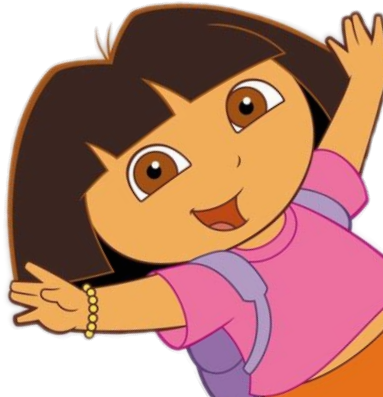
213/513/613 Linux/Git Bootcamp

Sophie, Urvi

Outline

1. SSH, bash, and navigating Linux
2. Using VIM
3. Setting up VS Code
4. Git



SSH

1. On macOS/Linux:

   ```
   $ ssh ANDREW-ID@shark.ics.cs.cmu.edu
   ```

   (don’t type in the “$”! This just means you’re typing what follows into terminal)

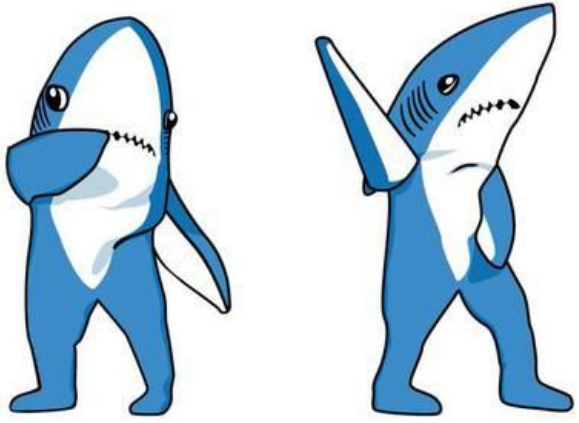
1. Type your password when prompted.
2. If you see a warning about SSH host keys, click or enter “yes.”

Windows computers???

- Use MobaXTerm for file transfer and ssh client!
- Instructions can be found here: http://www.cs.cmu.edu/~213/activities/linux-bootcamp/windows-setup.pdf

What are shark machines?

- Shark machines, linux.andrew.cmu.edu and unix.andrew.cmu.edu are all machines that access the same Andrew File System (AFS).
- Shark machines are explicitly set up for 213: they're standardized for benchmark tests and have correct versions of gcc, gdb and other tools.



Use the shark machines… otherwise your compiled code won't behave as expected!!!

Error: ssh: Could not resolve hostname ANDREW-ID@shark.ics.cs.cmu.edu

- log into a specific shark machine
- $ ssh ANDREW-ID@[favourite]shark.ics.cs.cmu.edu
- The list of shark machines is listed under the “Lab Machines” section of the 213 website

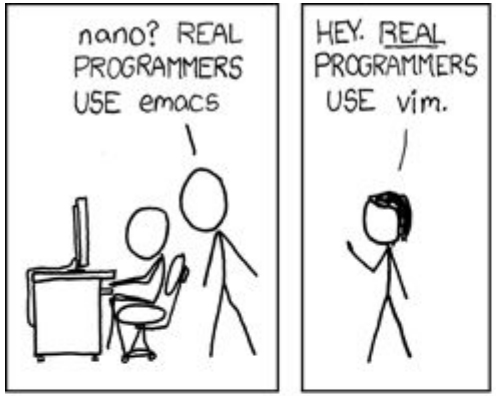
Navigating the shark machines

- **$ ls** list all files in folder. “-a” flag lists hidden files
- **$ pwd** print current file path
- **$ cd PATH** enter folder PATH. “.” is current folder, “..” is parent
- **$ mkdir NAME** make a folder called NAME
- **$ touch NAME** make a file called NAME
- **$ rm NAME** remove file called NAME
- **$ cat NAME** output file NAME’s content to commandline
- **$ mv FILE DEST** move FILE to DEST folder
- **$ cp FILE DEST** move FILE to DEST folder
- **$ tar OPT NAME** compress to tar file with name NAME or open tar file based on OPT flags
- **$ scp FILE ANDREW-ID@shark.ics.cs.cmu.edu:DEST** move FILE from local machine to DEST folder on shark machine

Editing files

Option 1: VIM

- Can be run on pretty much any terminal.
- Highly customizable.
- According to legend, if you learn all the keyboard shortcuts, the rate at which your fingers travel approach lightspeed, to the point of being a potential hazard to those in your general vicinity.



Option 1: VIM

1. Let's start by SSHing into the shark machines!

   ```
   $ ssh ANDREW-ID@shark.ics.cs.cmu.edu
   ```

1. From here, let's make VIM *spicy* by running the following:

   ```
   $ vim ~/.vimrc
   ```

1. Three big modes:
   a. Normal mode: press the "esc" key.
   b. -- INSERT -- mode: press the "i" key in normal mode.
   c. -- VISUAL -- mode: press the "v" key in normal mode.

Option 1: VIM

4. Press “i” and make sure you see “-- INSERT --” at the bottom. Then type that into the text buffer → → →
5. When done, press “esc” and then type in “:w” to save.
6. Type in “:q” to quit VIM. (This can be combined into “:wq” to save and quit in one command :-0)

```
colorscheme desert
set mouse=a
set number
set cursorline
set colorcolumn=81
set tabstop=4
set shiftwidth=4
set softtabstop=4
set expandtab
set smartindent
```

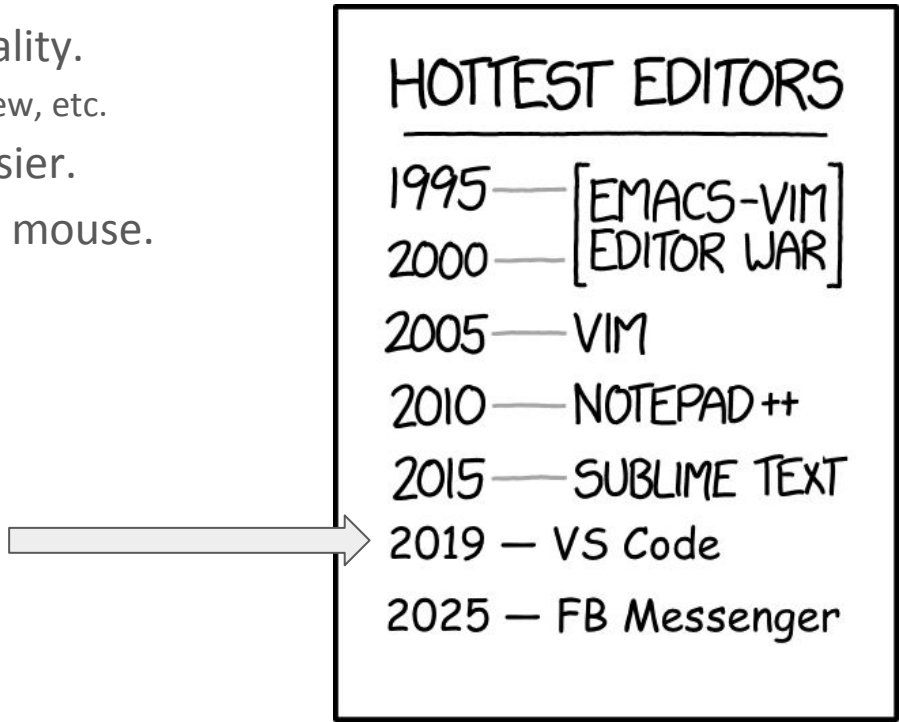
Option 1: VIM

- Normal mode: “esc” key
  - -- INSERT -- mode: “i” key
    - Type and stuff :-0
  - -- VISUAL -- mode: “v” key
    - Use arrow keys to highlight a selection
    - “Copy and paste”:
      - Highlight text, press “y” to yank (copy) and “p” to paste
      - Similarly, pressing “d” twice will delete the selection, which also makes it available to paste with “p”
  - Save: “:w”
  - Quit: “:q”
- With the given .vimrc, you can also scroll and click with the mouse
- Try **`$ vimtutor`** for more tips and tricks for VIM that we might not have covered!

Option 2: VS Code + SFTP

About VS Code:

- Text editor with lots of traditional functionality.
  - Tabs, easy window split, built-in terminal, tree view, etc.
- Cool plugins to make code prettier + life easier.
- People won't make fun of you for using the mouse.
  - (Except for VIM purists)
  - (Don't let the haters get to you)



Option 2: VS Code + SFTP

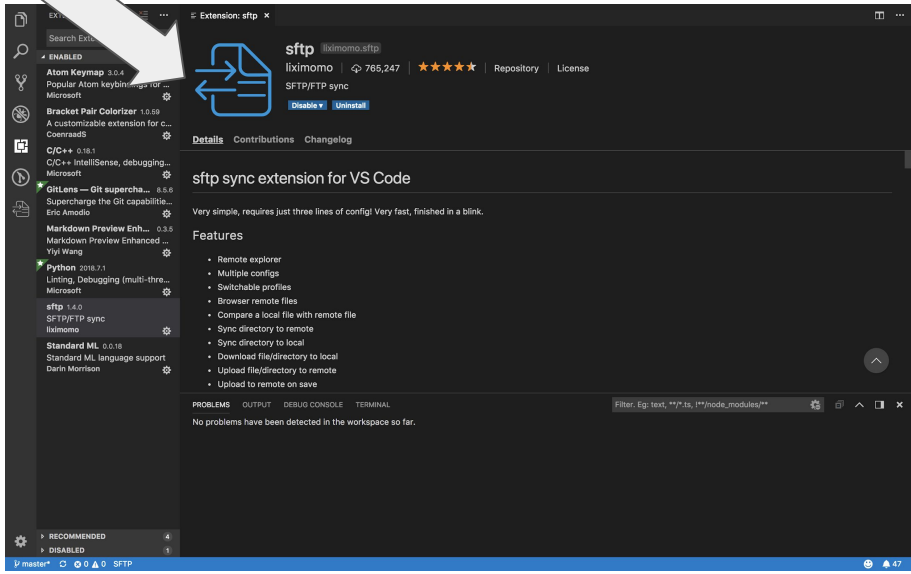
About SFTP:

- **S**ecure **F**ile **T**ransfer **P**rotocol
- Can be used to read from and write to files in your AFS directory by synchronizing changes to files saved directly to your machine
- **Note on academic integrity**: since the files exist on both your machine and AFS, you are responsible to make sure that no one gets access to either location!

Option 2: VS Code + SFTP

On your local machine:

- Download VS Code here: https://code.visualstudio.com/download
- You can check out some of the other extensions (linting for C for style???) on your own time, but download liximomo's SFTP plugin because that's how we're gonna be working with the Shark machines.



Option 2: VS Code + SFTP

```
{
    "name": "<this-can-be-anything>",
    "protocol": "sftp",
    "host": "<your-favorite-shark-machine>.ics.cs.cmu.edu",
    "port": 22,
    "username": "<andrew-id>",
    "remotePath": "/afs/andrew.cmu.edu/usr3/eyluo/private/15213/linux-bootcamp",
    "uploadOnSave": true,
    "downloadOnOpen": true,
    "ignore": [
        ".vscode",
        ".git",
        ".DS_Store",
        "admin"
    ]
}
```

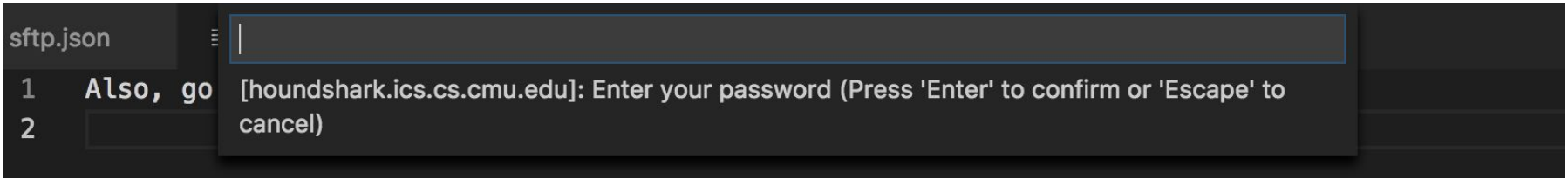
On your local machine:

- Create a local 213 folder.
- Inside there, create a folder called “linux-bootcamp.” Open it in VSCode.
- Ctrl + Shift + P (Windows) or Cmd + Shift + P (Mac) to open up Command Palette:
- Type in “SFTP: Config”
  - This should open “sftp.json”
- Type in the info in the top-right.
  - **Notes:** “uploadOnSave” will automatically save any local changes whenever you save the file. “downloadOnOpen” will automatically update your local version with the AFS version when you open the file.
- Visit https://github.com/liximomo/vscode-sftp/wiki/config for extra config options

Option 2: VS Code + SFTP

On your local machine:

- Create a file called “example.txt” and type whatever you want into it.
- When you save, this should prompt a popup to type in your SSH password.



Option 2: VS Code + SFTP

On your favorite Shark machine:

- SSH into your favorite Shark machine.
- `$ cd` into your example directory and list the files. You should see “example.txt” inside!

Option 2: VS Code + SFTP

Reminders:

- SFTP means you're downloading code from AFS onto your local machine, so take extra precaution to make sure that code is secure and no one steals it!
- Any time you run `$ make`, please do so on the Shark machines!!

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 will (with high probability) be 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, tells you information about what is going on
- **`$ git log`** show commit history. Can use --decorate --graph --all to make it pretty.

- **`$ git add`** stages files to be committed. Flags: --a (all files), -u (only previously added files)
- **`$ git rm`** stages files to be removed.
- **`$ git reset HEAD + file`** unstages “file” from the commit
-
- **`$ git commit -m`** commit the changes in the staged files. Write descriptive, meaningful messages!
- **`$ git push`** push changes to a remote server.
- **`$ git pull`** pull changes from a server

- **`$ git reset --hard + hash`** Used to reset to an old commit (with a commit hash)
- **`$ git checkout + filename`** Used to reset any changes made to a file to previous commit

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!)

.gitignore files

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

  `$ git add -a`

  you want git to ignore certain files

Example 1

- We will now walk through a git tutorial.
- Visit the following: https://github.com/Guptacos/git-bootcamp

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` to enter the working repository.

Committing, pushing, & pulling

1. `$ ls -a` we have 2 files and 2 folders here
2. `$ git status` branch is up to date with the server, nothing to commit
3. `$ git log --graph --decorate --all`
   Shows a pretty graph of the commit history.
4. `$ cd cprogramminglab-handout`
5. `$ vim queue.c` make some changes to queue.c (can use nano, emacs, etc)
6. `$ git status` now shows that we have unstaged files
7. `$ git add queue.c` stages the file to be committed
8. `$ git status` now shows that we staged example.txt
9. `$ git reset HEAD queue.c` unstages the file (to show you how to do that)
10. `$ git status` note that example.txt is unstaged again
11. `$ git add queue.c` stage the file again
12. `$ git commit -m "insert a relevant commit message here"`
13. `$ git status` shows you are 1 commit ahead of "origin" = remote server
14. `$ git push` this updates the remote server
15. `$ git log --graph --decorate --all` now we see the new commit on top of all the old ones

Appendix: advanced git

- `$ git branch` make a new branch
- `$ git checkout` switch to a different branch. Use -b to make a new branch
- `$ git merge name` merge “name” branch into your current branch

Example 2

- The beginning of this example covers basic git, and the later slides cover more advanced topics (branching)
- Visit the following: https://github.com/eyluo/linux-bootcamp
- Follow the instructions from example 1 to fork and clone the repository. Then `$ cd` to enter the repository and follow along with the next few slides.

Committing, pushing, & pulling

1. `$ ls -a` we have 5 files and a folder here
2. `$ git status` branch is up to date with the server, nothing to commit
3. `$ git log --graph --decorate --all`
    i. Shows a pretty graph of the commit history.
4. `$ vim example.txt` make some changes to example.txt (can use nano, emacs, etc)
5. `$ git status` now shows that we have unstaged files
6. `$ git add example.txt` stages the file to be committed
7. `$ git status` now shows that we staged example.txt
8. `$ git reset HEAD example.txt` unstages the file (to show you how to do that)
9. `$ git status` note that example.txt is unstaged again
10. `$ git add example.txt` stage the file again
11. `$ git commit -m "insert a relevant commit message here"`
12. `$ git status` shows you are 1 commit ahead of "origin" = remote server
13. `$ git push` this updates the remote server
14. `$ git log --graph --decorate --all` now we 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 now different files shown
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 log --decorate --graph --all`

   Note that now there is an origin/newbranchname branch
5. `$ git status` now branch is up to date with origin/newbranchname
6. `$ git checkout master`
7. `$ git status` we're far ahead of the remote server
8. `$ git push`