Multiple Alignments, Metagenomic analyses, FASTA manipulation, NGBI BLAST+, NGS: QC and manipulation, NGS: Picard, NGS: Assembly, NGS: Mapping, NGS: Indel Analysis, NGS: RNA Analysis, NGS: SAM Tools, NGS: GATK Tools, NGS: Peak Calling, SNP/WGA: Data: Filters, SNP/WGA: QC: LD: Plots, SNP/WGA: Statistical Models, and Human Genome Variation.
- Workflow Canvas (Center):** Titled "Running workflow 'Workflow constructed from history 'lab1'" with buttons for "Expand All" and "Collapse". It lists six steps:
 - Step 1: Input dataset:** Input Genes (1: Genes on chrX)
 - Step 2: Input dataset:** Input DNase HS sites (12: DNase HS hg18.bed)
 - Step 3: Cut**
 - Step 4: Filter**
 - Step 5: Get flanks**
 - Step 6: Join**
- Options:** Send results to a new history
- Action:** Run workflow (highlighted with a red star)
- History Panel (Right):** Shows a list of datasets in history, including:
 - lab1
 - 14: Join on data 12 and data 13
 - 13: flanks 500bp upstream of chrX genes with exons
 - 12: DNase HS hg18.bed
 - 7: DNase HS hg16
 - 5: chrX Genes <= 2 exons
 - 4: Genes chrX concise
 - 1: Genes on chrX

Make sure you select the proper datasets at the input dataset steps. That's all we need to do! Then click on **RUN WORKFLOW**. If nothing goes wrong, you should have all the steps run in sequence and finally complete with all steps becoming green as shown below.



Our output dataset is the last one. Let's click on the eye button and take a look at it.

chrX	15263581	15264081	-	uc010new.1	3	chrX	15263700	15263803	12_3	750
chrX	38545128	38545628	+	uc004dei.2	3	chrX	38545601	38545674	34_2	500
chrX	38545128	38545628	+	uc010ngz.1	2	chrX	38545601	38545674	34_2	500
chrX	41077094	41077594	+	uc004dfd.1	3	chrX	41077437	41077734	47_2	500
chrX	49530584	49531084	-	uc004doo.2	2	chrX	49530500	49531125	79_3	750
chrX	53127786	53128286	+	uc004drv.1	2	chrX	53128065	53128441	83_2	500
chrX	53466343	53466843	-	uc004dsi.1	3	chrX	53466161	53466383	84_3	750
chrX	70232250	70232750	+	uc010nkz.1	3	chrX	70232390	70232780	97_2	500
chrX	70232250	70232750	+	uc004dys.1	3	chrX	70232390	70232780	97_2	500
chrX	70232603	70233103	+	uc004dyt.1	3	chrX	70232390	70232780	97_2	500
chrX	70629024	70629524	-	uc004dzz.1	1	chrX	70629391	70629554	101_2	500
chrX	100549346	100549846	+	uc004ehm.1	2	chrX	100549292	100549706	110_3	750
chrX	100549346	100549846	+	uc004ehn.1	2	chrX	100549292	100549706	110_3	750
chrX	118776103	118776603	+	uc010nql.1	1	chrX	118776496	118776766	125_2	500
chrX	118889818	118890318	-	uc004esb.1	1	chrX	118889822	118889905	126_2	500
chrX	134013887	134014387	-	uc004eyf.1	1	chrX	134013907	134014564	144_3	750
chrX	134013798	134014298	-	uc004eyg.2	2	chrX	134013907	134014564	144_3	750
chrX	134305623	134306123	-	uc004eym.2	2	chrX	134305733	134305739	146_2	500
chrX	134305886	134306386	+	uc004eyr.2	3	chrX	134306304	134306879	147_3	750
chrX	153016382	153016882	-	uc004fjw.2	3	chrX	153016495	153017246	170_3	750
chrX	153360790	153361290	-	uc004fln.1	3	chrX	153360539	153360898	177_4	100
chrX	153428255	153428755	+	uc010nva.1	3	chrX	153427875	153428709	178_4	100
chrX	154147046	154147546	-	uc004fne.1	2	chrX	154146984	154147139	181_3	750
chrX	154341470	154341970	-	uc004fnj.1	1	chrX	154341170	154341526	182_2	500

Since in the workflow we selected to choose all genes with up to 3 exons, we now have more results than before (24 rows). We can also see in column 6 that we have values from 1 to 3 as expected. Just like we can share histories, we can share workflows. Try it! Go to the **WORKFLOWS** on the top menu and click on the **SHARE** options next to the workflow name.

Exercises

1. Create a new history and give it a name.
2. Download exon data for human chromosome 18 from the latest hg19 build from the UCSC table browser in BED format. Hint: you should first select to download UCSC genes as before, but there will be additional options to limit your download to only exons.
3. Download single nucleotide polymorphism (SNP) data for the same chromosome and same build from the same source in the same format. Hint: look under the "Variation and Repeats" group. Select Common SNPs (132). Rename your datasets!
4. How many exons are in your exon file? How many SNPs? (on chromosome 18 that is)

5. Join the two datasets together using the JOIN tool as we saw before. We want to place the exons (larger dataset) first (on the left) and the SNPs dataset next to it (on the right). Perform the join so that the genomic intervals in both files have a minimum overlap of 1 base and make sure the output file contains only those regions which have an overlap. What does this new dataset tell us and how is it different from the previous two? How many SNPs are there on exons on chromosome 18? Is this value different from the number of exons or the number of SNPs we saw before? Why? Hint: do any of the exon IDs in the joined dataset occur more than once?
6. Try the GROUP tool under the “Join, Subtract and Group” tool heading to create a new dataset which gets rid of multiple copies of exon IDs by grouping on them, as a result collapsing all of the other data into a single line per exon ID. While you are here at this tool, you can perform basic arithmetic operations on the grouped elements. Let’s count how many duplicate copies we originally had of each exon ID. How many exons contain SNPs on chromosome 18? Once you obtain this dataset, what does the second column with the values really mean? Hint: exon IDs are in the column after the END genomic coordinates.
7. Try the SORT tool under the “Filter and Sort” tool heading in order to sort this last file in descending order on the numeric column that we discussed in the previous section. What is the maximum (number of SNPs) that are present on any one exon?
8. Filter the previous dataset using the FILTER tool to only contain rows that have at least 5 in the numeric column. How many exons contain at least 5 SNPs each? What is the proportion of high SNP-density exons (with at least 5 SNPs) among all exons?
9. Rejoin this last dataset with our original exon dataset so that we now have a more complete exon dataset which also has counts of SNPs for each exon ID entry. Hint: use the JOIN tool and join by exon ID. Remember to place the larger dataset first (on the left) when performing the join.
10. Visualize this last dataset in the UCSC Genome Browser. Select one of the exons that Galaxy just submitted to UCSC genome browser and zoom in to see the SNPs in that exon.
11. Convert this history into a workflow that is designed to accept an input dataset with exons from a particular chromosome, and a second input dataset with SNPs on that chromosome. Change the workflow so that you end up with exons that have at least 7 SNPs each.
12. Run this workflow with the necessary data but this time from chromosome 21.

Second-Generation Sequencing

The second- and third-generation sequencing field moves so rapidly that it can be quite difficult to provide a current overview. Aside from the algorithmic problems, one of the main challenges that need to be overcome by the large sequencing facilities is data management. The following blog posts are worth exploring since the problem increasingly occurs in individual labs as sequencing becomes affordable even for smaller projects:

- <http://pathogenomics.bham.ac.uk/blog/2009/09/storage-on-a-budget/>
- <http://pathogenomics.bham.ac.uk/blog/2009/10/do-you-store-your-image-files/>
- <http://blog.bioteam.net/2009/10/16/storage-for-next-gen-sequencing/>

Initial quality control and basic analysis workflows are for the most part command line driven. Commercial systems like CLCbio (<http://www.clcbio.com/>) are still fairly rare outside of dedicated sequencing labs, but the publicly available software has matured enough to provide almost everyone with the ability to make sense of sequencing information:

- Quality control: FastQC, <http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/>
- Sequence manipulation: FastX Toolkit, http://hannonlab.cshl.edu/fastx_toolkit/
- Workflow generation: BioPieces, <http://code.google.com/p/biopieces/>
- Data comparison: BEDTools, <http://code.google.com/p/bedtools/>

This maturity in tool development is driven by a rapid convergence towards a small list of minimal standards in order to allow a more modular design of workflows as well as to facilitate data exchange between components:

- Sequence and quality information: FASTQ (<http://maq.sourceforge.net/fastq.shtml>)
- Alignments: SAM/BAM (<http://samtools.sourceforge.net/>)
- Variant calling: VCF (<http://vcftools.sourceforge.net/specs.html>)

These standards and other existing software frameworks facilitate the development of sequence analysis environments such as the Broad's Genome Analysis Toolkit (http://www.broadinstitute.org/gsa/wiki/index.php/The_Genome_Analysis_Toolkit), eventually even allowing non-programmers to mix and match their workflows as needed.

Applications

Any survey of applications for second-generation sequencing is bound to be outdated by the time it comes back from the printer. The easiest way to stay on top of new algorithms and workflows is closely monitoring the SeqAnswers forum (<http://seqanswers.com/>) and to browse through the accompanying Wiki (<http://seqanswers.com/wiki/Software>). Finally, a number of high-impact webblogs are almost mandatory to watch. A somewhat arbitrary selection includes:

- MassGenomics, <http://www.massgenomics.org/>
- GeneticFuture, <http://scienceblogs.com/geneticfuture/>

- PoITigenomics, <http://www.politigenomics.com/>
- Genetic Inference, <http://www.genetic-inference.co.uk/blog/>
- Fejes, <http://blog.fejes.ca/>

Workflows

You cannot go wrong by simply following the workflows outlined by large-scale genomic papers coming from any of the big sequencing centers, although this frequently requires delving through the supplementary material and online information. A number of reviews are also helpful:

- Ashley et al. Clinical assessment incorporating a personal genome. *The Lancet* (2010) vol. 375 (9725) pp. 1525-1535
- Harismendy et al. Evaluation of next generation sequencing platforms for population targeted sequencing studies. *Genome Biol* (2009) vol. 10 (3) pp. R32
- Mardis and Wilson. Cancer genome sequencing: a review. *Human Molecular Genetics* (2009) vol. 18 (R2) pp. R163-8
- Robison. Application of second-generation sequencing to cancer genomics. *Briefings in Bioinformatics* (2010) pp.

For our course the special issue on exome sequencing just published by *Genome Biology* (<http://genomebiology.com/content/12/9>) is worth exploring. An excellent and current step-by-step workflow can also be found on the SeqAnswers wiki (http://seqanswers.com/wiki/How-to/exome_analysis).

Getting started

We will be focusing on the application and combination of various sequencing tools within the Galaxy Framework (<http://usegalaxy.org>) which simplifies the analysis even further while providing data histories, access control, workflows and more. The course Galaxy instance can be found at <http://174.129.35.133/>. Please see the separate Galaxy handouts to get you started on using the system.

We will be retracing most of the steps required to get from a FASTQ sequence file (Illumina) received from a sequencing facility as part of an exome sequencing analysis all the way to variant calls and variant prioritization.

Quality Controls

We will start with the mandatory quality controls after receiving our sequence data from the core facility in FASTQ format. The FASTQ file contains output reads from the sequencer that need to be mapped to a reference genome for us to understand where those reads came from on the sequenced genome. However, before we can delve into read mapping, we first need to make sure that our preliminary data is of sufficiently high quality. This involves several steps:

1. Obtaining summary quality statistics on the reads and review diagnostic graphs
2. Eliminate sequencing artifacts
3. Filter out genetic contaminants (primers, vectors, adaptors)
4. Filter out low-quality reads
5. Recalculate quality statistics and review diagnostic plots on filtered data
6. Iterate through steps 2-5 until the data is of sufficient quality before proceeding to mapping.

Load the FASTQ file

1. Open up Galaxy from your machine as before (<http://174.129.35.133/>)
2. Start up a new blank history and give it a name.
3. In the top menu, move your mouse cursor over SHARED DATA and select DATA LIBRARIES. This allows you to import files into Galaxy that have been shared with you
4. You will be assigned to download a particular file. Browse to that file under folder “chr22” and mouse over the arrow to the right of the filename. In the context menu select “Download to Selected History”. On the next screen, select the history you are using. The file should now appear in your history.
5. Take a look at the file contents. Note it’s size. What differences do you see between the FASTA format and the FASTQ format? What additional information does FASTQ contain over FASTA?

Obtain Quality Statistics

1. Under the NGS:QC and manipulation tool heading, select the COMPUTE QUALITY STATISTICS tool and apply it to your FASTQ file.
2. Apply the Draw Quality Score Boxplot tool to your quality statistics file. What do you see? What does the boxplot tell you about the quality of your reads?
3. Apply the Draw Nucleotides Distribution Chart tool to your quality statistics file. What do you see? What does this chart tell you about the reads?
4. Try the FASTQ Summary Statistics tool under the ILLUMINA FASTQ tool subheading on your FASTQ data file. Do you observe similar quality statistics as before? What additional information do you get with this tool?
5. Compare the results with the pre-generated FastQC plots at : http://dl.dropbox.com/u/407047/Work/SequencingCourse/QA/7_100326_FC6107FAAXX-chr22_fastqc/fastqc_report.html OR http://dl.dropbox.com/u/407047/Work/SequencingCourse/QA/8_100326_FC6107FAAXX-chr22_fastqc/fastqc_report.html (or find the ‘FASTQC’ tool and generate your own report). Make sure you select the correct one depending on the FASTQ file that you have been assigned! What can you say about the per-base GC content? Does this match the human

genome? Why is there a shift in the GC distribution from the theoretical distribution? What is the meaning of the k-mer content tab?

6. Apply the COLLAPSE Sequences tool on your FASTQ dataset. What does this tool do?

Eliminate Sequencing Artifacts

Apply the Remove Sequencing Artifacts tool on your FASTQ file. How many artifact reads were eliminated?

Filtering Contaminants

1. Go on <http://www.ncbi.nlm.nih.gov/VecScreen/VecScreen.html> to visit the VecScreen online tool that allows you to determine which reads have a high probability of being of primer or vector source. (This section is for demonstrative purposes only)
2. Try the following test sequences:

> Test1

AATGATACGGCGACCACCGACAGGTTTCAGAGTTCTACAGTCCGA

>Test2

AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTC
TTCCGATCT

As you can see, not every tool is available through Galaxy (yet)! Some things need to be done outside of Galaxy using other online tools.

Filtering Out Low-Quality Reads

1. After reviewing the quality diagnostics in section 2, choose appropriate quality parameters you think would be good for this data.
2. Feel free to use the following tools to filter out low-quality reads from your data: Filter FASTQ or Filter By Quality. Choose appropriate values.

Quality Statistics on Filtered Data – Better?

Generate another quality report (see above). Do we have an improvement in read quality? What has changed and why?

Read Trimming and Wrap-Up

It is generally a good idea that once you have done all of the filtering steps that you trim your reads to cut away trailing read sections that are still of low-quality (if necessary). Is this necessary in your case? Use the TRIM SEQUENCES tool to finally cut away the read tails which are of low-quality

(if necessary). Make sure to save your finalized FASTQ file that has been filtered and improved. You will be using this file in the next section for mapping.

Exome Sequencing and Alignment

Next we are going to look at the steps we need to take once we have a clean, filtered FASTQ file that is ready for alignment. You can either use the filtered FASTQ file that you prepared in the previous section, or download a new one from the Shared Data Libraries in Galaxy. The alignment process consists of the following steps:

1. Choose an appropriate reference genome to map your reads against
2. Perform the read alignment using one of several alignment tools, Bowtie or BWA, creating an output SAM file
3. Convert the SAM file(s) to compressed BAM format
4. Generate BAM index statistics
5. Perform quality control on the exome enrichment using Picard, compare to external QC tools

Load the filtered FASTQ file and Reference Genome

1. Start up a new blank history and give it a name.
2. Either load your filtered FASTQ files from the previous tutorial, or download a filtered FASTQ file from the Shared Data Libraries in Galaxy. Make sure the Lane matches the one that you worked on in the previous tutorial.
3. To save disk space and computing time, we are going to be working with a reference genome (hg19) that only covers chromosome 22. Import this file into your history from the Shared Data Libraries.
4. Take a look at the file contents. Note their sizes. Calculate the GC content of the reference genome and the input FASTQ files using the GEECEE tool under the EMBOSS tool heading. Compare the GC contents of these two files. What do you conclude? Is there a difference? If so, why?

Perform Read Alignment

1. You have been assigned to run one specific read alignment tool (Bowtie or BWA). You can find these tools under the NGS: MAPPING tool heading. Make sure you select the corresponding tool for Illumina. Remember NOT to use a built-in reference genome. Select Single-End under “library mate pairing”. Look up the documentation to see what the various tool options do.
2. Perform the mapping using the default options.
3. Take a look at the output file. Note its size. How long did it take to run? Now extrapolate to how long you would expect this tool to run when mapping to the entire genome (approximately).

4. Take a look at the SAM format. What does it contain and how is the file structured?
5. If you have time, perform the mapping with the other alignment tool. How long did that take to run? Which one was faster? We'll be comparing the differences between these two alignments later on.

Convert SAM files to BAM files

1. Apply the SAM-to-BAM tool under the NGS:SAM TOOLS tool heading to convert your output SAM files to the compressed BAM format.
2. If you have both Bowtie and BWA created SAM files, pick one for the rest of the course.

Generate BAM Index Statistics

Calculate the index statistics using the BAM Index Statistics tool under the NGS: PICARD tool heading. What kinds of statistics are reported? What do they tell you about the alignment?

Quality controls of enrichment and alignment

We are now going to check how good our coverage of the target exome was. First, let's explore data visualization options within Galaxy using a test file. You will find a sample genomic bait region called 'hg19_bait_test.bed' in the visualization Shared Data folder.

1. First task, pull the bait test coordinates into the active history. What region/gene does the bait test overlap? This can be explored by pulling in all genes (or all genes on chr22) from the UCSC Table Browser into the active history and using the operations on genomic intervals to check what it intersects with.
2. The intersect tool depends on the order of input data. Try both orders and compare the different results. How do they differ?
3. Send the intersection result that includes a RefSeq gene identifier (NM_..) to UCSC. What is the name of the gene that the bait region overlaps? How would this have been different if we'd asked for one BED entry per exon instead of requesting the gene-based information from UCSC?

Quality Control of the exome capturing step

We will now explore the full target/bait information to your sequence alignment to get a sense of target enrichment, coverage and initial variation information.

1. Pull the overall target/bait BED files from the shared data library. The targets file contains genomic intervals of exons that we are aiming to capture. The baits file contains genomic intervals of those genomic regions that we practically can capture using our technology. Explore

- the target BED file and the bait BED file manually. How do the coordinates relate to each other?
2. Send the full target/bait BED file to UCSC for visualization (via the 'display at...' link in the details of the history panel). Get a sense of how well the targets align with genes/exons.
 3. Run the SAM/BAM Hybrid Selection Metrics tool under the NGS: PICARD tool heading on your BAM file.
 4. What do the metrics tell you about the efficiency and coverage of our exome capture experiment?
 5. A PDF of the exome sequencing quality control is available at <http://dl.dropbox.com/u/407047/Work/SequencingCourse/Presentations/exomeControl.pdf>. Work through it and pick two regions from the listed table, one with high coverage, one with low coverage. Identify the genes overlapping them via UCSC– what are they?
 6. Explore the data and coverage by sending the BAM file to UCSC. The viewer should remember the bait and target region information you sent previously. How uniform is the coverage in general? Can you spot regions where there are variants (red lines)? Inspect a single read (click on it) and explore the provided information. What level of detail can you obtain?
 7. Revisit the bait test region. Is there a variant in this gene based on the BAM file(s) you used? Any additional notes? What kind of variant is it?

Calling Variants

During this session you will take one of your BAM alignment files, call variants, check for basic annotation and compare the called variants to the original alignment using the Broad's Integrated Genome Viewer, IGV. Start by pulling in either a previously generated BAM file or one of the BAM files in the Visualization folder into your history; we will also need the chr22.fa reference again.

The Unified Genotyper (in the GATK tools section) needs additional information to be able to call variants. Set these via the 'Add or replace groups on data' from the Picard section. This includes giving the data a unique identifier, providing information on the sequencing technology, and filling out the remaining fields. Take a look at the help section if you are curious.

Run the Unified Genotyper. Explore the logfile to get some basic information, then take a look at the actual results and try to make sense of one or two entries (i.e., learn about the information contained in an VCF file, <http://www.1000genomes.org/node/101>)

Annotating Variants

You will be using a number of additional files that contain information about known variants (dbSNP and data from the HapMap project in VCF format). Data is in the Reference folder and was retrieved from the GATK resource bundle, http://www.broadinstitute.org/gsa/wiki/index.php/GATK_resource_bundle.

1. Pull in the dbSNP variant information (just the site information) from the reference library.
2. To make the processing easier filter for chr22 only using the 'Filter' tool from the Filter and Sort section
3. Use the VCFTools' Annotate function to merge your filtered chr22 dbSNP information with the GATK output. Check the new VCF file; the vast majority of variants now have an rs identifier in the 'ID' column. Look up one or two identifiers in the dbSNP database (<http://www.ncbi.nlm.nih.gov/projects/SNP/>) to get an idea of the kind of information that is available.
4. There is still a large number of SNPs left, even just for chr22 (how many?). Let's filter, only keeping the high confidence variants using the VCFTools 'filter'. We are using somewhat strict criteria this time around to keep the number of variants low; take a look at the GATK recommendations ([http://www.broadinstitute.org/gsa/wiki/index.php/Best Practice Variant Detection with the GATK v2#Making analysis ready calls SNP calls with hard filtering](http://www.broadinstitute.org/gsa/wiki/index.php/Best_Practice_Variant_Detection_with_the_GATK_v2#Making_analysis_ready_calls_SNP_calls_with_hard_filtering)) for more information.
5. For now, use the 'Filter' method from VCFTools and set: Filter by Quality 60, Filter 'QD 5 lt', Filter 'HRun 5 gt' (what do those terms mean?)
6. The 'Filter' tool only adds a new column to the VCF file that lists whether a given variant failed or passed the criteria. We only want to keep those which passed, so use the generic Filter tool to keep lines where column 7 states 'PASS'. How many are left?
7. Assume we are only interested in 'novel' mutations. Repeat the filter, this time removing anything with an rs identifier (a known dbSNP mutation) in the 'REF' column (column 3). How many novel variants are there? Write down one or two positions for later.

Comparing SNP calls and alignments

Let's visualize the data. Send the original BAM alignment file (imported right at the beginning) to IGV ('web'). Unfortunately, there currently is no way to send VCF files to IGV directly, so you will have to either download your final VCF file (or a VCF file from any stage) or use one of the URLs below (load from URL):

http://s3.amazonaws.com/hbc_projects/Sequencing_Course/all_variants.vcf

http://s3.amazonaws.com/hbc_projects/Sequencing_Course/novel_variants.vcf

After IGV starts:

1. Load the ENSEMBL genes (via the menu, load from server, use hg19).
2. Send your bait file to IGV. How does the test bait region compare between IGV and UCSC?
3. Take a look at the gene 'BCL2L13'. Explore the coverage at the first exon. Go further upstream to 'ATP6V1E1' – what happened to this gene?
4. Import the VCF files you are interested in (suggested: all variants called, novel variants) and index them (using File -> Run IGVTools -> Select file, set selector to index, run), then load

them into VCF (load from file). The VCF files available via the S₃ URL have been indexed already.

5. Explore the display. Check for good matches between variant calls and the reads. Try to find homozygous and heterozygous variants.
6. Explore one or two of the novel variants you wrote down. Are they real? If they have poor support, i.e., only 2-3 reads, go back to the VCF file and find novel entries with a reasonably high number in the 'DP' column, then go back to IGV and inspect the position.

Prioritizing variants

A vast number of algorithms exists to quantify the likely impact of a genetic variant. We will explore just a few options.

Predicting SNP effects

To simplify the analysis we will be using an aggregator, i.e., a tool that combines multiple annotation services, which accepts VCF-formatted data as input. Upload the 'novel' variants VCF file to:

<http://useast.ensembl.org/info/docs/variation/vep/index.html#web>

and decide on the annotation you want. Make sure to include SIFT, Polyphen and Condel. In the output, check for deleterious/damaging non-synonymous mutations. Do all predictors agree on the severity of a change? Pick one mutation and review it in IGV -- what is its coverage? What kind of change is the variant causing?

Additional tools can be used to re-prioritize SNPs. Most have been developed to gain additional information from GWA results by exploring variants beyond those reaching genome wide significance, but many can be adapted to combine candidate regions or genes from other data sources.

GRAIL

Grail uses text mining approaches, that is, information obtained from analyzing PubMed abstracts, to link candidate genes. Use the 'sampleSNPs.txt' file (provided) and visit the GRAIL website at:

<http://www.broadinstitute.org/mpg/grail/grail.php>

Submit the SNPs and select parameters as follows: hgr8, CEU population, Pubmed 2011, gene size correction is requested, query all genes, seed genes equal query genes. While the system is running (this may take up to an hour) take a look at the query set regions. Why are some dbSNP identifiers associated with one gene whereas others are linked to a dozen genes? What gene region did rs2395175 hit?

DAPPLE

While GRAIL is running obtain the 'sampleRegions.txt' file (provided) and submit it to the DAPPLE system at:

<http://www.broadinstitute.org/mpg/dapple/dapple.php#>

Make sure to state that this is a 'test run' with only 50 iterations, ask for a plot to be returned, and set 'nearest gene' to false. Similar to GRAIL you can set 'seed' genes -- genes that are known to have an association with the disease of interest. Here we assume no prior knowledge and ask for links between all genes rather than just connections between query genes to seed genes.

Once the system mails you your results take a look at the generated PDF and the prioritized genes. Submit the genes to the GeneMania system at

<http://www.genemania.org/>

using default parameters. What functional enrichment is detected in the generated network? Once you get your GRAIL results, how does the functional enrichment detected in the PPI-based prioritization relate to the PubMed-generated ones? In the GRAIL results, why did CTLA4 get such a high score? What disease are you most likely dealing with?