Version control: `git` with it

Dave Messina
originally by Jessen Bredeson

v2 2014

What is version control and why is it important?

- The gist: keeping a history of our code development so we can recall previous versions of that code at a later date.
- Why do we care about the history of our code?
  1. Provides a safety net against breaking our code.
  2. Allows us to explore different ways to solve the same problem.
  3. Easier to collaborate writing/sharing code with others.
  4. It's the like a notebook, and the scripts are the methods.

`git` into the habit of `committing` your changes to the repo: creating a repo

- Creating a new `git` repository (only need to do this once for every repo)

```
git init <repo_name>
```
← creates new directory if necessary

- Must `cd` into `git` repo directory to run `git` commands.
- We can check the current status of the repo (and its "tracked" files)

```
## Create a file in our repository, then:
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   myscript.pl
nothing added to commit but untracked files present (use "git add" to track)
```

`git` gives us useful hints for what to do next!

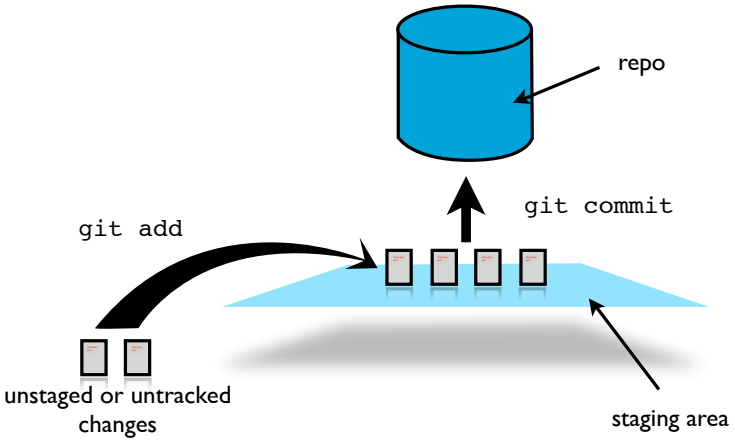
- Files have "tracked" status if they have been committed (more on what this means on the next slide). They are said to be "untracked" if the files exist within the repo directory, but have not yet been committed.

`git` into the habit of `committing` your changes to the repo: `add` and `commit` overview

- `git` uses an `add` and `commit` framework for saving code changes to the repo.

```
git add file1.pl [file2.pl [...]]
git commit -m "useful description of changes"
```

- `adding` adds new files, or changes in existing files, to a "staging area" where they wait to be `committed` *en masse*.
- `committing` saves all of the `added` changes to the repo.
- A `commit` message (`-m "message"`) is mandatory.



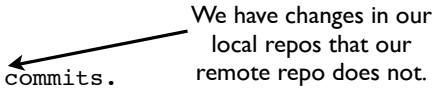
Remote repositories

- Why would we want a remote repository?
  - It's our repo's backup (exact copy).
  - Easiest way to share code with collaborators.
  - Easiest way to copy code between machines (local-local, local-server, etc).

- Free cloud-based remote repositories:
  - *github.com*
    - Free GUI repo management client
    - Unlimited number of collaborators
    - Unlimited public repos
    - Repos must be public (no private repos)
    - Wikis, issue tracking, project front pages
  - *bitbucket.org*
    - compatible with both `git` and `mercurial`
    - Free GUI repo management client
    - Unlimited private repos
    - Unlimited public repos
    - Small collaboration teams only (unlimited when registered with academic email)
    - Wiki, JIRA integration (issue tracking)

Syncing with remote repositories

- If we also have a remote repository (at github, bitbucket, etc.), we occasionally want to sync the histories between the two repos. This is called `pushing`, but before we do that, it is wise to `pull` down any changes that may have been made by someone else or that we made on a different machine.




```
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commits.
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   myscript.pl
#
no changes added to commit (use "git add" and/or "git commit -a")
$ git pull
Already up-to-date.
$ git push
Counting objects: 1, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (1/1), done.
Writing objects: 100% (1/1), 1.36 KiB | 0 bytes/s, done.
Total 1 (delta 1), reused 0 (delta 0)
To git@bitbucket.org:username/repo.git
   25f9c25..c5b98ce  master -> master
```

`git` into the habit of `committing` your changes to the repo: `add` and `commit` in detail

We `git add` the script, and now it is staged and waiting to be committed.

Another useful hint: If we want to remove the untracked script from the staging area.

```
$ git add myscript.pl
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   myscript.pl

$ git commit -m 'new script "myscript.pl"'
[master (root-commit) 0c7f52c] new script "myscript.pl"
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 myscript.pl

$ git status
# On branch master
nothing to commit, working directory clean

# sync with remote repository
$ git push origin master
```

Summary of changes made to all files.

All changes `committed` and no untracked files.

Comparing version differences

- Look at the differences between your latest version and what is in the repo.
- Useful if we want to review the changes we have made.

`git diff [file_name]` ← if the code is added to the staging area.

`git diff --cached [file_name]` ← if the code is `committed`.

```
## Make some changes to our code, then:
$ git diff myscript.pl
diff --git a/myscript.pl b/myscript.pl
index e69de29..1f1c293 100644
--- a/myscript.pl
+++ b/myscript.pl
@@ -0,0 +1 @@
+#!/usr/bin/perl
```

`@@ -0,0 +1 @@` ← one line inserted, relative to repo.

`+#!/usr/bin/perl` ← inserted lines will be green, deletions red.

Unstaging additions to staging area

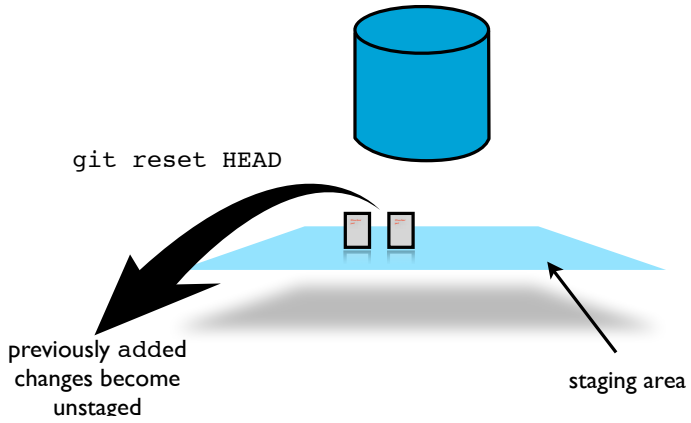
- If we decide we do not want to `commit` the changes added to the staging area, we can remove those changes from the staging area.

```
$ git add myscript.pl
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   myscript.pl
#

$ git reset HEAD myscript.pl
Unstaged changes after reset:
M   myscript.pl
```

`git` reminds you how to do it.

Indicates the file has been modified, but changes have not been added or `committed`.



`git commit` history

- We may also examine the history of our changes.

```
$ git add myscript.pl
$ git commit -m 'my useful message'
$ git log
commit 6b0063cba5f0de7b74fcc39357d0bd708a6c3f68
Author: username <youremail@host.com>
Date:   Tue Oct 15 21:40:01 2013 -0400

    added shebang line to script

commit 0c7f52c6b231f42e9100a1809a56aa2b2618bdf1
Author: username <youremail@host.com>
Date:   Tue Oct 15 20:57:01 2013 -0400

    new script "myscript.pl"
```

Commits stored as SHA1 checksums; very secure.

`git` records the relevant meta-data on your commits.

- Two months from now, you may want to pick back up on a project you started, but need to remember where you were in development; descriptive commit messages make your life easier!

undoing mistakes

- Sometimes we experiment with writing new features into our code, and if we discover that these changes introduced bugs, or we just couldn't get the feature to work, we can recover the version of the code that did work.

```
# find the checksum of the last commit
$ git log | head -1
commit 6b0063cba5f0de7b74fcc39357d0bd708a6c3f68

# revert that commit
$ git revert 6b0063cba5f0de7b74fcc39357d0bd708a6c3f68

[master 6a72a1f] Revert "adding some comments"
 1 file changed, 3 deletions(-)

$ git log
commit 6a72a1f03debd05945cf7a8200f82aa92145a0cf
Author: Dave Messina
Date:   Fri Oct 17 15:26:06 2014 -0400

    Revert "adding some comments"

    This reverts commit aa6b055a6794fe339ed33f4d23ace511eec25050.
```