Intro to Git

DSC 80 -- Aaron Fraenkel

Reference material:
https://www.slideshare.net/HubSpot/git-101-git-and-github-for-beginners

First, the terminal...

- This tutorial will focus on using the terminal for git usage.
- There are Desktop GUI tools for using git (not suggested).
- We will be focusing on Bash specifically (OSX, Unix)
  - Good for everyone to know!
- Windows is different, with similar syntax and concepts.

The Terminal

- The terminal lets you work with files on your computer.
- You can run commands on files
  - Entered on a prompt and printed below
  - Commands are terse for historical reasons
  - File are strings separated by slashes:
    - `/Users/afraenkel/Desktop/my_special_file.txt`
- Special file names:
  - Current Directory: `'.'`
  - Directory up one level: `'..'`
  - Root directory: `'/'`
  - Home directory: `'~'`

Basic terminal commands

• **cat** *file*	**Concatenate or type out a file**
cat *file1 file2* **...**	**Type out a number of files**
• **cd** *directory1*	**Change current directory to** *directory1*
cd /usr/bin	**Change current directory to /usr/bin**
cd	**Change back to your home directory**
• **clear**	**Clear the current screen**
• **cp** *file1 file2*	**Copy** *file1* **to** *file2*
cp *file1 file2* **...** *dir*	**Copy a number of files to a directory**
• **ls**	**List the files in the current directory**
ls /usr/bin	**List the files in the /usr/bin directory**
• **lpr** *file1*	**Print** *file1* **out**
lpr *file1 file2* **...**	**Print a number of files out**
• **more** *file*	**Look at the content of a file with paging, use 'q' to get out**
• **mkdir** *directory*	**Create a directory**
• **mv** *file1 file2*	**Move** *file1* **to** *file2*, **like rename.**
mv *file1 file2* **...** *dir*	**Move a number of files into a directory**
mv *dir1 dir2*	**Move or rename a directory**

Git Version Control System

What is Version Control?

- Keeps records of your changes
- Allows for collaborative development
- Allows you to revert any changes!

What is Git?

- Distributed Version Control
- Code history is kept on your computer (local git)
- Can keep code synced on remote servers (e.g. github)

Git is Pervasive

Git is popular and good to learn

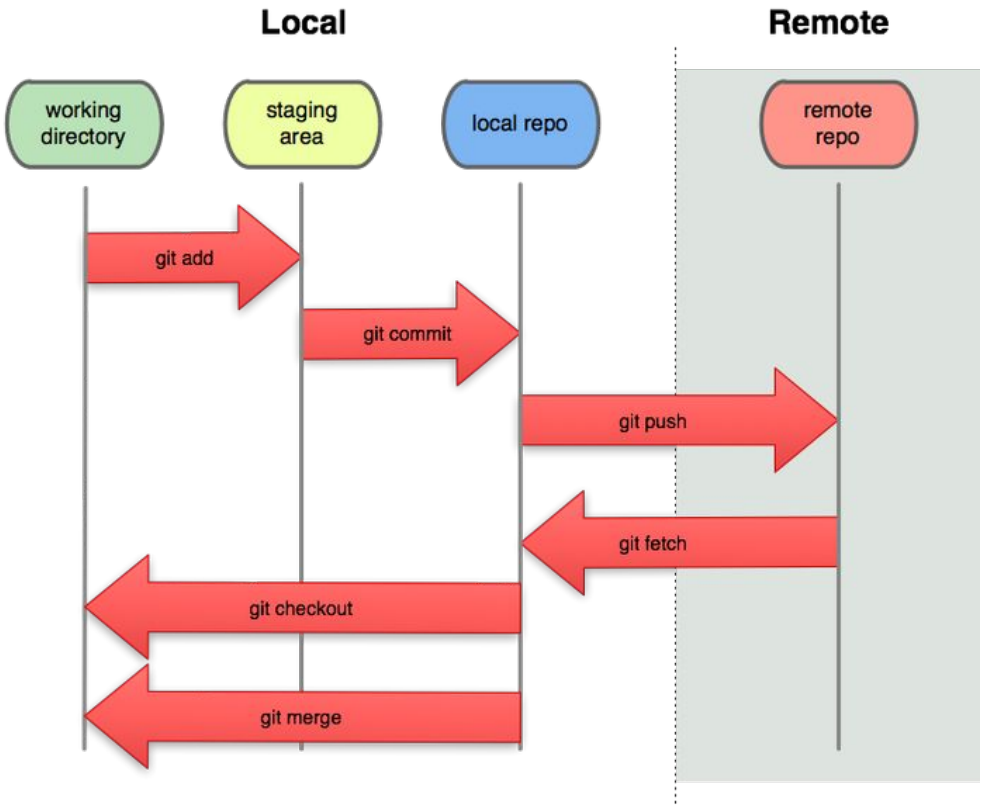
- Git is used to version the course website
- ~50% of software developers version control with git

Git isn’t the only version control system

- **Mercurial (Hg)**
- **Subversion (svn)**
- **Perforce**

How does git work?

- What are the key concepts?
- What's all this terminology?
- We will start **very** simple and build understanding through the quarter!



“Snapshots”

- Git keeps track of your code history through snapshots!
- Records what you files look like at a give time.
- You decide when to take a snapshot (“save”).
- You can revisit old snapshots.
- Git only keeps *differences between* the snapshots!

“Commit”

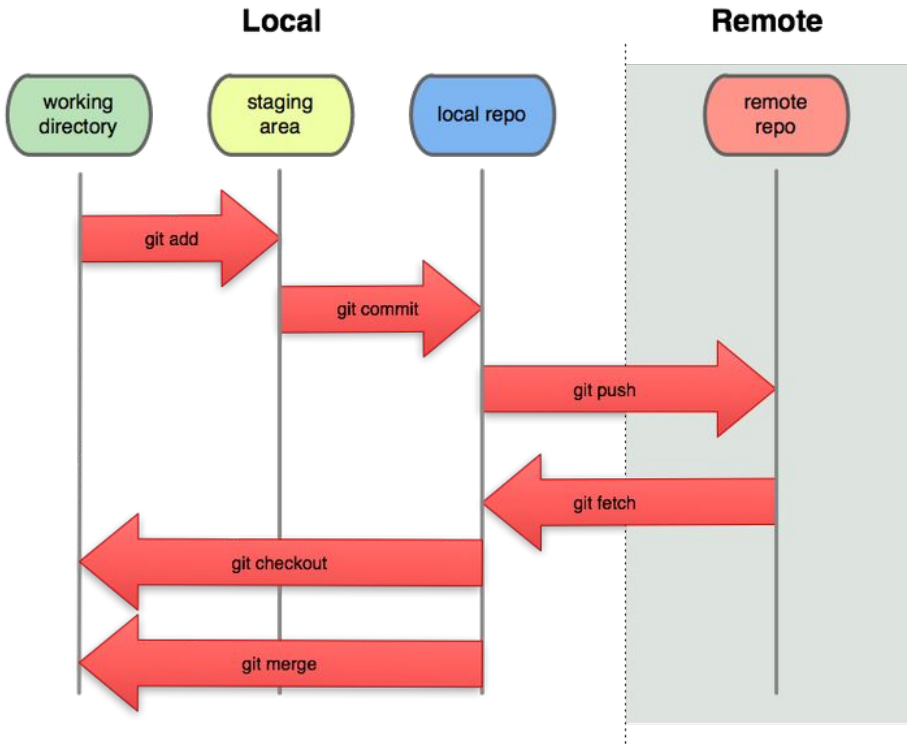
- The act of creating a snapshot.
- Roughly translates to “saving a file” in git
- A project is made up of commits.
- A commit consists of:
  - Information of how the files changed from previous commit
  - A reference to the previous commit
  - A unique identifier (hash code)
- Command: `git commit -m “commit message”`

Repositories (“Repo”)

- A collection of files and their history.
- A repo is a collection of commits.
- Can live on a local machine or a remote server (e.g. github)
- The act of copying a repository from a remote server is called **Cloning**
  - Command: `git clone <repository url>`

Repositories: local vs remote

- The process of downloading commits that don’t exist on your machine from a remote repository is called **fetching** changes.
- Related to fetch is **pull**.
- The process of adding your local changes to a remote repository is called **pushing** changes.

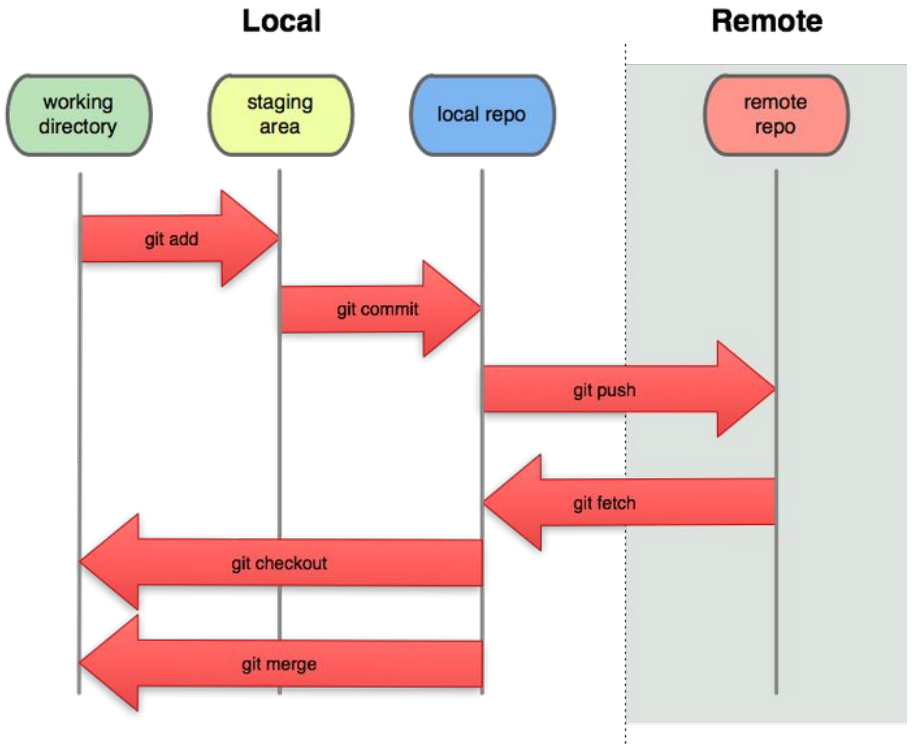


“Branches”

- Branches help you keep different work separate.
- Multiple people working on the same code should always use different different branches!
- One person working on multiple tasks in a single repository should use different branches for each task.
- The main branch is called the **master** branch; it’s created automatically.
- We will only use master to start.

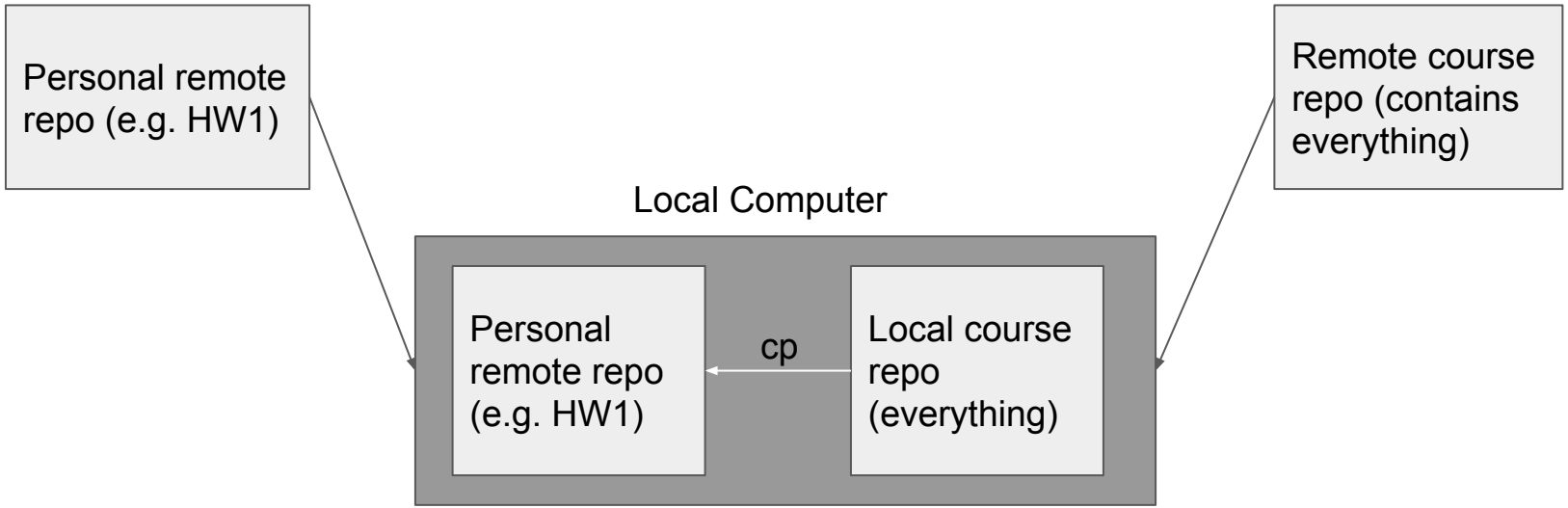
Working vs Staging vs Snapshot

- The **working directory** are the files that you are currently working on.
- The **staging area** are files that are being tracked by git, and ready to commit. You must add a file to staging to commit the file.
- A **snapshot** is the state of the staging area when you commit your work.



How does this apply to DSC 80?

- Course repository is a remote repo on github:
  - https://github.com/ucsd-ets/dsc80-wi19/
- You keep your HW code in *your own* private github repository.



Step I: Set-up your repository

- See the class tutorial for details.
- Open the terminal and go to where you want to work:
  - `cd <working dir>`
- Clone class repository (creates directory dsc80-wi19):
  - `git clone https://github.com/ucsd-ets/dsc80-wi19.git`
  - You now have all the class info ready to consume on your computer!
  - Treat this as read only! Don't make changes to files you want to keep.
  - In the future, you can pull changes to the class repo here!
- Create a repository for your first assignment:
  - Go to GitHub page and create a repository named, e.g. dsc80-wi19-hw01-<username>
  - You will have a new repo for each new assignment
  - See the class tutorial for more details.

Step II: Move content to your repo

- Create an assignments dir and copy HW01 into the directory:
  - `mkdir assignments`
  - `cp -R dsc80-wi19/hw/hw01 assignments`
  - `cd assignments/hw01`
- Follow the github instructions on initializing a new repo:
  - `git init`
  - `git add .`
  - `git commit -m “first commit”`
  - `git remote add origin https://github.com/username/…`
  - `git push origin master`
- Look at the material pushed to your github!
- Start working

Step III: And you work away...

- I finish question 1 and want to save my work.
  - `git status  # checks what changes haven’t been committed.`
  - `git diff    # shows the lines that have been changed`
  - `git add hw01.py  # stages the file hw01.py to be commited`
  - `git commit -m “question 1 done”  # commit the work.`
  - `git push origin master     # push your changes to github`
- Look at you changes on github; make your work available from anywhere.
- Commit often, to chunk up your progress
  - Easier to revert a change!

Step IV: You've started working…

- … and there's a fix to the homework.
- How do you incorporate the fixes?

Steps:

- Commit your current work on the HW! (see last slide).
- Pull changes to course repo (in the course repo directory)
- Copy the updated hw01 directory into your assignment directory
  - It will look like you are clobbering all your work! Make sure you commit before doing this!
- Use "line-by-line" commits to make only the changes you want!
  - Read the stackoverflow post to understand the options!
- Commit only the changes to the prompts/tests; discard the rest.

Step IV: The commands.

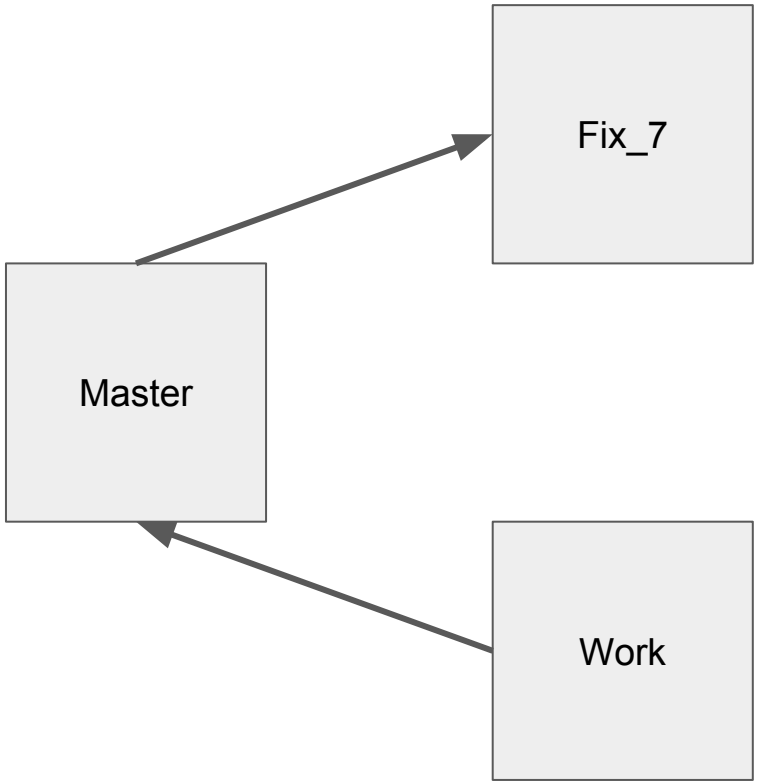
- `git commit -m “save work”  # in your personal repo`
- `cd <course repo>`
- `git pull origin master  # pull the new changes`
- `cd <personal repo>`
- `cp -R <course repo>/hw01/* .   # overwrite everything!`
- `git add --patch hw01.py   # add pieces of changed file`
- `git commit -m “put in hw01.py fixes from prof”`
- `git checkout --   # this discards changes not committed.`

All Good! Now repeat for other changed files (e.g. the notebook file).

Demo...

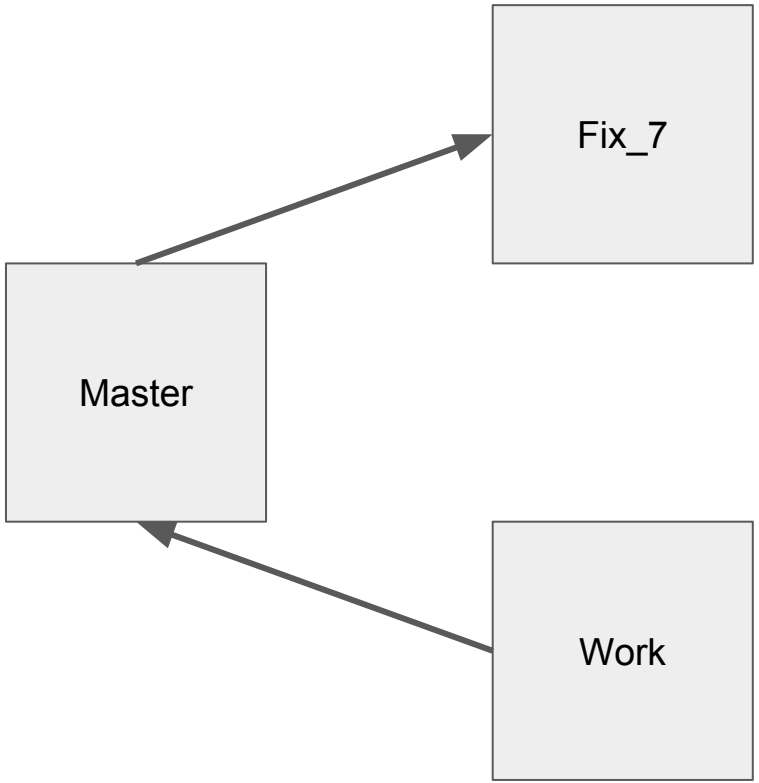
The better way (branches)

- Good practice not working in master.
- Easier to understand code-change history using branches.
- The merging of the code happens automatically, instead of picking which parts you want to add manually.
- *Caveat:* if the fix and your working code are changing the same lines, then you have to manually fix the conflicts (git add --patch)



The better way (branches)

- Master contains the (up-to-date) blank assignment. You **never** work in master.
- You always work in your branch (here, called “Work”). Commit your work in this branch.
- When a fix to the blank assignment occurs, create a branch *from Master* (here, called “Fix_7”) and update the assignment with the fix.
- Lastly, merge the fix into master, then merge the new master into your working branch



Step IV: The commands.

- `git commit -a -m “save work”  # in branch `work``
- `git checkout master    # make sure it’s up-to-date`
- `git checkout -b fix_7  # create/co new branch from master`
- `cp -R <course repo>/hw01/* .   # overwrite everything!`
- `git commit -a -m “new fix from course staff”`
- `git checkout master`
- `git merge master   # create a correct, blank assignment`
- `git checkout work  # go to working branch`
- `git merge master   # bring in the assignment fix`

Demo...