DRAFT VERSION JANUARY 8, 2013
Preprint typeset using LaTeX style emulateapj v. 11/10/09

GIT COMMANDS

IAN CZEKALA

1. INTRO

Git only tracks changes to files, not storing the actual file.

Good for simple stuff, programming is an iterative process.

`man git-commit` and other flags have a wealth of information.

It's good to keep stable versions while developing. When developing a feature, keep a separate branch.

There are various diff programs you can use. kdiff? See Archwiki. git config –global core.diff opendiff (for mac) will show the interactive changes. Familiarize yourself witha diff program. For example, to compare commits, you can do git diff (hash1) (hash2) or alternativel git diff HEAD 1 HEAD 2 (these are the changes from 1 to 2).

This will also work by passing the two branches to git diff. `git diff feature-branch master` git config –global core.diff opendiff (for mac) will show the interactive changes. Familiarize yourself witha diff program. For example, to compare commits, you can do git diff (hash1) (hash2) or alternativel git diff HEAD 1 HEAD 2 (these are the changes from 1 to 2).

This will also work by passing the two branches to git diff. `git diff feature-branch master` git directories are completely self-contained within a directory, so when you want to tar up a file you can do the whole thing. So, this also works great for synching too, Dropbox or rsync.

GUI's gitk is a nice graphical representation.

2. STARTING NEW PROJECTS

`git init` to create a new `.git` directory.

`git clone url` to clone a new project (located at `url`) to your current directory.

3. SNAPSHOTTING YOUR PROJECT

DE: purpose of the staging area? You can incrementally stage what you want to commit, besides commiting everything at once.

`git add file1 file2...` is used to add files to your *staging* area before you *commit* them to the project. Even if the file was tracked before, you still need to add the new files before proceeding, if you want their changes committed. You will need to do `add` after each change. Or, simply use `git commit -a` to ignore having to do the add besides the initial add.

Probably will most commonly use `git add .`

`git status -s` used to list the status of all of the files that the project is aware of. The `-s` is for short output.

`git commit` starts *VIM* and then allows you to create a commit message. Only files which have been added are committed. However, if instead you use `git commit -a` then all previously tracked files are re-added and then committed. You would still need to use `git add file1` to add a new file to be tracked, however.

`git commit --amend` does not include all recent changes

`git commit --amend -a` will include all recent changes

`git commit -a` will track all changes regardless of whether they have been staged. This is generally a good practice to follow.

`git rebase` allows you to squash your commits down to one when they are trivial.

`git rm --chached file_name` will remove a file that you have added but not committed yet. Keeps file on disk.

4. BRANCHING AND MERGING

By default, there is the *master* branch. Generally, this is good to keep as the stable version.

`git branch` will list all the current branches. The * shows which current branch you are in.

`git branch branch_name` will create a new branch.

`git checkout branch_name` will *checkout* the new branch.

`git checkout commit_hash` will *checkout* a previous commit.

`git checkout -b branch_name` will create a new branch and switch to it immediately. This is a "copy" of the current branch.

`git branch -d branch_name` will delete an entire branch.

`git rebase -i master` apply changes to/from the (which branch?) since the two branches have brached. Interactive window.

Difference between rebase and merge? Allows you to pick which commits you want, and squashes trivial commits down to a single commit. Pass in branch that you want to pull branches from. Brian recommends using rebase -i over merge.

`git rebase --continue` to move forward with the synch

ALWAYS make a commit before checking out other branches. For example, if you delete a file in one branch and then checkout a different branch, the file will be deleted in the other branch. This essentially merges the commits together, if there are conflicts than the merge will not happen[1]

5. TRACKING

`git log` shows the commits in chronological order. Newest is at the top.

`git log --stat` shows the commits with all commit messages.

git log –pretty=oneline shows commits w/ title git log –pretty=full title + explanation git log –pretty=format:'

`git reset --hard HEAD~1` to go back to previous change after a commit (move backwards by one commit). `--hard` deletes the most recent change and resets the entire repository.

HEAD is where you are in the list of changes, a pointer to the latest change.

[1] `http://www.gitguys.com/topics/switching-branches-without-c`

Can also do this on individ files

`git reset -- analysis.py` resets to the last version `git reset HEAD~2 analysis`

6. DIFF

There are various diff programs you can use. kdiff? See Archwiki. git config –global core.diff opendiff (for mac) will show the interactive changes. Familiarize yourself witha diff program. For example, to compare commits, you can do

`git diff (hash1) (hash2)` or alternatively git diff HEAD 1 HEAD 2 (these are the changes from 1 to 2).

`git diff feature-branch master` compare two branches. If no text spits out, the most recent COMMITS within each branch are the same.

7. GIT WORKFLOW

Generally, create a new branch for every issue, bug, or feature you're trying to build or fix. That way, you can always have deployable code in the master branch.

`git stash` saves your current changes without commiting them and reverts back to the previous commit. Works for both staged or unstaged pages. Matches the HEAD commit.

`git stash pop` then opens this back up. Think of this as kind of `<C-z>` on the command line.

8. REMOTE

`git remote show origin` will help drastically with showing the status of your configuation with your remote repository.