Git is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed non-linear workflows

Use cases:

1. Keep a history of previous versions
2. Develop simultaneously on different branches
3. Easily try out new features, integrate them into production or throw them out
4. Collaborate with other developerrs
5. Push and pull code from hosted repositories such as Github

Istallation:

1. Linux: apt-get install git-core
2. Mac: <http://code.google.com/p/git-osx-isntaller/>
3. Windows: http:/msysgit.github.com/ git bash
4. Github GUI

First time setup

1. Git config --global user.name “name”
2. Git config --global user.email “email”
3. The email in 2 above most be registered in your github

Basic workflow

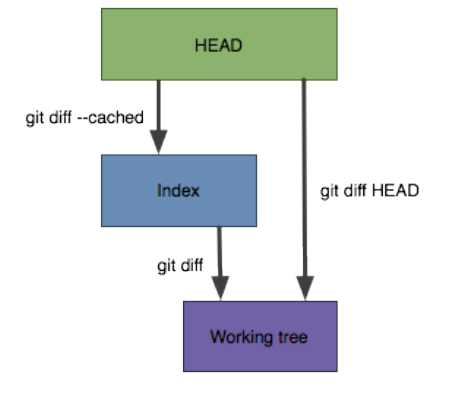
Add the file, edit the file, stage the file, commit

1. Git init: create git project in existing directory
2. Add files to be commited with git add <filename>
3. Create a commit (a snapshot) of added files with git commit, folllowed by a commit message git commit -m ‘’message’’
4. Use git status to see the current status of your workiung tree
5. The git log output. To terminate a git log operation, press “q” key

The staging area

1. Git add takes the snapshot of the file that will be committed. You can change the file after adding it
2. To unstage a file, but retain your changes in the working tree. Git reset HEAD <filename>
3. To discard current changes in the working tree, and make it look like the last commit: git checkout -- <filename>
4. Be careful, you wil lose your changes and not get them back
5. To remove a file from the working tree and in the next commit: git rm <filename>
6. To remove it from the next commit, but keep the file in the working tree, git rm --cached <filename>

Viewing difference of files



To exit from git diff, press “q”

Git diff shows only those changes of tracked files which are present in working directory. Git diff --cached shows only those changes of tracked files which are present in staging area

The .gitignore file

1. Specifies files that you don’t want git to track under version control
2. Commonly used for compiled files, binaries, large asset files e.g images
3. Can use wilcards e.g .pyc, .png, images/\* etc
4. Be carefule - if you add a file to .gitignore after its already been tracked, potential issues may arise
5. A list of recommended .gitignore files: check <https://github.com/github/gitignore>

Branching

1. What is a branch
2. Branching commands
3. The HEAD pointer
4. Basics of merging
5. Basics of rebasing
6. Aside: the git reset command

What is a branch

Visualize a projects devellpment as a linked list of commits, when a development track splits, a new branch is created, in Git, branches are actually just a pointer to these commits

Branching commands

1. git branch: list all branches in the project
2. Git branch <branchname>: create a new branch
3. Git checkout <branchname>: switch to a branch
4. Git checkout -b <branchname>: create and immediately switch to a new branch
5. Git branch -d <branchname>: delete a branch

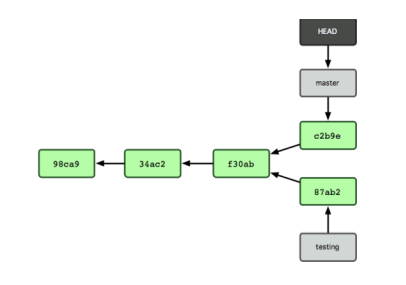
Git stashing

The git stash command takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy. Git stash is a super useful command that helps you save changes theat you are not yet ready to commit. You can stash changes and then come back to them later. Use git stash pop to reapply recently stashed changes. Git stash apply, allows you to take what ever changes that are in the stash and apply without removing them from the stash. Git stash list helps you view the list of stashes. Git stash clear clears the stash and git stash remove removes the stash.

The HEAD pointer

Recall: all branches simply are simply a pointer to a commit

HEAD: special pointer to the current branch, moves around as you switch branches



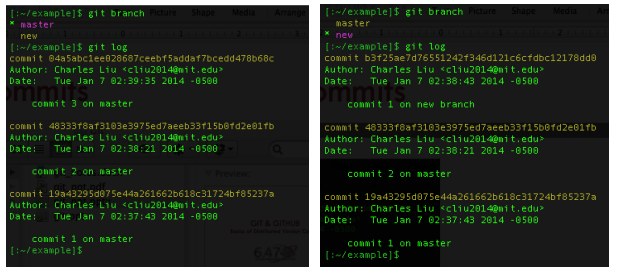
Merging

1. Git chechout master
2. Git merge feature

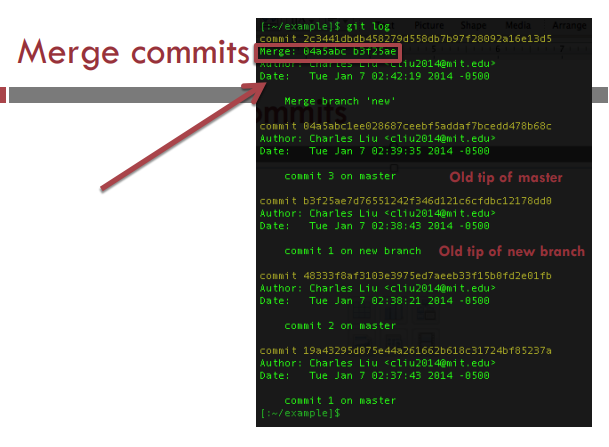
Note: merge creates a new commit with 2 parents

Merge commits

Before merge.. master and new\_branch have diverged (commit 3 on master vs commit 1 on new branch)



Merge commits



Merge conflits

1. Sometimes, two branches will edit the same piece of code in different ways
2. Must resolve the conflict manually, then add the conflicting files and explicitly commit.