Operating Systems!

OS Tech Bootcamp
(Part 1)

Prof. Travis Peters

Montana State University

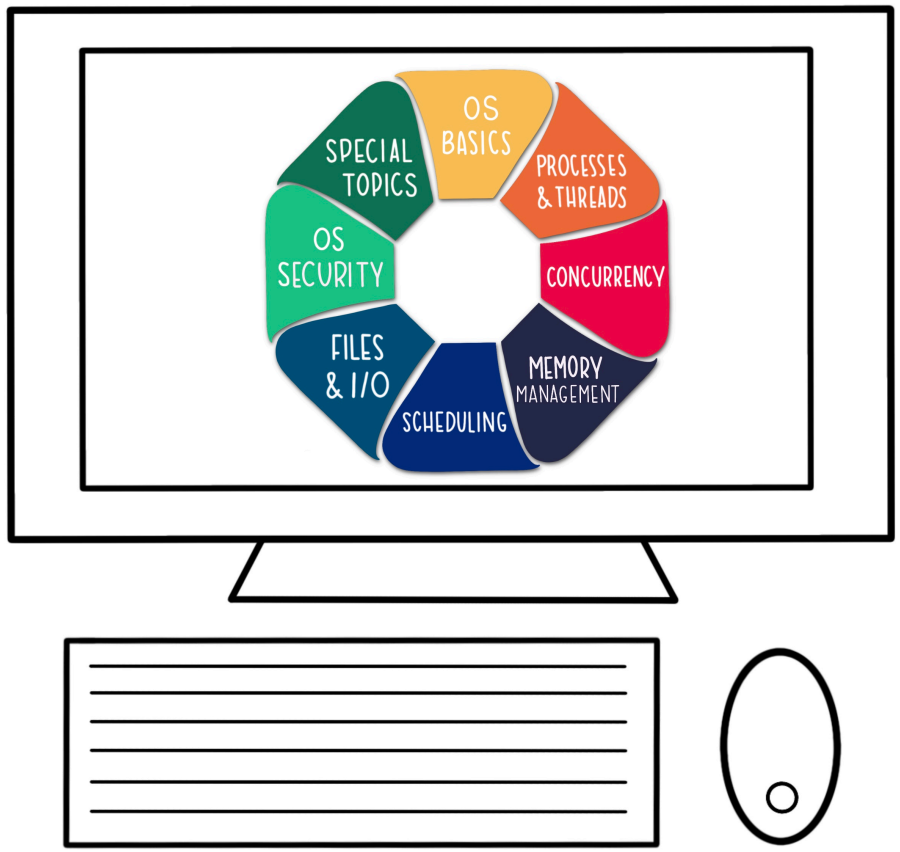
CS 460 - Operating Systems

Fall 2020

https://www.cs.montana.edu/cs460

OS Tech Bootcamp (Overview)

- VirtualBox & Vagrant
  - A sandbox
  - standardized work environment
- Basic Command Line
  - An interface to the computer / OS
- Working in C
  - Automating compilation w/ make
  - Debugging w/ gdb
- Teamwork & Communication
  - READMEs & Markdown
  - Git & GitHub
- *Coming Up:*
  - System programming (fork(), exec())
  - OS internals (e.g., procs, threads)



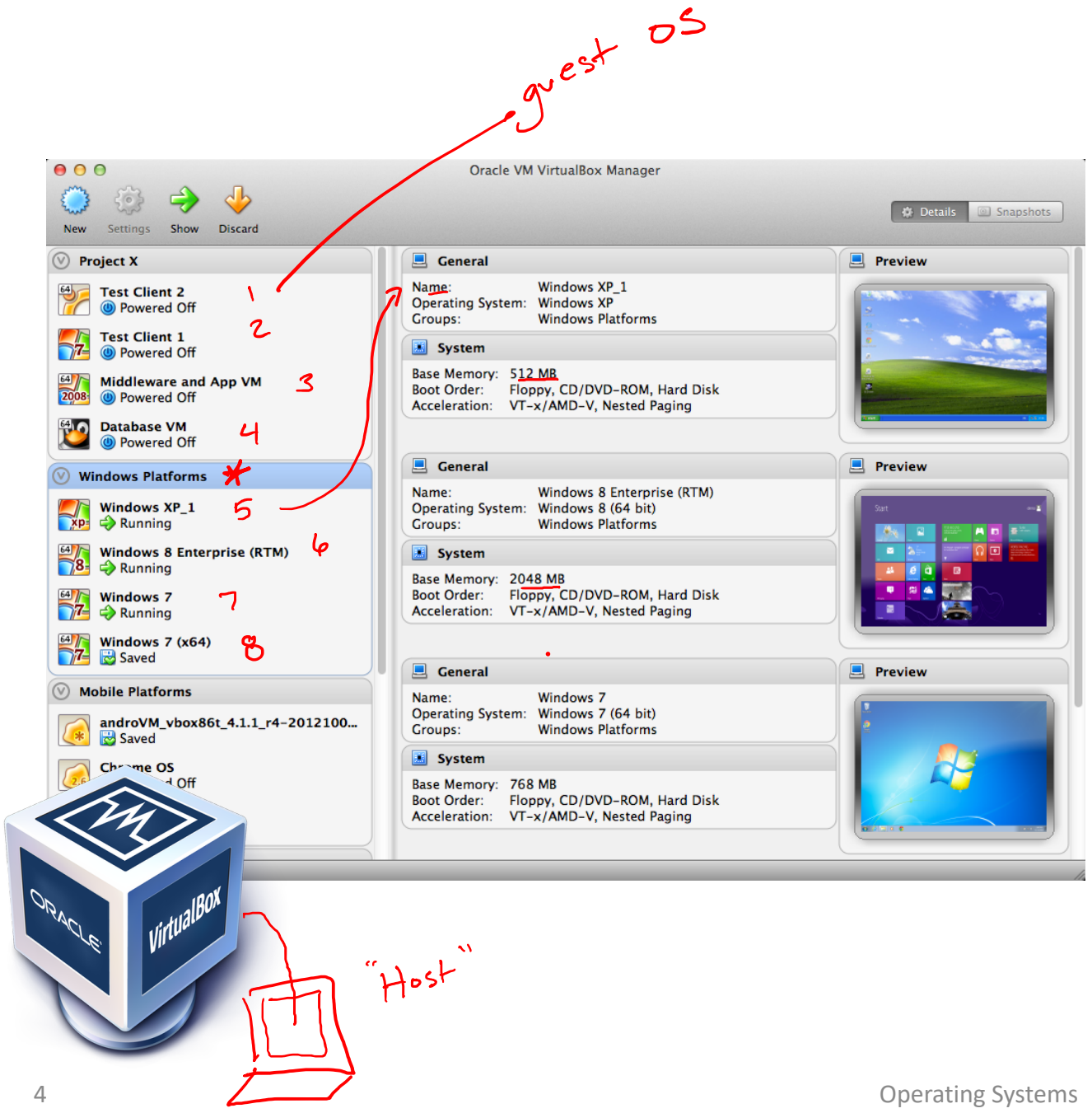
Throughout "bootcamp" we will work through (many) parts of PA0.

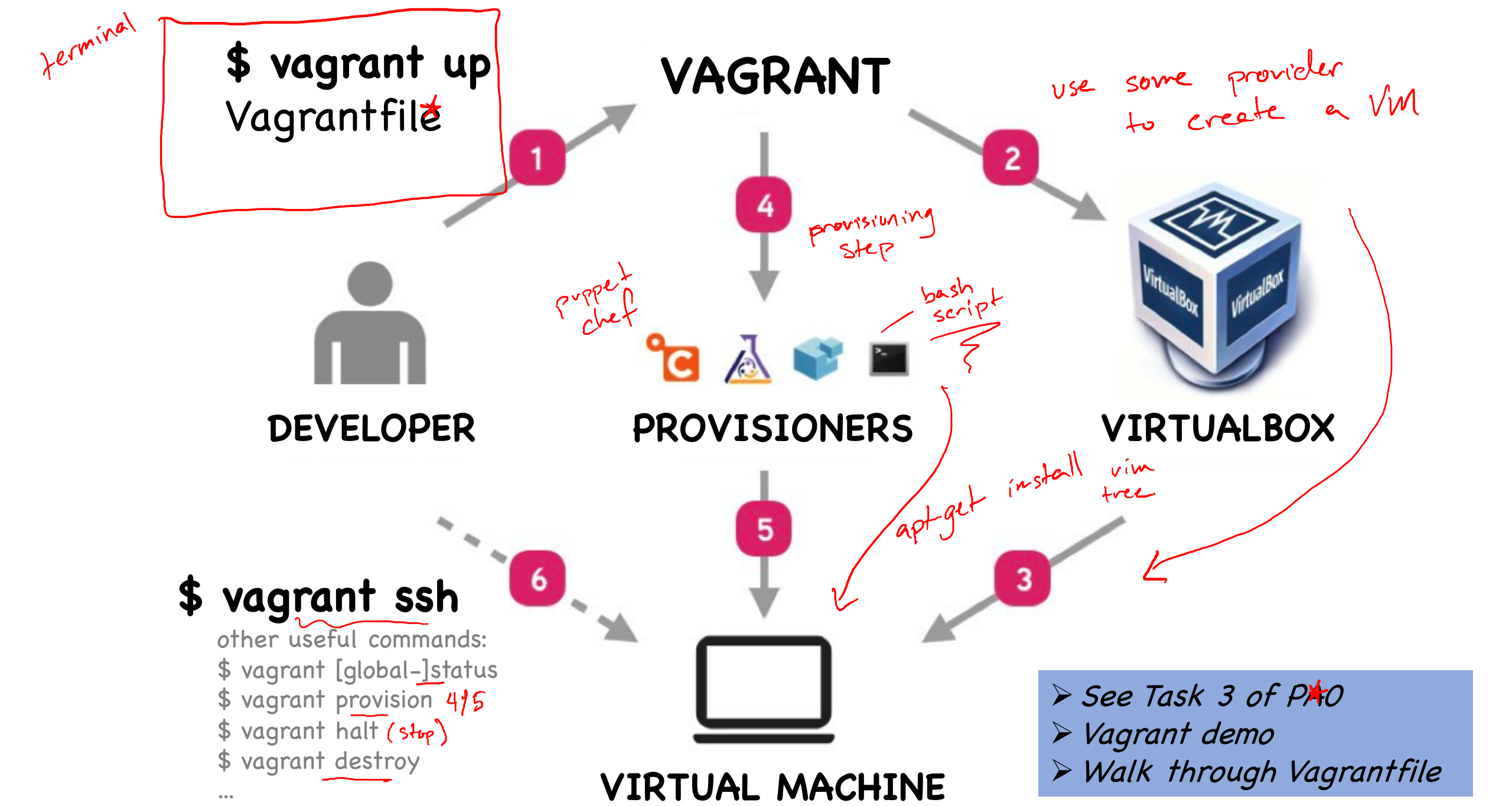
VirtualBox & Vagrant (Task 3)

Because VirtualBox can help us to standardize our virtual environments...
and because Vagrant can automate a lot of the tedious stuff!

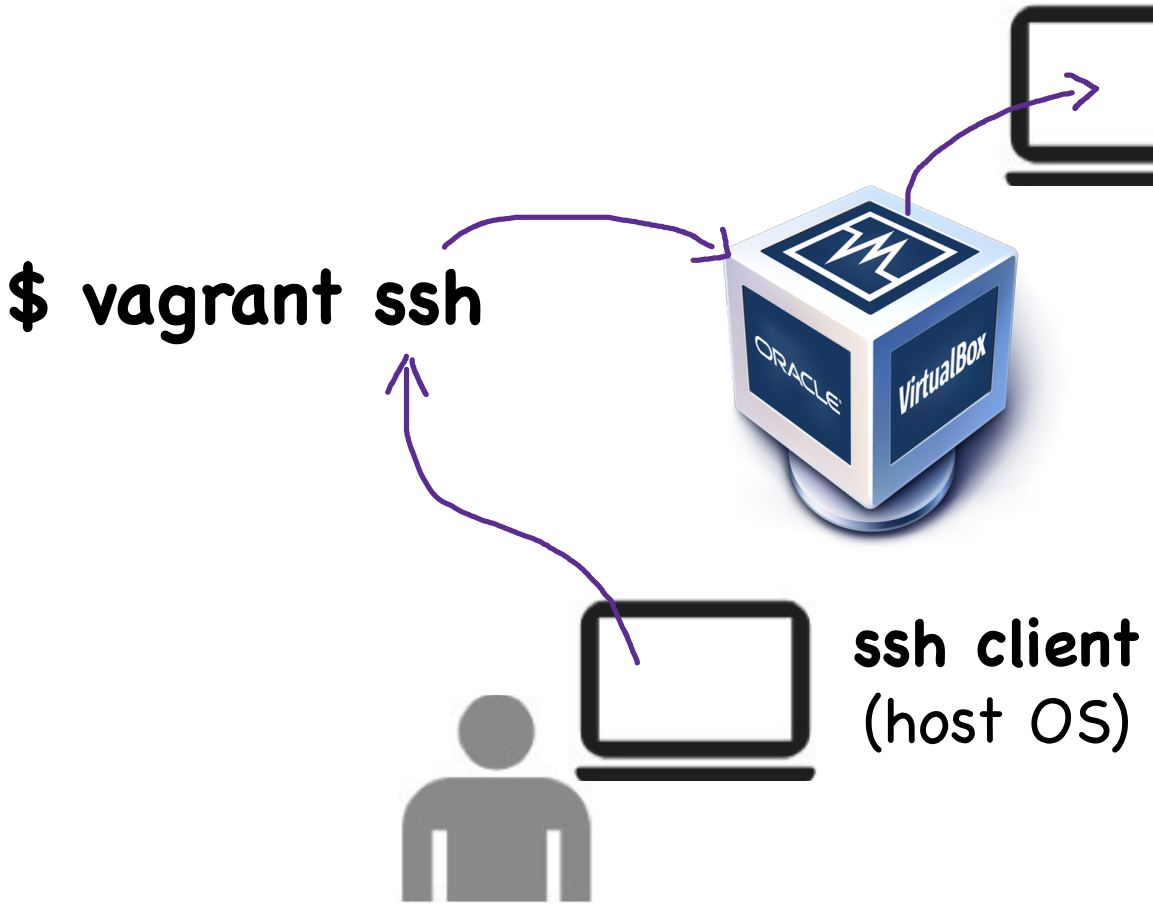
VirtualBox

- A free and open-source hosted **hypervisor** for x86 virtualization, developed by Oracle
  - VirtualBox can be run under Windows, GNU/Linux, macOS, Sun Solaris and FreeBSD.
  - VirtualBox can run multiple **guest OSs** under a single **host OS**
  - It supports the creation and management of **guest virtual machines** running Windows, Linux, BSD, Solaris, etc.




terminal
$ vagrant up
Vagrantfile
VAGRANT
use some provider to create a VM
1
2
4
provisioning step
puppet chef
bash script
DEVELOPER
PROVISIONERS
VIRTUALBOX
apt-get install vim tree
5
6
3
$ vagrant ssh
other useful commands:
$ vagrant [global-]status
$ vagrant provision 4/5
$ vagrant halt (stop)
$ vagrant destroy
...
VIRTUAL MACHINE
See Task 3 of P0
Vagrant demo
Walk through Vagrantfile

Using vagrant to interact with your VM




```
# Run vagrant commands from
# top-level directory on HOST
$ vagrant up
$ vagrant ssh
$ vagrant [global-]status
$ vagrant provision
$ vagrant halt
$ vagrant destroy
...
```

Recommended Setup

```
# Create a top-level folder on your HOST where you plan to do all CSCI 460 work
$ mkdir -p ~/projects/cs460/
$ cd ~/projects/cs460/

# Download private repo as a subdirectory, which contains the Vagrantfile
$  git clone YOUR-PRIVATE-REPO # git full command from GitHub

# Link your Vagrantfile to the top-level (all subdirectories synced to VM)
$ ln -s ~/projects/cs460/YOUR-PRIVATE-REPO ~/projects/cs460

# Run vagrant commands from top-level directory
$ vagrant up
$ vagrant ssh
$ vagrant halt
$ vagrant destroy
...
```

READMEs & Markdown (Task 2)

Because communicating important aspects of our projects is... important!

```
#Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipisicing elit, quis
**nostrud exercitation** ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in
*voluptate velit*.

###Code

```javascript
var foo = 'bar';
if(true) foo = 'foo';
```

###Tables

First Header	Second Header
Content from cell 1	Content from cell 2
Content in the first column	Content in the second column

###Lists

- [x] @mentions, #refs, [links](), **formatting**, and <del>tags</
del> supported
- [x] list syntax required (any unordered or ordered list
supported)
- [x] this is a complete item
- [ ] this is an incomplete item
```

Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipisicing elit, quis **nostrud exercitation** ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in *voluptate velit*.

Code

```
var foo = 'bar';
if(true) foo = 'foo';
```

Tables

First Header	Second Header
Content from cell 1	Content from cell 2
Content in the first column	Content in the second column

Lists

- [x] @mentions, #refs, links, **formatting**, and ~~tags~~ supported
- [x] list syntax required (any unordered or ordered list supported)
- [x] this is a complete item

Markdown Activity (Breakout?)

.md = markdown

➔ *See Task 2: READMEs & Markdown (PA0)*

Projects?

pandoc

mbedTLS

⋮

Observations?

Troubleshooting section

Setup (how to get code up & running)

Overview

Documenting features

→ examples (code section)

Operating Systems!

OS Tech Bootcamp
(Part 2)

Prof. Travis Peters

Montana State University

CS 460 - Operating Systems

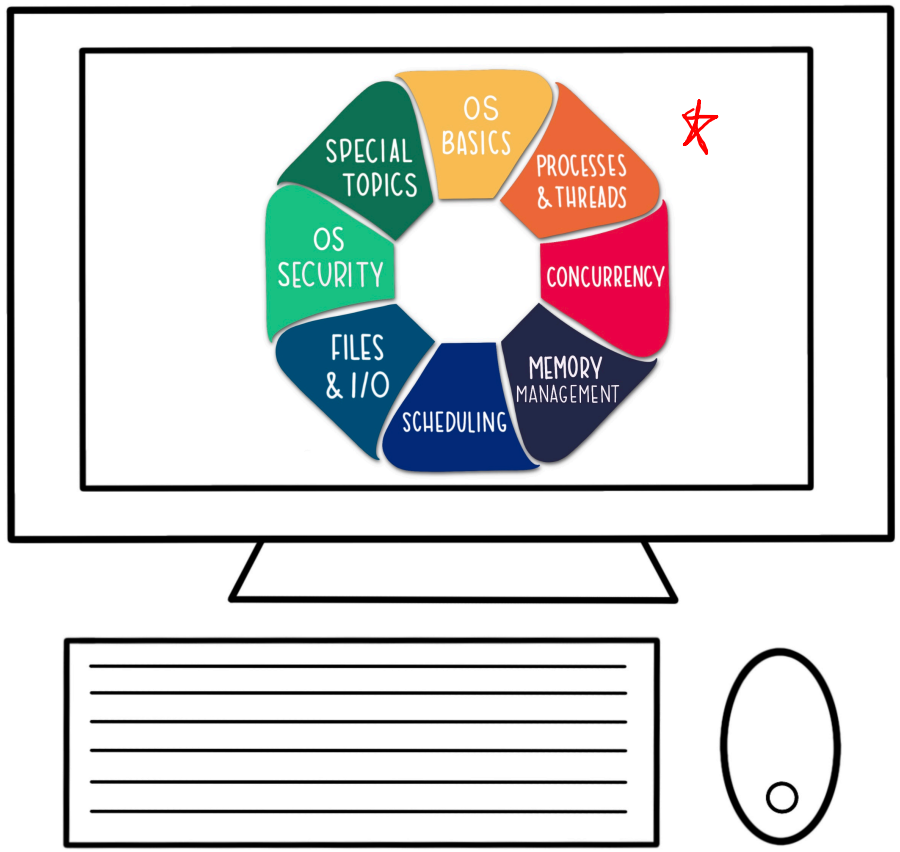
Fall 2020

https://www.cs.montana.edu/cs460

OS Tech Bootcamp (Overview)

- ~~VirtualBox & Vagrant~~
  - ~~A sandbox~~
  - ~~standardized work environment~~
- Basic Command Line
  - An interface to the computer / OS
- Working in C
  - Automating compilation w/ make
  - Debugging w/ gdb
- Teamwork & Communication
  - READMEs ~~& Markdown~~
  - Git & GitHub
- *Coming Up:*
  - System programming (fork(), exec())
  - OS internals (e.g., procs, threads)

Meet Reese!



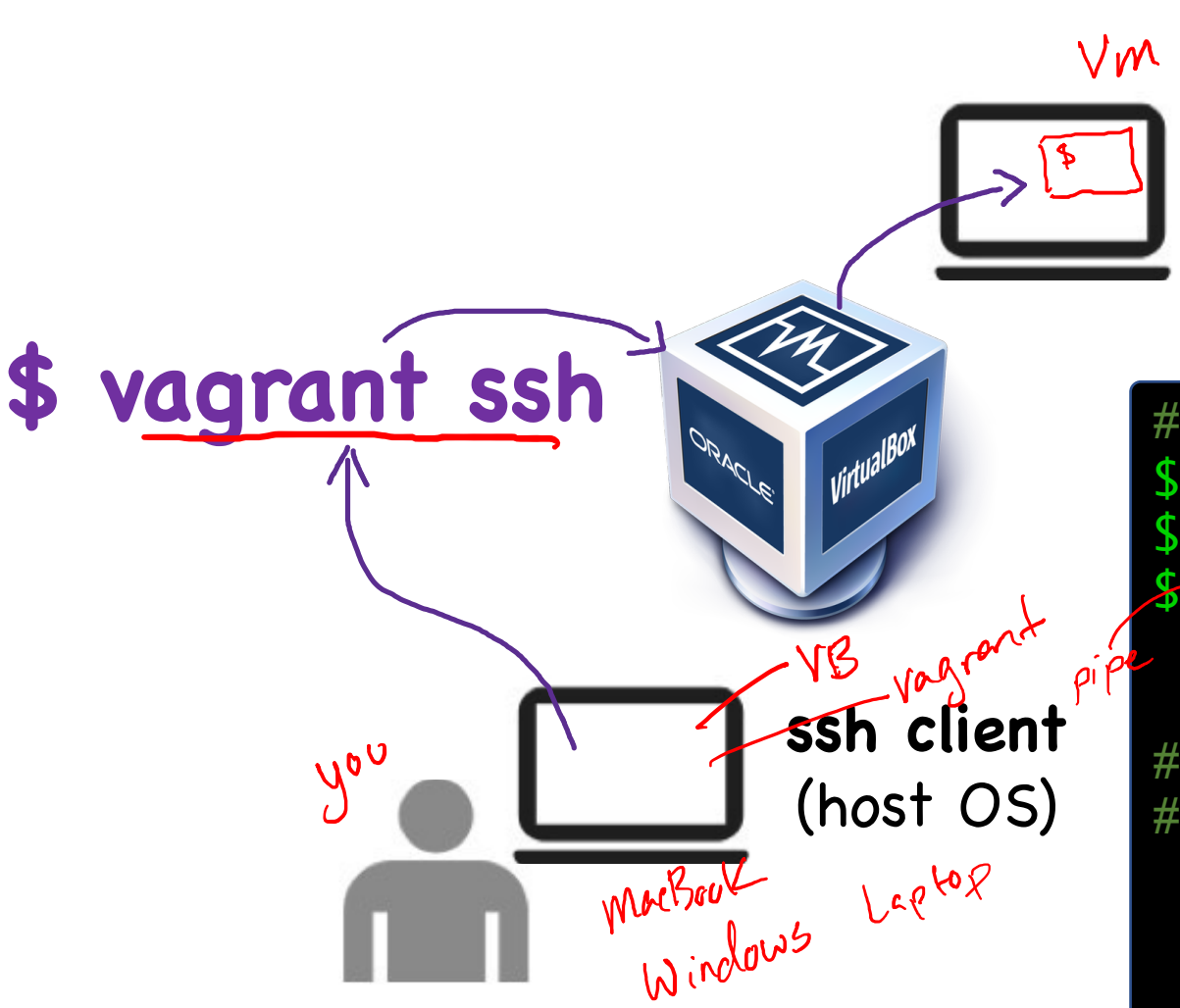
Throughout "bootcamp" we will work through (many) parts of PA0.

Command Line (Task 4)

Because us humans need a way to control and interact with our computers!

Command Line Activity

See Task 4: Command Line (PA0)



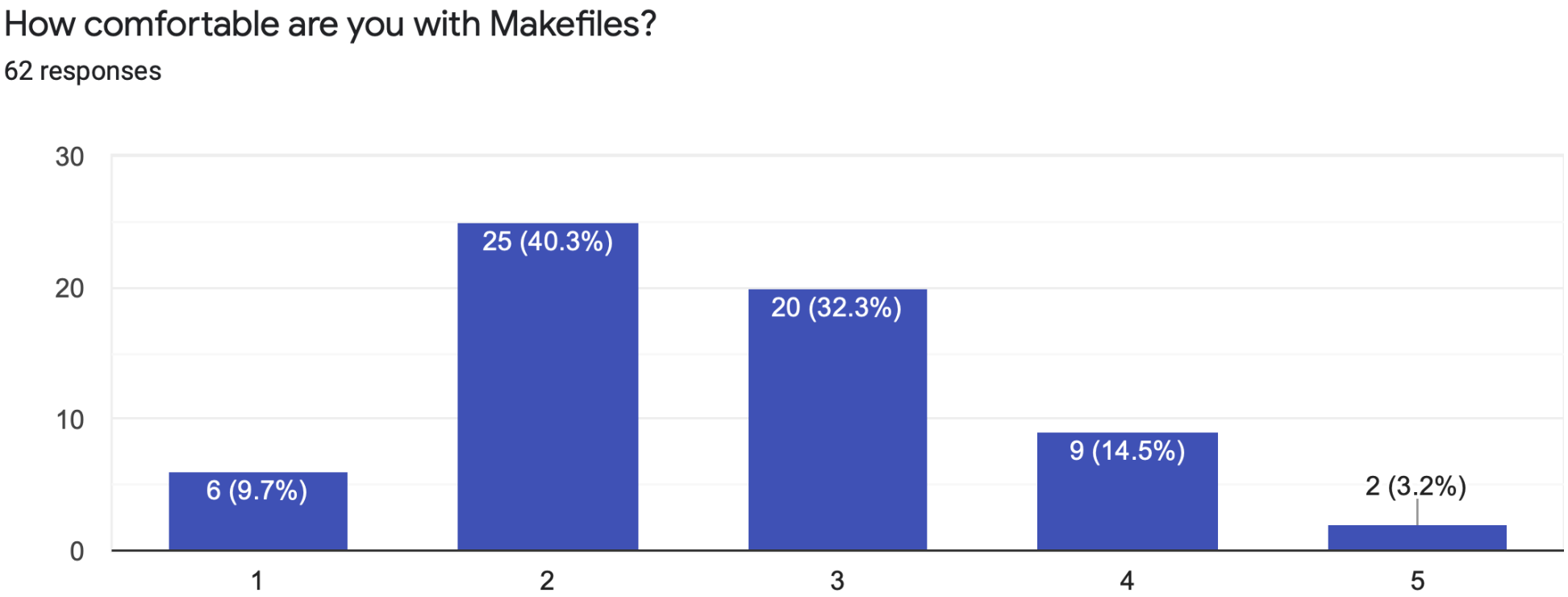

```
# Task 4, Part 1 (a few commands)
$ cd ~
$ env | grep PATH
$ sudo lshw -html > hardwareinfo.html

# Task 4, Part 3
# Some of your Favorite commands?
```

Makefiles (Task 5)

Because having a language/tool to automate more complex compilations (and other stuff) is a game changer!

You Tell Me: Why Do We Need Makefiles?



Automate Something *(Breakout Activity!)*

Your Objective:

How can you explain to me, with ZERO AMBIGUITY,
how to get dressed before going outside.

wear the same thing each day

step by step commands / script

take into account
- preference
- weather
- social norms (socks)
- health concerns

Automating Builds

```
target: [dependencies]
    <shell command to execute>
    <shell command to execute>
```

- Make is a language for automating large compilations.
- You need to recompile a file ("target") only if it is older than one or more of its dependencies. *(Thus, you don't need to recompile if the target exists AND it is newer than all of its dependencies.)*

Put Simply:

Makefiles are just text files...

...with clear rules that express how to do something...

*...like **build files** you want to build or **do some task** in a logical order!*

A Non-Technical Example

See Task 5: Makefiles (PA0)

Makefile

```
dress: trousers shoes jacket
    @echo "All done. Let's go outside!"
jacket: pullover
    @echo "Putting on jacket."
pullover: shirt
    @echo "Putting on pullover."
shirt:
    @echo "Putting on shirt."
trousers: underpants
    @echo "Putting on trousers."
underpants:
    @echo "Putting on underpants."
shoes: socks
    @echo "Putting on shoes."
socks: pullover
    @echo "Putting on socks."
```

Thanks! *Afraid of Makefiles? Don't be!*
https://endler.dev/2017/makefiles/

1. Download this file
(See our GitHub: /week02/makefile-intro/Makefile)

2. SSH into your VM and change to the directory where you put your Makefile. Run the Makefile:

```
$ make
```

3. What do you see? Does it make sense? Explain...

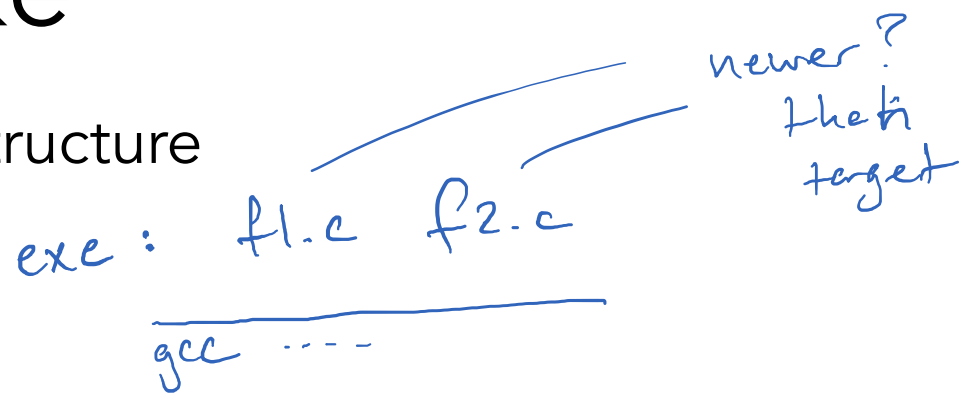
< Breakout Activity! >

```
Putting on underpants.
Putting on trousers.
Putting on shirt.
Putting on pullover.
Putting on socks.
Putting on shoes.
Putting on jacket.
All done. Let's go outside!
```

Automating Builds w/ Make

- In general, each build step has the following structure

```
target: [dependencies]
    <shell command to execute>
    <shell command to execute>
```



- Some good things to knows...
  - The first target in a Makefile will be executed by default if you run make w/out args
  - Shell commands MUST be indented with a TAB
  - .PHONY: *list targets here if target isn't a file you build (e.g., clean, install, run, check)*

***Congrats!* You now know 90% of what you really need to know about Make and Makefiles!**

Really, the rest is mostly just abbreviations, macros, and default rules that help you write shorter/simpler Makefiles

Automating Builds w/ Make *(cont.)*

- A few things that you might come across:

```
# CC is a default macro that you can override to set the default compiler
CC = gcc

# CFLAGS is often used to hold flags given to gcc for compilation.
# To use CFLAGS, you need to put it inside "$(...)" – see below.
CFLAGS = -g -DDEBUG

# Make supports pattern rules using '%' as a wildcard
%.o: %.c
    gcc $(CFLAGS) -c $<

# Here is an example from Makefiles we will use & look at (not shown here: definitions for PROG, OBJS, LIBS)
$(PROG): $(OBJS)
    $(CC) $(CFLAGS) $^ $(LIBS) -o $@
```

See also: Makefile-ideas

In our GitHub: /week02/makefile-intro/Makefile-ideas

- Oh and FYI
  - **@** *= put before a shell command; suppresses echoing command when running it*
  - **$@** *= full name of the target*
  - **$^** *= ALL of the files listed on the dependency line (everything after ':')*
  - **$<** *= the FIRST dependency listed on the dependency line*

Operating Systems!

OS Tech Bootcamp
(Part 3)

Prof. Travis Peters

Montana State University

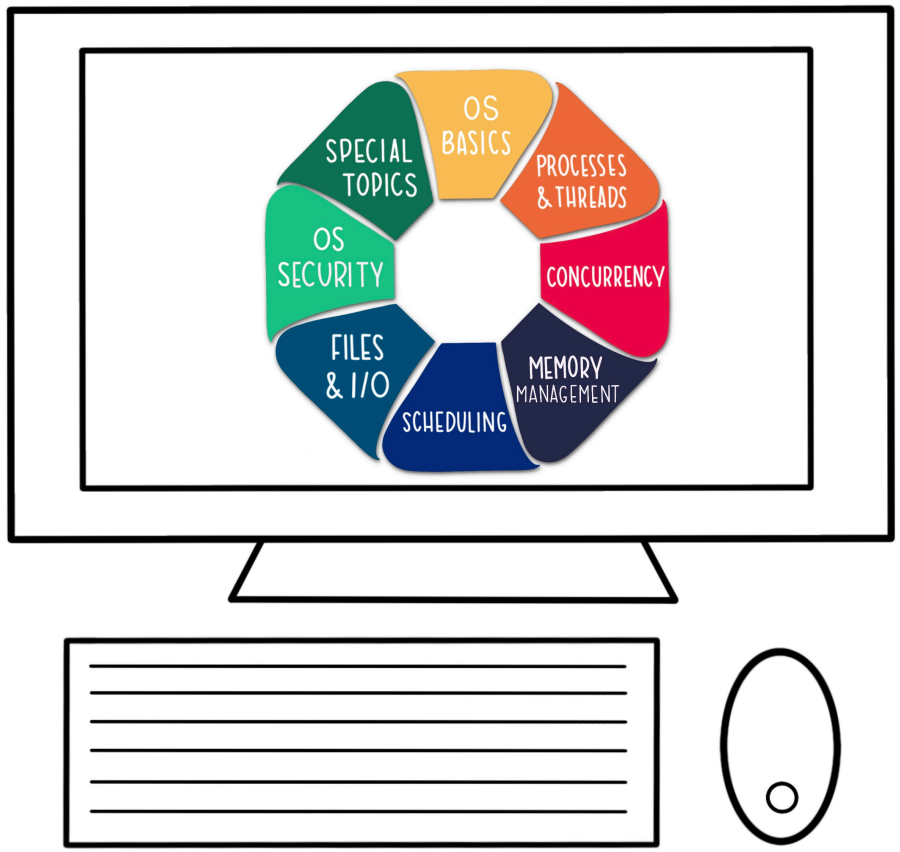
CS 460 - Operating Systems

Fall 2020

https://www.cs.montana.edu/cs460

OS Tech Bootcamp (Overview)

- ~~VirtualBox & Vagrant~~
  - ~~A sandbox~~
  - ~~standardized work environment~~
- ~~Basic Command Line~~
  - ~~An interface to the computer / OS~~
- Working in C
  - ~~Automating compilation w/ make~~
  - Debugging w/ gdb
- Teamwork & Communication
  - ~~READMEs & Markdown~~
  - Git & GitHub
- *Coming Up:*
  - System programming (fork(), exec())
  - OS internals (e.g., procs, threads)



Throughout "bootcamp" we will work through (many) parts of PA0.

Programming in C (Task 6)

Because in systems programming, we need a high-level language that hides as little as possible, and because most OSs are written in C/C++.

Some C Code Examples

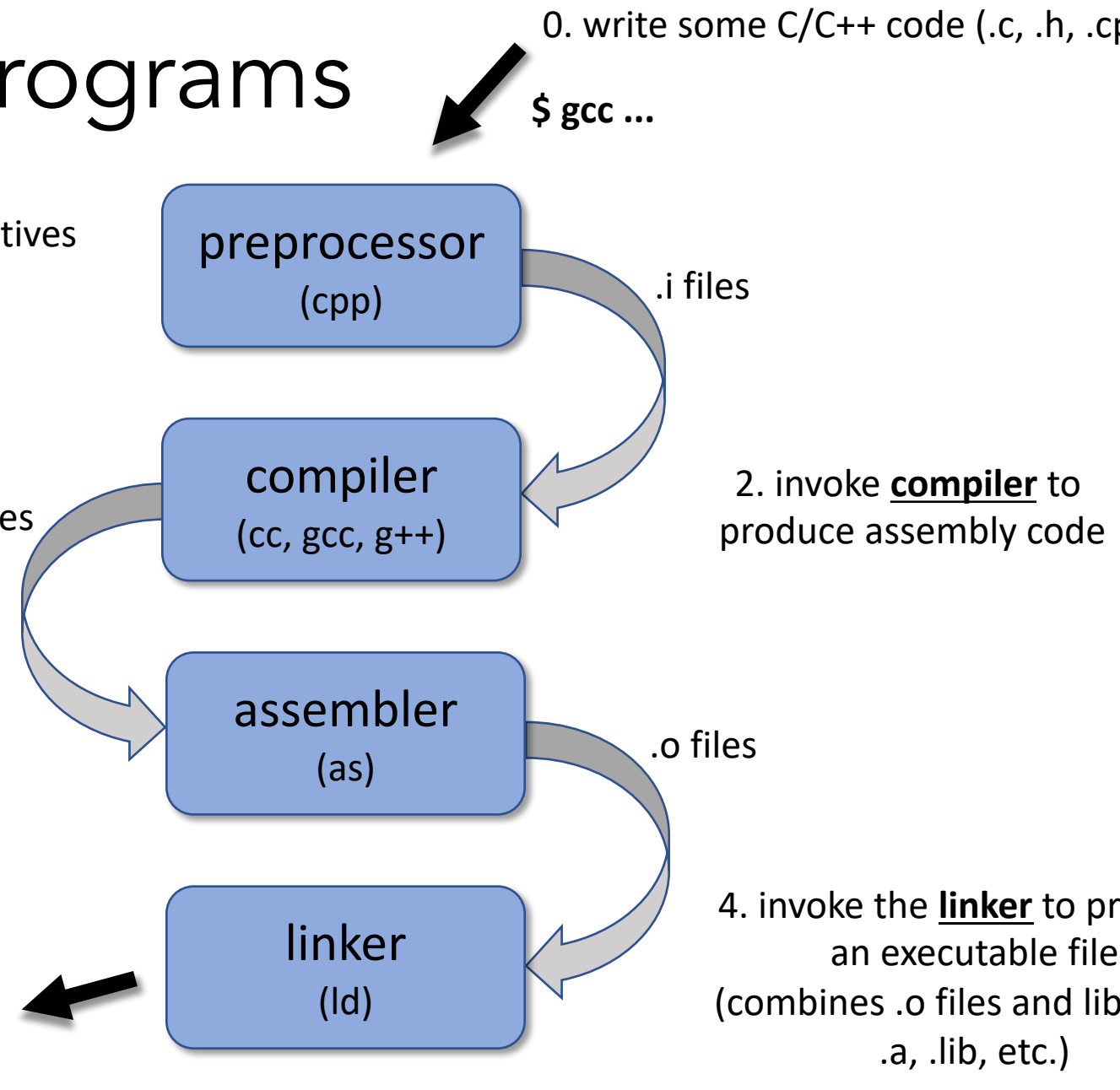
- `week02/`
  - `hello-world-1`
  - `hello-world-2`
  - `beer-song`
  - `secret (csci112 example)`
  - `sll`
  - `stategame*`

*Inspired by *Friends*

All the states in six minutes or Chandler's dumb states game

https://www.youtube.com/watch?v=22HXTrqn468

Compiling C Programs



Common `gcc` Flags

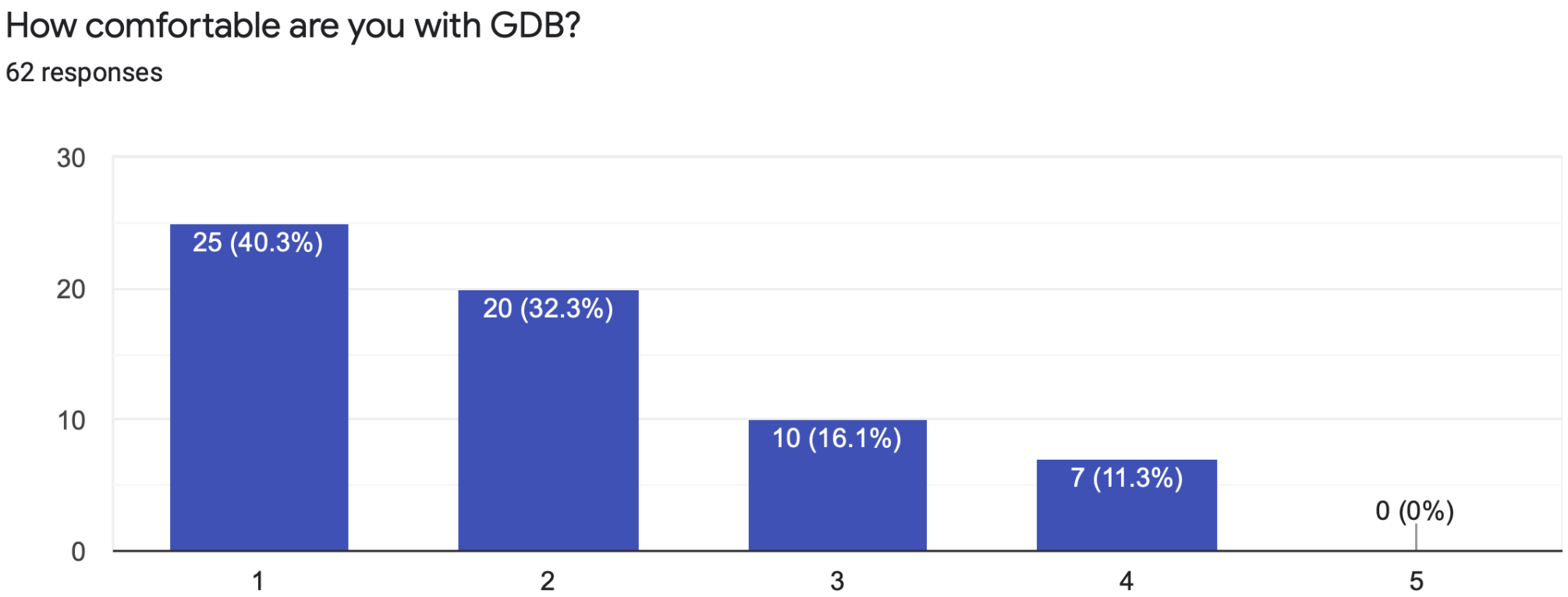
- `gcc -o` *`OUTFILE`* *set the name of the resulting executable (default = a.out)*
- `gcc -c` *compile but do not link (produces .o files)*
- `gcc -D`*`VAR`* *acts like* ***#define*** *in the source code; it sets the value of a symbol (default is 1)*
- `gcc -I`*`../headers`* *specify include file in a non-standard directory.*
- `gcc -l`*`name`* *link a library*
  - *NOTE: library "name" is linked (system search e.g., /usr/lib/lib**name**.a)*
  - *NOTE: this is a lowercase L ("ell") not an uppercase I ("eye")*
  - *NOTE: you can tell gcc to look in a non-standard location first with –L../libs/lib*
  - *NOTE: must be run at the end*

Also check out: `-Wall, -ggdb, -O`*`N`*`, -m32 -fno-builtin, ...`

Debugging C Programs with **gdb** (Task 7)

Because print statements can only get you so far... ☺

You Tell Me: Why Do We Need **gdb**?



Debugging w/ **gdb**

Why?

- **gdb** lets you run a program, stop execution within the program, examine and change variables during execution, call functions, and trace how the program executes.

How?

- You must compile executable with the **-ggdb** flag to generate w/ debug symbols

```
$ gcc -ggdb ... -o myprogram
$ gdb -q myprogram
(gdb) ...

# try some commands in gdb shell
- break
- run
- list
- info
- print
- step / next
- backtrace
- quit
```

See tutorial on **Debugging with GDB and Valgrind**:
https://www.traviswpeters.com/classes/debugging-gdb-valgrind/

A Note About "*Core Dumps*" On Our VMs

- What is a core file / core dump?
  A core file is an image of a process that has crashed. It contains all process information pertinent to debugging: contents of hardware registers, process status, and process data.
- In general:
  - We don't want to dump core files in a shared folder
  - We don't want to leave LOTS of core dumps around
  - We always dump core to the same place / same filename.
    (The only potential downside is that core files are overwritten each time you dump core)

```
# gdb program [core-dump]
# all VMs are setup to dump core to ~/core
$ gdb -q yalnix ~/core
(gdb) ...
```

Git & GitHub (Task 8)

Because we need a way to collaborate and version control our software!

Slides adopted from CSCI 366. Thanks to Prof. Mike Wittie!

Git & GitHub

- Git and Mercurial (hg) are distributed version control systems (DVCS)
- GitHub is a hosted version of git, but there are others (e.g., BitBucket, GitLab)

- Unlike centralized version control systems (CVS, SVN) distributed version control does not require checking out (locking) of files

Git Graphs

[master] 6c6faa5 My first commit - John Doe

[develop] 3e89ec8 Develop a feature - part 1 - John Doe

[develop] e188fa9 Develop a feature - part 2 - John Doe

[master] 665003d Fast bugfix - John Fixer

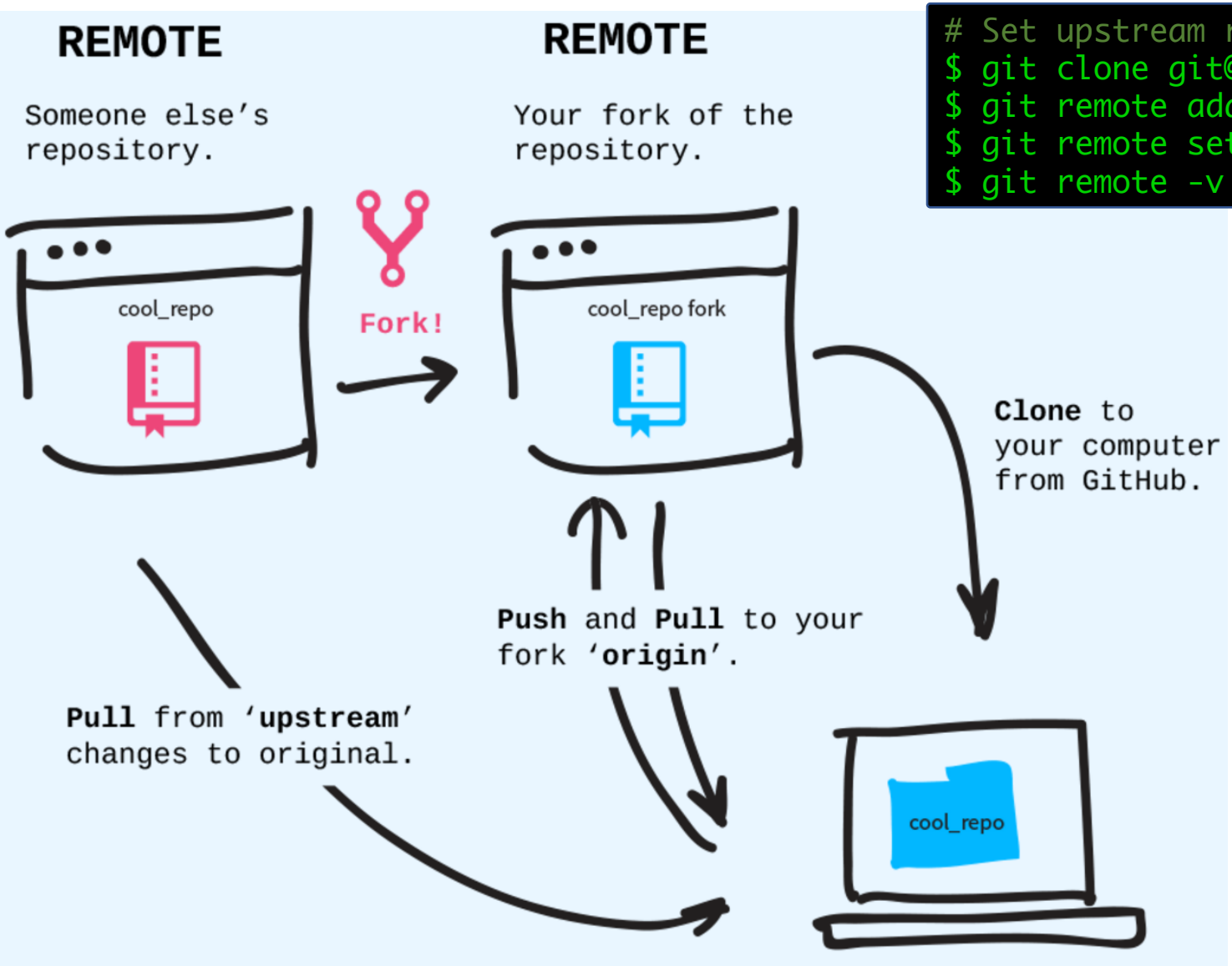
[myfeature] eaf618c New cool feature - John Feature

[master] 8f1e0e7 Merge branch `develop` into `master` - John Doe

[master] 6a3dacc Merge branch `myfeature` into `master` - John Doe

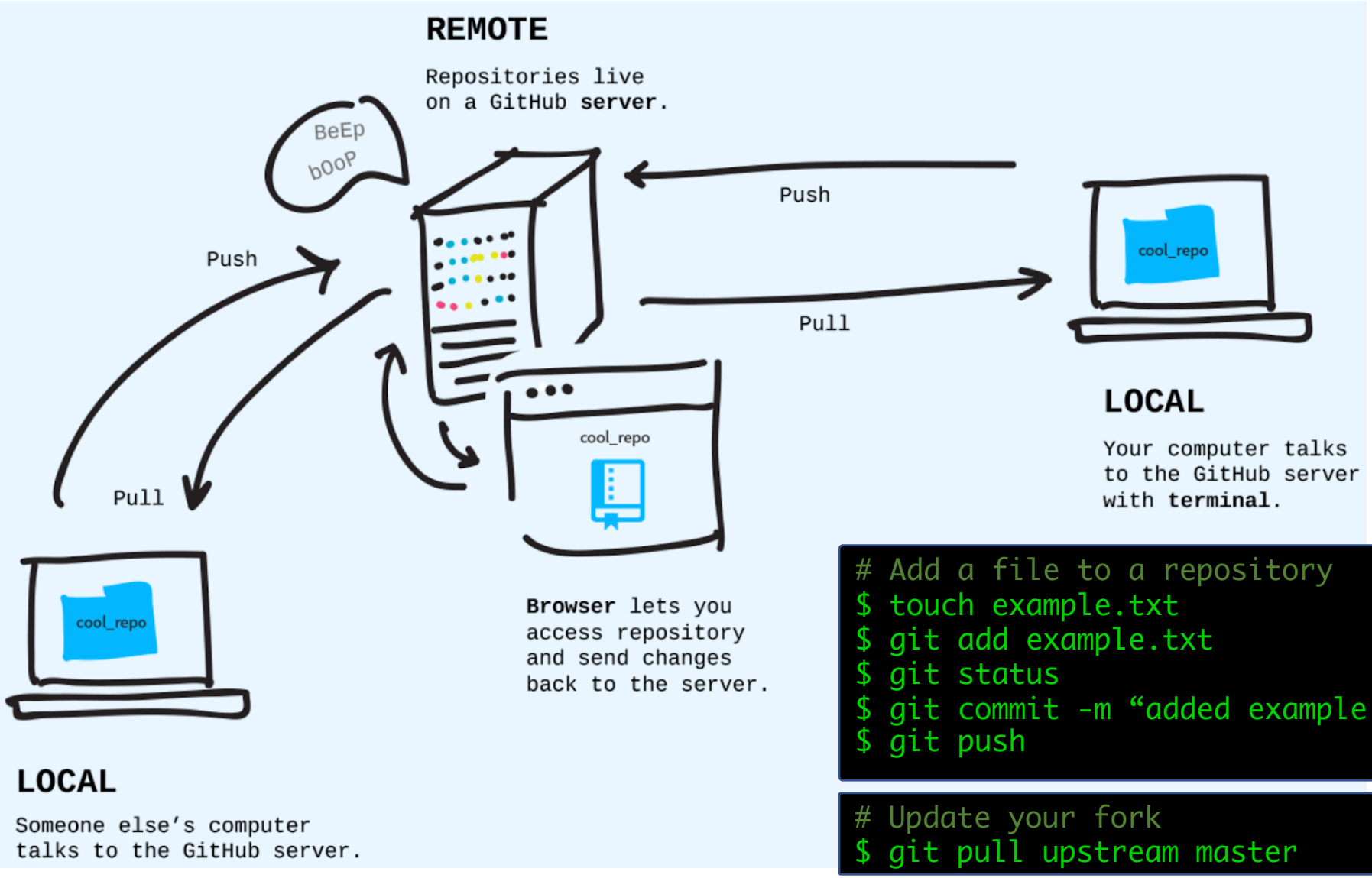
[master] abcdef0 Release of version 0.1 - John Releaser

Cloning vs Forking a Repository




```
# Set upstream remote
$ git clone git@github.com:USERNAME/YOUR-FROKED-REPO
$ git remote add upstream ORIGINAL-REPO
$ git remote set-url --push upstream no_push
$ git remote -v
```

Pushing and Pulling

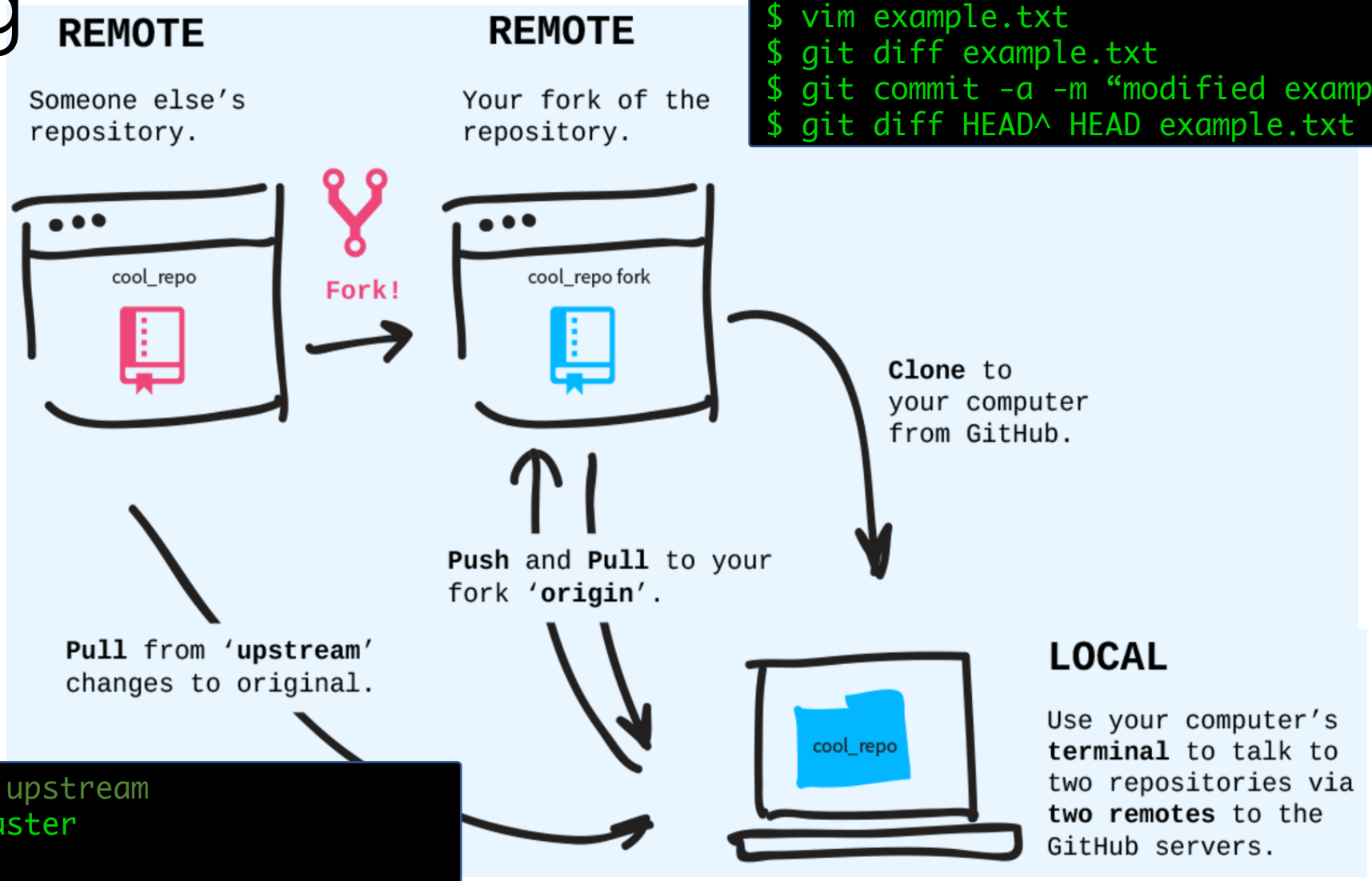



```
# Add a file to a repository
$ touch example.txt
$ git add example.txt
$ git status
$ git commit -m “added example.txt”
$ git push
```

```
# Update your fork
$ git pull upstream master
```

Updating

```
# Update your file
$ vim example.txt
$ git diff example.txt
$ git commit -a -m “modified example.txt”
$ git diff HEAD^ HEAD example.txt
```




```
# Merge a change from upstream
$ git pull upstream master
$ vim example.txt
$ git status
$ git commit -a -m “resolved a merge conflict”
$ git push
```

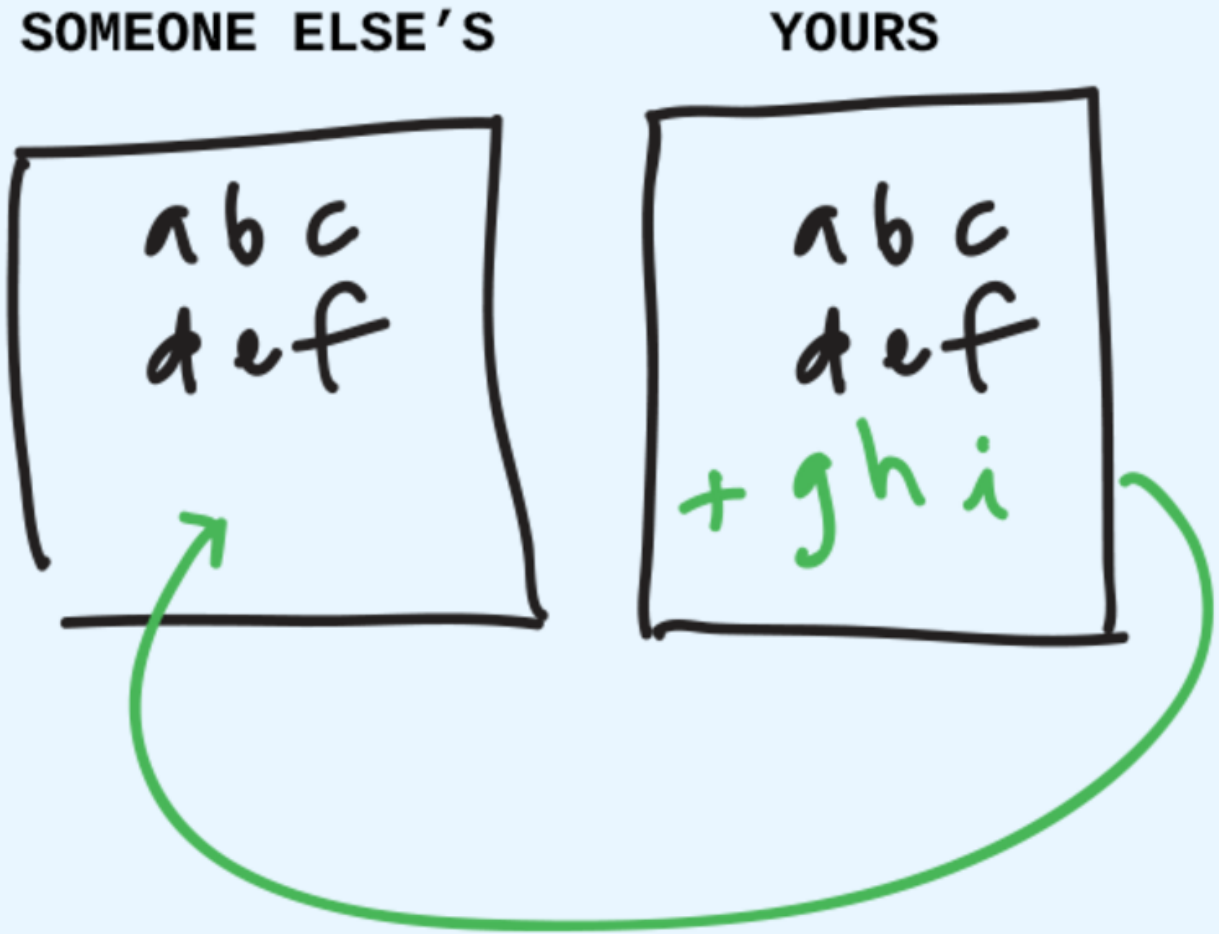
Pull Requests

Dear someone else,

I'd like to request you pull in the changes I've made to this branch.

As you can see I've made an addition that I believe many will find useful.

Thanks much,
Me



Go Forth & Git

- Work through Learn Git Branching https://learngitbranching.js.org/

- Other git commands to know:
  - **git branch** - create a new branch
  - **git checkout** - move HEAD to branch or a commit hash
  - **git merge** - merge two branches into a new commit
  - **git rebase** - change base (HEAD^)
  - **git reset** - undo changes to a previous commit
  - **git revert** - undo changes but save as a new commit
  - **git cherry-pick** - merge with specific commits
  - **git tag** - add a tag to a commit

(Recall Recommended Setup)

```
# Create a top-level folder on your HOST where you plan to do all CSCI 460 work
$ mkdir -p ~/projects/cs460/
$ cd ~/projects/cs460/

# Download private repo as a subdirectory, which contains the Vagrantfile
$  git clone YOUR-PRIVATE-REPO # git full command from GitHub

# Link your Vagrantfile to the top-level (all subdirectories synced to VM)
$ ln -s ~/projects/cs460/YOUR-PRIVATE-REPO ~/projects/cs460

# Run vagrant commands from top-level directory
$ vagrant up
$ vagrant ssh
$ vagrant halt
$ vagrant destroy
...
```