213/513 ~~Linux/~~Git Bootcamp

Eugene, Niko, Matt, and Oliver

GIT

What is git?

- Version control system
  - Better than:
    - copy pasting code
    - emailing the code to yourself
    - taking a picture of your code and texting it to yourself
    - zipping the code and messaging it to yourself on facebook
- git ≠ github
- **using git this semester is mandatory!!!** ~*style*~ point deductions if you don't use it

Important commands

`$ git init`	make a new repository
`$ git clone`	initialize a repository locally from a remote server
`$ git status`	MOST IMPORTANT COMMAND
`$ git log`	show commit history. Can use --decorate --graph --all to make it pretty
`$ git add`	stages files to be committed. Flags: --a, -u
`$ git commit -m`	commit the changes in the staged files (use good messages!)
`$ git push`	push changes to a remote server (--set-upstream origin branchname)
`$ git pull`	pull changes from a server
`$ git branch`	make a new branch
`$ git checkout`	switch to a different branch. Can use -b to make a new branch
`$ git merge name`	merge "name" branch into your current branch
`$ git reset HEAD`	Used to unstage files
`$ git reset --hard + hash`	Used to reset to an old commit (with a commit hash)

Example

https://github.com/eyluo/linux-bootcamp

if that link is too long, try:

https://tinyurl.com/goKnicks213

Configuring git

```
$ git config --global user.name "<Your Name>"

$ git config --global user.email “<Your Email>”

$ git config --global push.default simple
```

(Make sure the email is your Andrew ID, and make sure to add that email to your GitHub account!)

Cloning the repository

1. Go to to link in previous slide and click “fork” in the top right corner to copy the repository to your Github account
2. Make sure you are in your account, and click the green “clone or download” on the right
3. Copy the link
4. Open up a terminal window (or xterm for windows users) and ssh into a shark machine
   a. `$ ssh ANDREW-ID@shark.ics.cs.cmu.edu`
   b. navigate to a folder where you want to do this example
5. `$ git clone` + the link you copied
   a. This will initialize the git repository on your computer, with GitHub as the remote server
6. `$ cd` switch into the repository

Committing, pushing, pulling

1. `$ ls` we have 4 files here
2. `$ git status` branch is up to date with the server, nothing to commit
3. `$ git log --graph --decorate --all`
    1. Shows a pretty graph of the commit history.
4. `$ vim example.txt` lets make some changes to example.txt
5. `$ git status` now shows that we have unstaged files
6. `$ git add example.txt` stages the file to be committed
7. `$ git reset HEAD example.txt` unstages the file (to show you how to do that)
8. `$ git add example.txt` to restage the file
9. `$ git commit -m “insert a relevant commit message here”`
10. `$ git status` shows you are 1 commit ahead of “origin” = remote server
11. `$ git push` this updates the remote server
12. `$ git log --graph --decorate --all` now we can see the new commit on top of all the old ones

Merging

1. `$ git log --graph --decorate --all`note the other branch "realistic ending" that branches away from master
2. `$ git checkout realistic_ending`switch to the other branch
3. `$ git branch` shows all of our branches
4. `$ ls` note that there are different files here
5. `$ vim example.txt` we can see the story is different than in the master branch-finish it!
6. Add and commit the file, push to the server.
7. `$ git checkout master` switch back to the master branch
8. `$ git merge realistic_ending`will attempt to merge the two branches, but there's a conflict
   a. `$ git status` shows that the conflict is in example.txt
   b. `$ vim example.txt` fix the story
   c. `$ git add example.txt`
   d. `$ git commit -m "appropriate message for a merge"`now the merge is complete
9. `$ git log -- decorate --graph --all`shows that now you still have 2 branches, but they've been merged and point to the same files

Resetting, Branching

1. `$ git log --decorate --graph --all` copy the commit hash of a past commit (first 6ish characters usually fine)
2. `$ git branch newbranchname` make a new branch
3. `$ git checkout newbranchname` switch to the new branch
4. `$ git reset --HARD + hash` from old commit
5. `$ git log --decorate --graph --all` note that now HEAD is at the old commit, master is still at the merge commit from last slide
6. `$ ls` the files are different now
7. `$ vim example.txt` the story is different too. Add a line or two to it
8. Add and commit
9. `$ git log --decorate --graph --all` now we can see how it has separated from the rest of the tree
    a. This is how you would test out new feature. If you decide you like it, you can later merge it into the master branch. If not, you can just leave it and switch back to master.

Adding your new branch to the remote server

1. `$ git status` note that it says nothing about the origin remote server
2. `$ git push` doesn't work, there is no "upstream branch" (nothing on the server)
3. `$ git push --set-upstream origin newbranchname`
   a. This creates a new branch on the origin server, and sets it as the "upstream" of your current branch. In the future when you push, you can just do git push and it will work.
4. `$ git status` now branch is up to date with origin/newbranchname
5. `$ git checkout master`
6. `$ git status` we're far ahead of the remote server
7. `$ git push`

$ .gitignore files

- Make one in each of your projects
  - Can use touch, emacs, vim, whatever you want
- *.o will ignore all .o files (object files)
- Useful because when you add a lot of new files with $ git add -a you want git to ignore certain files