
OSU DevOps BootCamp Documentation

Release 0.0.1

**OSU OSL
OSU LUG**

October 30, 2018

1	Ready to Learn DevOps? <i>Lesson 0: Start Here</i>	3
2	DevOps BootCamp: Fall 2018	5
3	DevOps Open Office Labs	7
4	Schedule	9
4.1	Fall	9
4.2	Open Office Labs	9
5	Donate	11
5.1	Lesson 0: Start Here	11
5.2	Lesson 1: First Steps	15
5.3	Lesson 2: Operating Systems	21
5.4	Lesson 3: Docs & Communication	24
5.5	Lesson 4: Shell Navigation	31
5.6	Lesson 5: Users, Groups, Permissions	38
5.7	Lesson 6: Files	43
5.8	Lesson 7: Packages, Software, Libraries	48
5.9	Lesson 8: Version Control	51
5.10	Lesson 9: Programming	55
5.11	Lesson 10: Frameworks	62
5.12	Lesson 11: Testing	66
5.13	Lesson 12: Continuous Integration	69
5.14	Lesson 13: Security	71
5.15	Lesson 14: Databases	76
5.16	Lesson 15: Dev Processes & Tools	84
5.17	Lesson 16: DNS	89
5.18	Lesson 17: Configuration Management	94
5.19	Lesson 18: Application Isolation	100
5.20	Lesson 19: Cloud Infrastructure	103
5.21	Lesson 20: Contributing to Open Source	105
5.22	About	107
5.23	Setting up SSH	109
5.24	Setting up Docker	109
5.25	Schedule	110
5.26	Running DOBC	111

DevOps BootCamp (DOBC) is a free course hosted by the [OSU Open Source Lab](#). The course is dedicated to teaching core software development and systems operation skills to passionate OSU students and community members.

DOBC is always 100% free for in-person *and* online students.

Ready to Learn DevOps? *Lesson 0: Start Here*

DevOps Bootcamp's curriculum is available for you to learn at your own pace. Get started now!

DevOps BootCamp: Fall 2018

DevOps BootCamp is a single-day event with one track to help attendees kick off the year with fundamentals of system administration and software development. The hands-on workshop is designed to teach participants DevOps, a program development process that includes building, testing, and releasing software.

Please [register](#) if you're planning on attending BootCamp this fall.

DevOps Open Office Labs

Following the Fall Kickoff, we will be hosting bi-weekly open office labs in Milne 224 in two hour blocks. These labs initially will be structured around the content on the website but will aim to be mostly hands on. OSL students and staff will be on-site to help you work through the content. Content discussed at each lab will depend on the attendees interest. We will also be opening up our Milne server room for students to learn and interact with actual hardware.

Some interesting topics we may end up discussing include (expanded on the lessons we already have on the website):

- Automated Linux installs
- Network switch configuration
- Out of Band (IPMI) configuration and usage
- Installing servers in racks and configuring them
- Setting up servers to run various services (email, web, DNS, etc)
- Open Source software contributions

Our Milne server room includes three [OpenCompute](#) Racks donated from Facebook (includes a total of 90 compute nodes), managed network switches, and other various rack mounted server hardware.

Schedule

The DevOps BootCamp content is available for free but meet-space guided lectures are offered throughout the year. Check the schedule below for our in-person lectures; each lecture covers a different part of the curriculum covering the entire course during the OSU academic school year.

Warning: If you are working ahead be aware that the schedule and slides may be subject to change. Check back regularly.

Fall

Lessons Covered	Date/Time	Location	Description
0 - 7	Oct 27, 2018 9:30am-3:30pm	OSU KEC 1001	DevOps BootCamp Fall Kickoff

Open Office Labs

Each lab has two time slots (please choose one) to help assist with students being able to attend. All labs will in Milne 224.

Description	Slot 1	Slot 2
Lab #1	Oct 31, 2018 10am-12pm	Nov 1, 2018 2-4pm
Lab #2	Nov 14, 2018 10am-12pm	Nov 15, 2018 2-4pm
Lab #3	Nov 28, 2018 10am-12pm	Nov 29, 2018 2-4pm

Donate

We appreciate the help! To donate, go to <http://osuosl.org/donate>.

Lesson 0: Start Here

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

<p>Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our General Feedback Survey.</p>
--

About the Program



Definition: *System Administration*

- Responsible for systems (typically servers) running code, applications, and services
 - Keeping applications running (they crash, sometimes a lot)
 - Updates, Security
 - Monitoring, Logging
- Automates significant amounts of work with infrastructure
 - This enables a small team to administer hundreds or thousands of servers
- Involved in infrastructure architecture and decisions
- Can be involved in QA/Development work as well

Definition: *System Engineers*

- Responsible for creating the platforms code is run on
 - Work at a lower-level
 - Generally make infrastructure decisions for others
 - Often have expertise with some particular sub-system (networking, filesystems, etc)
 - Not necessarily on-call, but can be

- Sometimes intermixed with Systems Administrators who want Engineer in their title

Definition: *DevOps Engineers*

- Newer position
- Mix of Systems (Operations) and Development work
- Involved where the application and its platform meet
- Responsibilities include a mix of both Ops and Dev, usually:
 - General infrastructure/automation
 - Continuous Integration and Testing
 - Developer Environments/Workflow
 - Logging
 - Often on-call

Definition: *Site Reliability Engineers (SRE)*

- SRE and DevOps Engineers share the same foundational principles
- SRE is viewed as a “specific implementation of DevOps with some idiosyncratic extensions”
- SRE was originally created at Google as a process to improve managing their services
- Most large tech companies now follow SRE processes

DevOps

DevOps is a field which takes skills from **Software Development** and **Operations Engineering** to create and run applications more effectively.

TLDR: Development + Operations == Better Services

DevOps defines 5 key pillars of success:

1. Reduce organizational silos
2. Accept failure as normal
3. Implement gradual changes
4. Leverage tooling and automation
5. Measure everything

What *DevOps BootCamp (DOBC)* is

TLDR: Couch to DevOps in 1 school year

DOBC is a free education program offering:

- **Mentors** teaching DevOps related tools and concepts.
- **A challenge** for anybody willing to put in the effort.
- One-on-one **Apprenticeship**.

- **Hands-on** training and lectures
- **Free and Open Source** course materials!

What DOBC is *not*

DevOps BootCamp is not:

- A for-credit OSU class
- A Student job
- Easy

Why DOBC Exists

DOBC was created because the OSU OSL:

1. **Merged** with the school of EECS.
2. Wanted to help students **meet Company demands and expectations** of recent graduates.
3. Needed to **bridge the “Skills Gap”** of the OSU EECS curriculum.
4. Wanted to build a **DevOps Learning community**.

What You Will do

You will Learn:

- Linux systems
- Networking
- Software development
- Tools and why they matter

You will build:

- Functioning applications on the cloud
- Cloud infrastructures

Who Teaches DOBC

The teachers of DOBC include:

- OSL Students
- OSL Faculty
- Guests from *The Industry*
- You!

Bi-Weekly Open Office Labs

Discuss more advanced topics and also have hands-on with server and network equipment in our lab.

- Automated Linux installs
- Network switch configuration
- Out of Band (IPMI) configuration and usage
- Installing servers in racks and configuring them
- Setting up servers to run various services (email, web, DNS, etc)
- Open Source software contributions
- Software Development

The ‘Agreement’

You get out what you put in. DOBC is not meant to be easy. Stick with it, persistence is rewarded.

Student Benefits: A free education on industry topics, tools, and concepts

Student Responsibilities: Show up if you can, keep up if you cannot, put forth effort, and don’t forget to have fun.

Give us feedback.

- There will be a survey you, should take it.
- Honesty is the best policy.

Getting Involved

Where To Ask Questions

- Slack
- During Lecture and Hand-on Lessons
- *More on the [About](#) page...*

How To Ask Questions

- Always be respectful to those helping you.
- Stay calm and articulate.
- Explain you are trying to achieve and be thorough.

Next: *[Lesson 1: First Steps](#)*

Lesson 1: First Steps

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

<p>Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our General Feedback Survey.</p>
--

Vocabulary

A 10,000ft view of the world

General Topics:

- **Software:** A program that runs on a computer.
- **Operating System:** Computer software that manages other software.
- **GNU/Linux:** A *free* Operating System.
- **Computer Security:** Like physical security but harder to solve with a baseball bat.
- **Virtual Machine:** A computer emulated in software.
- **Containers:** Not virtual machines, but basically virtual machines.

Development:

- **Version Control:** A way to track changes and contributions to a project.
- **Continuous Integration:** Releasing updates *continuously*.

Buzzwords:

- **FOSS:** Free (and Libre) Open Source Software. Free as in Speech, not Free as in Pizza (but that too usually).
- **'The Cloud':** Computers somewhere else.
- **Docker:** Software that manages Linux containers

Exercise: What Vocabulary Do You Know?

- What other vocabulary can you think of related to DevOps?
- What about Silicon Valley, Programming, System Administration, etc?

Notation

- Variable (use whatever word you like here e.g., foo, bar, baz)

`$ONE_VARIABLE_NOTATION`

`<another notation for variables>`

- Literal (copy this exactly): `copy_me_exactly`
- Comments (parts of the code just for humans)

```
this_is(code)  # everything after the octothorp is a comment!
```

```
other_code(line)  // This can also be a comment. It depends on the  
                  // language!
```

Code-block:

```
#!/usr/bin/env python  
# This is a code block.  
# Most of the time you can copy this code and run it exactly as is.  
# It should be clear Where it 'goes' and how to run it based on  
# context.  
print('Hello world!')
```

```
# Copy the text after '$' into your terminal & press enter.  
$ echo Hello World
```

Exercise: Reading Examples

Trick question: how would you read this

```
#!/bin/python  
dogs = ['$BREED_ONE', '$BREED_TWO', '$BREED_THREE']  
for breed in dogs:  
    print(breed)
```

Actually prints...

```
$BREED_ONE  
$BREED_TWO  
$BREED_THREE
```

Answer: Reading Examples

Replace the \$BREED_N with actual dog breeds.

```
#!/bin/python  
dogs = ['corgie', 'pug', 'french bulldog']  
for breed in dogs:  
    print(breed)
```

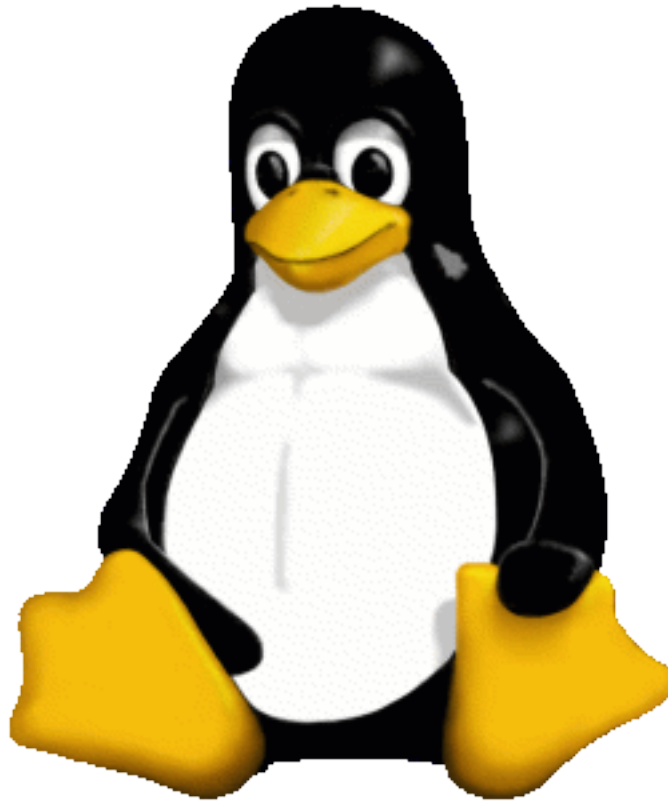
Actually prints...

```
corgie  
pug  
french bulldog
```

SSH: Secure Shell

- Secure Shell (SSH) provides a secure channel to access a Linux machine remotely via command line.
- It's a primary tool for almost every DevOps engineer
- Designed as a replacement to Telnet which provides unsecured remote shell access
- Allows for password logins and private/public key-based logins which are more secure
- Some tricks you can do with SSH
 - Run a single command remotely
 - Secure file transfer (via `scp` or [WinSCP](#))
 - Port forwarding, SOCKS proxy or tunnel
 - [SSHFS](#) – userspace filesystem which uses SSH

Getting Setup on Linux



There are a variety of ways to run Linux!

- Dual-boot Windows+Linux
- Virtual Machine (VMWare, Virtualbox, cloud server, etc)
- Container (Docker)
- Windows Linux Subsystem

Docker Setup

We suggest you [install Docker](#) and [Docker Compose](#), a tool which makes it easy to run small [Linux Containers](#) on your system in a safe sandbox without requiring to install Linux on your own machine. This is the same setup we used in the lecture.

Make sure you read the install documentation for Docker to ensure your system supports running it and have the required BIOS settings enabled.

After you have it installed, run this to start a container:

```
$ git clone https://github.com/DevOpsBootcamp/Bootcamp-Exercises.git
$ cd Bootcamp-Exercises
$ docker-compose up -d
$ docker-compose run -p 8080:8080 dobc bash
```

You can log out by typing `exit` and then enter which will stop the container.

To stop the container, run the following:

```
$ docker-compose kill
$ docker-compose rm --all
```

Feel free to try other Docker images, some that we recommend include:

- ubuntu
- debian
- centos
- fedora

To run those, do the following:

```
$ docker run -it --rm <docker image name> bash
```

You can find have more images at the [Docker Hub](#). We also recommend you read [Getting started with Docker](#) to have a better understanding of how it works.

Virtual Machine Setup

Instead of using Docker, you can also run a Linux [Virtual Machines](#) on your computer. This will give you a full Linux environment as if it were on a real machine.

We suggest you [install Vagrant](#), a tool which makes it easy to run and acquire [Virtual Machines](#).

You may also need to [install VirtualBox](#) or [install VMWare](#) (Requires TEACH access) a tool necessary for Vagrant to function.

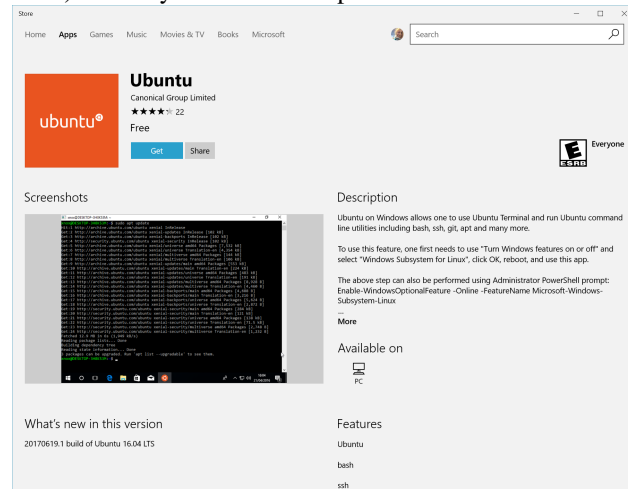
After you get Vagrant and either VirtualBox or VMWare installed, clone our vagrant repo (make sure you [install Git](#) first!) and then start the VM:

```
$ git clone https://github.com/DevOpsBootcamp/vagrant.git
$ cd vagrant
$ vagrant up
$ vagrant ssh
```



Windows Subsystem for Linux Setup

The [Windows Subsystem for Linux](#) (Bash on Windows) allows you to run userspace Linux software on Windows,



while using less resources than a virtual machine. If you installed the Fall Creators Update for Windows 10, you can install one or more Linux distributions through the Windows Store.

Exercise: Change Your Password!

Challenge *Change your password on your Linux machine.*

```
$ passwd
Changing password for user <user>.
Changing password for <user>.
(current) UNIX password: # Enter old password, hidden
New password: # Enter new password, also hidden
Retype new password:
passwd: all authentication tokens updated successfully.
```

Don't forget: when you login next time, use the *new* password you just set.

Further Reading

- More information on [Linux Containers](#) and [Virtual Machines](#).
- **'Install Putty'** if you want to access a remote Linux box.
- [Install Docker](#) if you want to run a local Linux container
- [Install Vagrant](#) if you want to run a local Linux Virtual machine.
- [Install VirtualBox](#) in addition to Vagrant for local virtual machines.
- [Install VMWare](#) in addition to Vagrant for local virtual machines.
- [Windows Subsystem for Linux](#) if you want to use Linux without VM overhead.

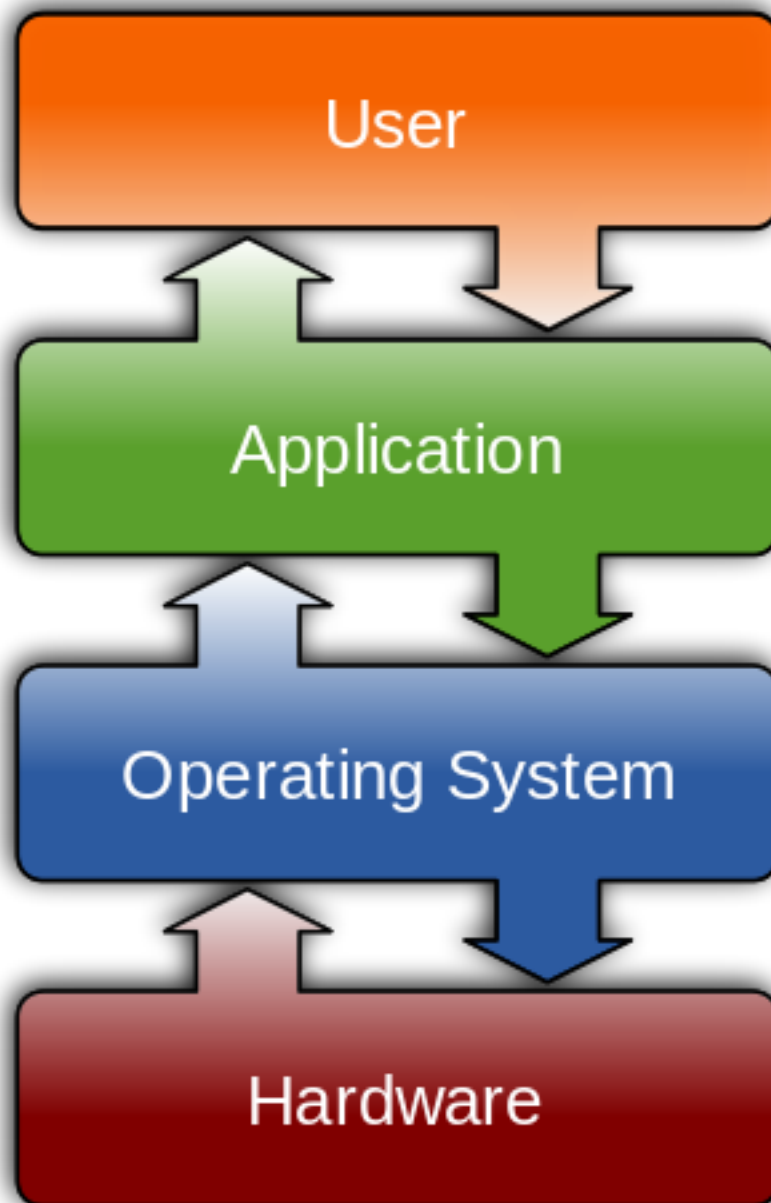
Next: [Lesson 2: Operating Systems](#)

Lesson 2: Operating Systems

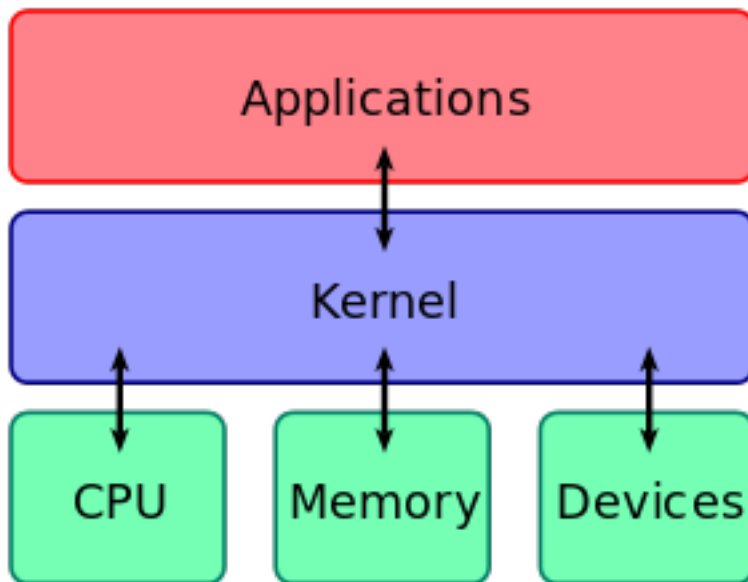
Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

What an Operating System is



Anatomy of an OS



- **User Interface:** What you interact with. Window Managers for instance.
- **Application Layer:** What developers use to make software run.
- **Kernel:** *The Core of the OS.* Makes communication between hardware and applications sane.
- **Hardware:** What does the actual computations. The thing your keyboard is plugged into.

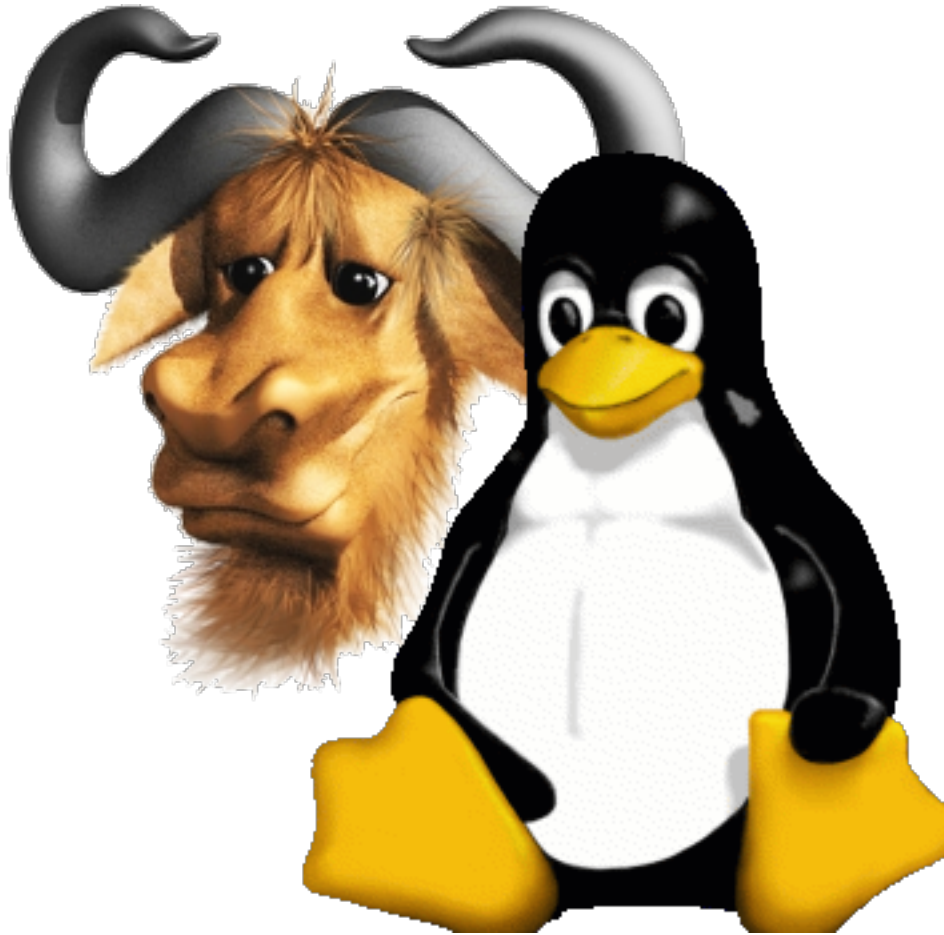
Types of Operating Systems

Popular Operating Systems

- UNIX
 - Linux
 - * Android
 - * Debian
 - * RHEL
 - MacOS / Darwin
 - FreeBSD
- Windows

GNU/Linux

Welcome to the Family



Flavors of Linux

- **Debian**
 - Ubuntu
 - * LinuxMint
- **RedHat**
 - RHEL
 - Fedora
 - Centos
- **Gentoo**
 - ChromeOS
- **Slackware**
- **ArchLinux**

Exercise: Pop Quiz

1. What are some different types of Operating Systems?

2. What constitutes a ‘Distribution’ of Linux?
3. How is Linux different from Windows? OSX?
4. How is Debian different from Gentoo?

Further Reading

OSU Courses:

CS 312: Linux System Administration

- DOBC in class form
- Not currently offered, however course content is online
- <http://cs312.osuosl.org>

OSU Courses:

CS 344: Operating Systems I

- Required course for all CS Students at OSU.
- **Covers fundamentals of low-level programming concepts.**
 - Multi-threaded programming
 - Read / Write operations
 - Socket programming

OSU Courses:

CS 444: Operating Systems II

- Required course for all CS Students at OSU.
- **Covers kernel hacking and low-level OS design.**
 - IO / Process scheduling
 - Building kernel modules
 - Memory management

Free Online Resources: OSDev.org is a wiki dedicated to helping people develop their own operating systems. It’s a big leap from this lesson, but great if you’re interested in learning the nitty-gritty.

[Operating Systems Design and Implementation](#) by Andrew S. Tanenbaum is a classic in the world of OS Development. It’s also a big leap, but can teach you more about how Operating Systems work than you ever thought there was to know.

Next: *Lesson 3: Docs & Communication*

Lesson 3: Docs & Communication

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

<p>Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our General Feedback Survey.</p>
--

When in doubt

```
$ <program> --help
$ <program> -h
```

Most programs allow you to pass a help flag which will print out basic usage. This is useful as a quick reference for how to use the program.

Man Pages

```
$ man <program>
```

- Type / and then enter a keyword to see where that word appears.
- Press n to go to the next (and p to go to the previous) occurrence of that word.

```
$ man man
```

```
MAN(1)                      Manual pager utils                      MAN(1)
```

```
NAME
```

```
man - an interface to the on-line reference manuals
```

```
SYNOPSIS
```

```
man [-C file] [-d] [-D] [--warnings[=warnings]] [-R
encoding] [-L locale] [-m system[,...]] [-M path] [-S list]
[-e extension] [-i|-I] [...]
```

```
DESCRIPTION
```

```
man is the system's manual pager. Each page argument given
to man is normally the name of a program, utility or
function. The manual page [...]
```

Anatomy of a Man Page

Most Man Pages include:

- Name
- Flags
- Description
- Basic Usage
- Authors

If you're lucky they will also include:

- A Good description
- Advanced Usage.
- Examples
- History
- See Also

Sections of Man

man pages are also organized by *section*. To read man page for a program/library in a specific section type `man # <program or library>` where # is the section number.

For instance:

```
$ man 2 open # Displays the kernel documentation for open (section 2)
$ man open   # Displays the documentation for openvt (section 1)
```

If there is a collision in man-page naming (like `open` and `open()`) man will pick the page which appears in the lowest-value section.

1. Executable programs or shell commands
2. System calls (function provided by the kernel)
3. Library calls (functions provided from within libraries)
4. Special files (usually found in `/dev`)
5. Files formats and conventions eg `/etc/passwd`
6. Games
7. Miscellaneous (including macro packages and conventions), e.g., `man(7)`, `groff(7)`
8. System administration commands (usually only for root)
9. Kernel routines [Non standard]

Note: Some distros use `info` instead of `man`. To learn more about the `info` command, see Further Reading.

Project Docs

Projects also document themselves *beyond* the manpage. These can include tutorials, a README, and Q&A. If you need more information about a tool or a specific answer these docs will probably be your best bet.

These docs may also answer any technical or contributing questions. These docs can be updated more frequently than local man pages so should also be referred to for bleeding-edge information.

Where to look:

- <http://docs.some-random-project.io/>
- <http://some-random-project.io/docs/>
- <http://organization.com/some-random-project/>

Communication

Communication is very important for DevOps engineers. Whether they are talking to their own team or working either external projects they use.

It's important to be familiar with the chat platforms that these projects use which include:



- Internet Chat Relay (IRC)
- Slack
- Mailing lists
- Discourse
- Forums

IRC

Quick Facts:

- Internet Relay Chat (IRC)
- Very old (RFC 1459, May 1993)
- Works on everything (Terminal, GUI, Web-browser, etc)
- The people you want to listen to are there
- Oregon State ran one of the first servers ever!

```
Oregon State University Linux Users Group | Tuesdays 18:00 KEC 1007 | http://lu
preempting a process.
15:00 < radens> I also suspect that the duplicated kernels in guest VMs will be
harder on the cache too.
15:00 < radens> But VMs provide a strong security boundary, unlike containers.
15:02 < radens> (until you need guests to do something like share a GPU, that's
a great way to hack sister VMs)
15:03 < radens> And Hypervisors can easily be faster and have a lighter
footprint than you think, especially with some widespread
hardware features like VPIDs.
15:05 <+pop> Skimming the article it basically says that unikernels (which are
as close to a container as you can get as a VM) have comparable or
even better performance do stock Docker containers plus security
benefits.
15:05 <+pop> * at scale.
15:07 < radens> Yeah, every once and a while unikernels come up. Here's an
article suggesting that unikernels are unfit for production:
15:07 < radens> https://www.joyent.com/blog/unikernels-are-unfit-for-production
15:07 < ocelbot> radens's shortened url is http://bit.ly/2AcAvTA
15:07 < ocelbot> Title: Joyent | Unikernels are unfit for production
15:10 < radens> One quote sticks out: "[with unikernels] the principle of least
privilege is violated: any vulnerability in an application
[17:26 [Act: 1,2,3,4,5,7,9,10,15,16,17,18,19,22,23,27,28,29,30,35,36,37,39,41]
[#osu-lug]
```

Exercise: Getting on IRC

To get on IRC, Use *irssi* or *weechat* in screen:

```
# This step is optional, but persistent IRC is cool
$ ssh <username>@<a remote linux server>

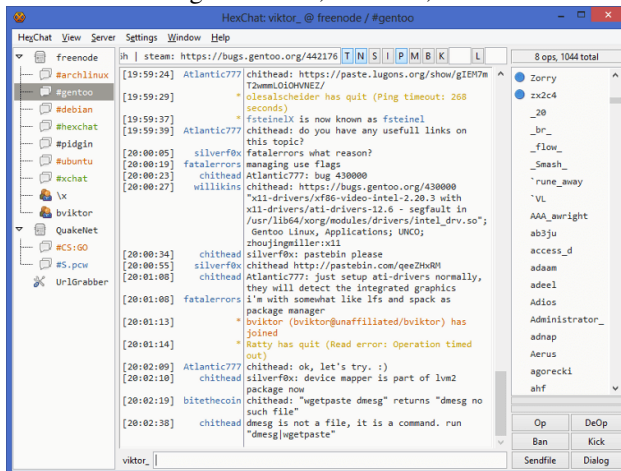
# start screen with the name 'irc'
$ screen -S irc

# start your client in the 0th window of the screen session
$ irssi
# or
$ weechat-curses

# exit irc screen with CTRL+a, CTRL+d
# exit ssh session with CTRL+d or 'exit'
# to get back to irc:
$ ssh <username>@<preferred shell host>
$ screen -dr IRC
```

Other IRC Clients

If you're not interested in using the command line there also an assortment of *graphical* IRC clients including [Hexchat](#), [MIRC](#), and [KiwiIRC](#). Look those up if you're interested in them.



There are also a variety of mobile clients for each platform that work well enough. You can also use a mobile SSH client and connect to your server in a pinch.

Unfortunately IRC isn't very mobile friendly.

Connecting and Setup

In the IRC client run these commands (irssi):

```
/connect irc.freenode.net
/nick <myawesomenickname>
/msg nickserv register <password> <email>
/nick <myawesomenickname>
/msg nickserv identify <password>
/join #devopsbootcamp
```

For weechat, do the following:

```
/server add freenode irc.freenode.net
/connect freenode
```



```

/nick <myawesomenickname>
/msg nickserv register <password> <email>
/nick <myawesomenickname>
/msg nickserv identify <password>
/join #devopsbootcamp

```

Commands and Tips

Command	Description
/list	Reports all the channels on a server.
/topic	Reports current channel topic.
/names	Reports nicks of users in channel.
/join <channel>	Join a new channel.
/whois <nick>	Learn about a person.
/msg	Directly message an individual.
/help <command>	Provides help for commands

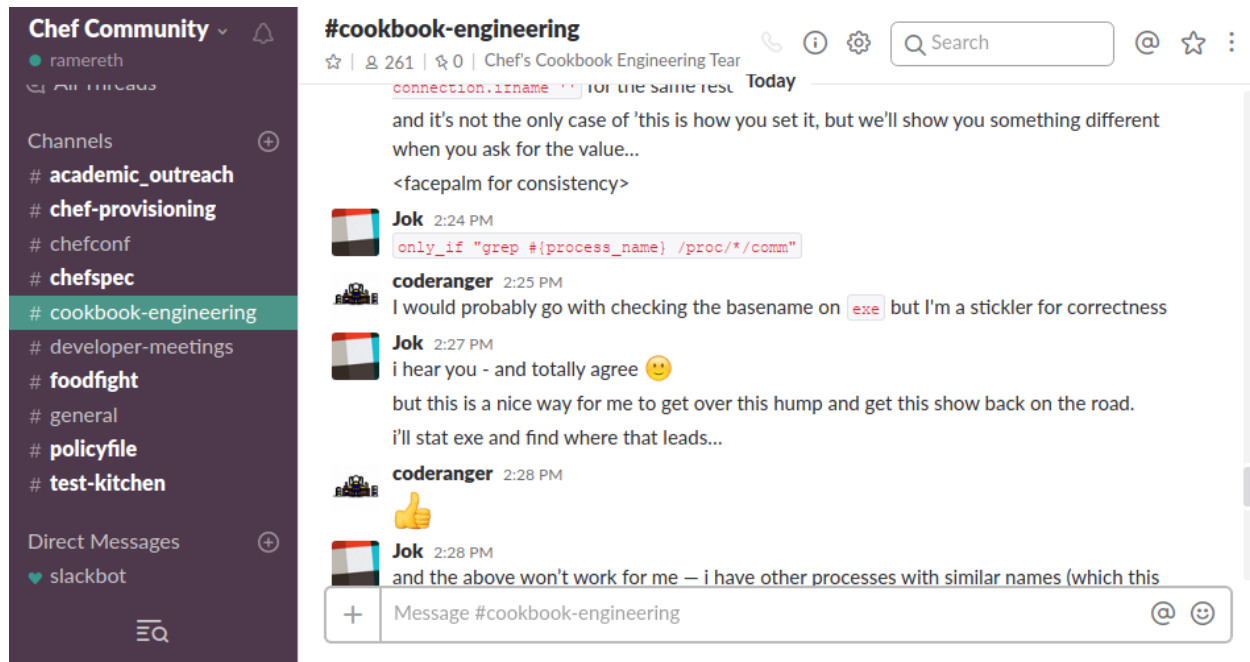
- Tab-completion works with nicks
- You get a **hi-light** when your name is said.
- Symbols (@, +) are not part of names, show status in channel.
- chanserv and nickserv are robots.
 - /msg nickserv help to get nick help.
 - /msg chanserv help to get channel help.

IRC Jargon

Term	Description
channel	Chat rooms with with '#' prefixed in front of their names
ping/pong	'I would like to tell you something.' / 'I'm here, tell it to me.'
tail	~
hat	'@' Denotes admin status in a channel.
nick	Your name.
netsplit	When the IRC servers lose connection with each other.
kick/ban/k-line	Force someone off the channel or server, typically for abuse

Slack

Modern messaging platform which featureful desktop and mobile clients



- Launched in 2013 and stands for “*Searchable Log of All Conversation and Knowledge*”
- Has many IRC like features, with additions such as rich text and emojis
- Proprietary platform, however there are several open source “clones” that can be self hosted
- “New kid on the block” – Many new projects prefer Slack over IRC
- Join our Slack team! <http://devopsbootcamp.slack.com>

Asking for Help

It’s okay to ask for help. Here are some things to keep in mind:

1. Ask yourself what should be happening?
2. Ask yourself what is actually happening?
3. Google the problem(s).
4. Skim the manuals of each component.
5. Identify a friend, mentor, or IRC/Slack channel who could help.
6. When they’re not busy, give them a quick synopsis of points 1 and 2, mentioning what possibilities you’ve ruled out by doing steps 3 and 4.

Contributions = expertise + time

Further Reading

- [About info](#): `info` is an alternative to `man` that some distros use instead.

Next: [Lesson 4: Shell Navigation](#)

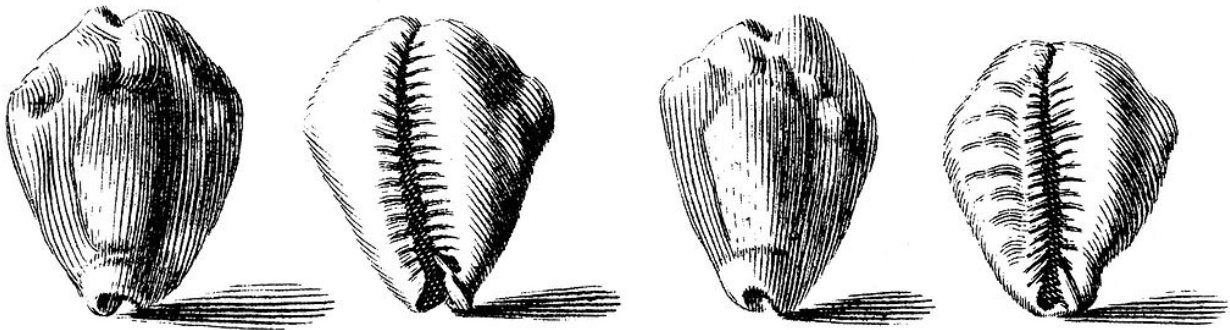
Lesson 4: Shell Navigation

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

The Shell

A shell is a text-based user-interface for a computer.



Shell Examples

sh	Required by all POSIX Operating Systems.
bash	Default on most GNU/Linux-based Operating Systems.
csh	Default shell on most BSD (Unix) based Operating Systems
zsh	<i>The hip new shell on the block.</i>
fish	<i>Yet another hip new shell on the block.</i>

Navigation Concepts

Basic Shell Commands

```
# Prints the current working directory (where you are)
$ pwd
# Prints the contents of the current working directory
$ ls
# Navigates to a new directory.
$ cd <path/to/other/directory>
# Prints a string to the screen.
$ echo "some thing $AND_VARS"
# Prints the contents of a file(s) to the screen.
$ cat foo.txt bax.txt
# Searches 'file.txt' for the string 'foo'
$ grep foo file.txt
# Prints a file to the screen so you can arrow up/down.
$ less file.txt
# Prints environment variables to the screen.
```

```
$ env
# Prints out current user
$ whoami
# When in doubt, always type help.
$ help
```

Shell Scripts

about_me.sh

```
#!/bin/sh
if [ $(whoami) == "root" ]; then
    echo "You're root!"
else
    echo "Your username is $(whoami)"
    echo "Your home-directory is $HOME"
    echo "Your current directory is $PWD"
    echo "Your computer's host-name is $HOSTNAME"
fi
```

Invoke with:

```
# Tell Linux that this can be run as a program
$ chmod +x about_me.sh
# Invoke the script.
$ ./about_me.sh
Your username is dobc
Your home-directory is /home/dobc
Your current directory is /home/dobc
Your computer's host-name is dobc
```

Useful Symbols

```
$ grep 'searchstring' files/* | less

$ true || echo 'never gets here'
$ false && echo 'never gets here'

$ echo 'this now an error message' 1>&2 | grep -v error
this is now an error message

!$ # last argument to last command
$ cat /dir
cat: /dir/: Is a directory
$ cd !$
cd /dir
$ pwd
/dir
```

More Useful Symbols

```
$ for x in 1 2 3; do echo $x; done # Use seq for longer sequences
1
2
```

3

```
$ var='this is a var'; echo ${var//this is } # Deletes 'this is '
a var
```

```
$ ls -l `which bash`
-rwxr-xr-x 1 root root 1029624 Nov 12 15:08 /bin/bash
```

Combining These Together

```
$ set -a blocks
$ blocks="10.0.0.0/24"
$ set -a ips
$ ips=`fping -g 10.0.0.0/24 2>&1 | grep unreachable | tr \\ \\n`
$ for ip in $ips; do
$   nmap -p 22 $ip && ips=`echo ${ips//$ip} \
$   | tr -s \\n`
$ done
$ echo $ips
```

Function Definitions

```
name () {
# code goes here
}
```

Internal Variables

You should know the following:

Variable	Meaning
\$*	All arguments passed
\$?	Return code of last command run
"\$@"	All arguments passed as a list
\$CDPATH	Colon-delimited list of places to look for dirs
\$HOME	Location of user homedir
\$IFS	Internal Field Separator
\$OLDPWD	Previous PWD

Internal Variables

Variable	Meaning
\$PATH	Colon-delimited list of places to find executables
\$PWD	Present Working Directory
\$SHELL	Path to running shell
\$UID	User ID
\$USER	Username

You should also read the EXPANSION section of the bash man page.

File Paths

.	The current directory
..	The parent directory
~	Alias for your home directory
/	Separates directories: one_dir/another_dir/last_dir Alone, or at the start of a path, it is the root directory.

```
$ tree -F
.
-- bar/
|   -- one/
|   -- two
-- baz/
-- foo/
    -- a/
        -- b
5 directories, 2 files
```

Special Characters

Wildcard (*) Used as a stand-in for any character(s).

Example: `cat *.log` cats all files in the current working directory ending in `.log`.

End of line (\$) Used to specify the end of a regex. We'll cover what regex is later.

Curl braces ({ }) Used to specify a set.

Example: `ls {foo,bar,baz}ley-thing` expands to `ls fooley-thing barley-thing bazley-thing`

Escape special characters (treat them as normal characters) with the escape character (`\`).

Type Less, Tab More

Pressing the `tab` key auto-completes a command, file-path, or argument in your shell.

Pressing `tab` multiple times completes the command to the best of the shells ability and then lists the possible completions (if there are any).

```
$ ls b      # <tab>
$ ls ba     # <tab>
bar_thing/ baz_thing/
$ ls bar    # <tab>
$ ls bar_thing
```

Text Editor: Nano

```

GNU nano 2.2.6          New Buffer          Modified
I am typing in nano!
I can hold control and press o to save (WriteOut)

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
  
```

- User types like normal.
- Arrow keys used to to navigate the cursor.
- ^ + <key> Commands (control + key)

Nano is a great terminal text editor to start with. Later in your career you may start using emacs or vi/vim but to start with nano is familiar, easy to use, and gets the job done.

To use nano simply execute it like any other command in the terminal.

```

$ nano                # Open with empty file
$ nano <file_name>    # Edit a specific file
  
```

This editor is almost exactly like any word processor or plain-text editor except that you don't have a mouse – only keyboard shortcuts. The instruction bar at the bottom of the screen is explains all of the key-bindings from saving, to exiting, to cut and pasting.

Bash Hello World

Using nano create a file called `hello_world.sh` and put the following in it:

```

#!/bin/bash
# declare STRING variable
STRING="Hello World"
# print variable on a screen
echo $STRING
  
```

Now make the script executable using `chmod` and run the script. What does it do?

```

$ chmod +x hello_world.sh
$ ./hello_world.sh
Hello World
  
```

Passing arguments to the bash script

When you pass arguments to a bash script, you can reference them inside of the script using `$1`, `$2`, etc. This means if you do something like `foo.sh bar`, `$1` will return `bar`. You can also use `$@` to reference all arguments.

For example:

```
#!/bin/bash
echo $1    # prints argument #1 given to script
echo $@    # prints all arguments given to script
```

Reading User Input

You can also take input using the `read` command which then stores it in a variable. If you pass `read -a`, it puts the input into a bash array.

For example:

```
#!/bin/bash
echo -e "Tell me a word: \c"
read word
echo "Your word is $word"
```

```
$ ./read.sh
Tell me a word: foo
Your word is foo
```

Simple Bash if/else statement

Bash conditionals use `if`, `else`, `then` and `fi` operators. You can compare strings, files and even command output. An example:

```
#!/bin/bash
if [ "$1" == "foo" ] ; then
    echo "You said $1"
else
    echo "You did not say foo"
fi
```

```
$ ./sayfoo.sh foo
You said foo
$ ./sayfoo.sh bar
You did not say foo
```

Exercises: Bash

- Using the `tar` command, write a script named `backup.sh` which backs up the `dobc` user home directory into a file `/tmp/dobc-backup.tar.gz`. *Hint: use the man page for `tar` or type `'tar -h'`.*
- Create a script called `args.sh` that takes three arguments and prints all of the args and then prints them in reverse using `echo`.
- Create a script named `input.sh` which takes input for three args and then prints them in a sentence (of your choosing).

- Create a script named `ifelse.sh` which takes two arguments. If both arguments match, print "Yay, they match!", if they don't, then print "Boo, they don't match :(".

Exercise Answer Key

Simple Backup script

```
#!/bin/bash
# The flags for tar do the following:
# v - verbose
# c - compress
# z - use gzip
# f - output to file
tar -vczf /tmp/dobc-backup.tar.gz /home/dobc
```

```
$ ./backup.sh
tar: Removing leading '/' from member names
/home/dobc/
/home/dobc/backup.sh
/home/dobc/hello_world.sh
/home/dobc/.bash_profile
/home/dobc/.bashrc
/home/dobc/.bash_logout
```

Bonus: How could you list the contents of the file?

Passing arguments to the bash script

```
#!/bin/bash
echo $@
echo $3 $2 $1

$ chmod +x args.sh
$ ./args.sh DOBC is awesome
DOBC is awesome
awesome is DOBC
```

Bonus #1: What happens if you give the script nothing? Bonus #2: What happens if you give it the string "DOBC is awesome" with quotes?

Reading User Input

```
#!/bin/bash
echo -e "Tell me a noun: \c"
read noun
echo -e "Tell me a verb: \c"
read verb
echo -e "Tell me an adjective: \c"
read adj
echo "I plan to $verb a $adj $noun"

$ ./input.sh
Tell me a noun: apple
Tell me a verb: eat
```

```
Tell me an adjective: large
I plan to eat a large apple
```

Simple Bash if/else statement

```
#!/bin/bash
if [ "$1" == "$2" ] ; then
    echo "Yay, they match!"
else
    echo "Boo, they don't match :("
fi

$ ./ifelse.sh foo foo
Yay, they match!
$ ./ifelse.sh foo bar
Boo, they don't match :(
```

Further Reading

BASH Programming - Introduction HOW-TO A free online resource of learning bash programming. Covers some concepts we'll get to later in DOBC, but a good resource to have on hand.

Advanced Bash-Scripting Guide An in-depth exploration of the art of shell scripting. Covers more advanced concepts with Bash.

Running rm -rf / on Linux This video demonstrates what happens when you 'delete your hard-drive' on Linux. A fun watch!

Next: *Lesson 5: Users, Groups, Permissions*

Lesson 5: Users, Groups, Permissions

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

What are users?

You, right now.

```
$ whoami      # your username
$ who         # who is logged in?
$ w           # who is here and what are they doing?
$ id          # user ID, group ID, and groups you're in
```

Not just people: Apache, Mailman, ntp. "system users"

Users have

- Username
- UID
- Group
- Shell
- Usually (but not always) password
- Usually (but not always) home directory

/etc/passwd:

```
root:x:0:0:root:/root:/bin/bash
username:password:uid:gid:uid info:home directory:shell
```

Managing Groups and Users

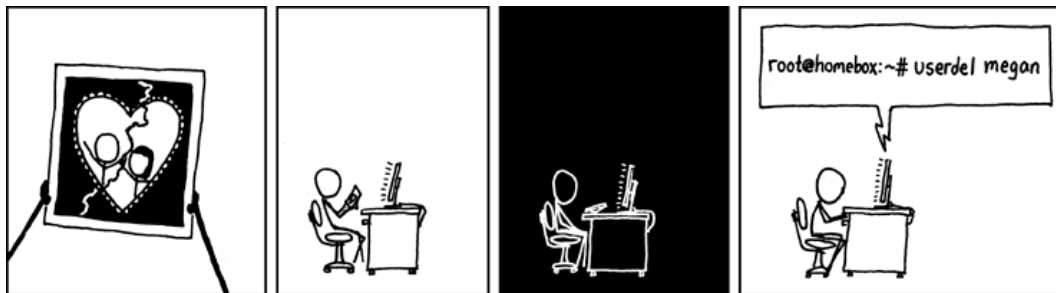
As someone interacting with servers, even as a developer, it's necessary to understand how to manage users and groups on a Linux machine.

To view all user information on a system check the file /etc/passwd:

```
$ cat /etc/passwd
# username:x:UID:GID:GECOS:homedir:shell
```

To add, delete, and change the password of a user respectively run the following commands:

```
$ useradd <user_name>  # vs adduser, the friendly Ubuntu version
$ userdel <user_name>
$ passwd
```



What are groups?

To add a group, or the permissions of a user/group run `groupmod`, `usermod`, and `groupmod` respectively. Similarly to /etc/passwd, /etc/group carries group information.

```
$ groupadd
$ usermod
$ groupmod
$ cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
```

```
adm:x:4:
tty:x:5:
# group name:password or placeholder:GID:member,member,member
```

Users won't be active in new group until they "log back in"

Passwords

/etc/shadow, **not** /etc/passwd

```
user@localhost ~ $ ls -l /etc/ | grep shadow
-rw-r----- 1 root shadow 1503 Nov 12 17:37 shadow
```

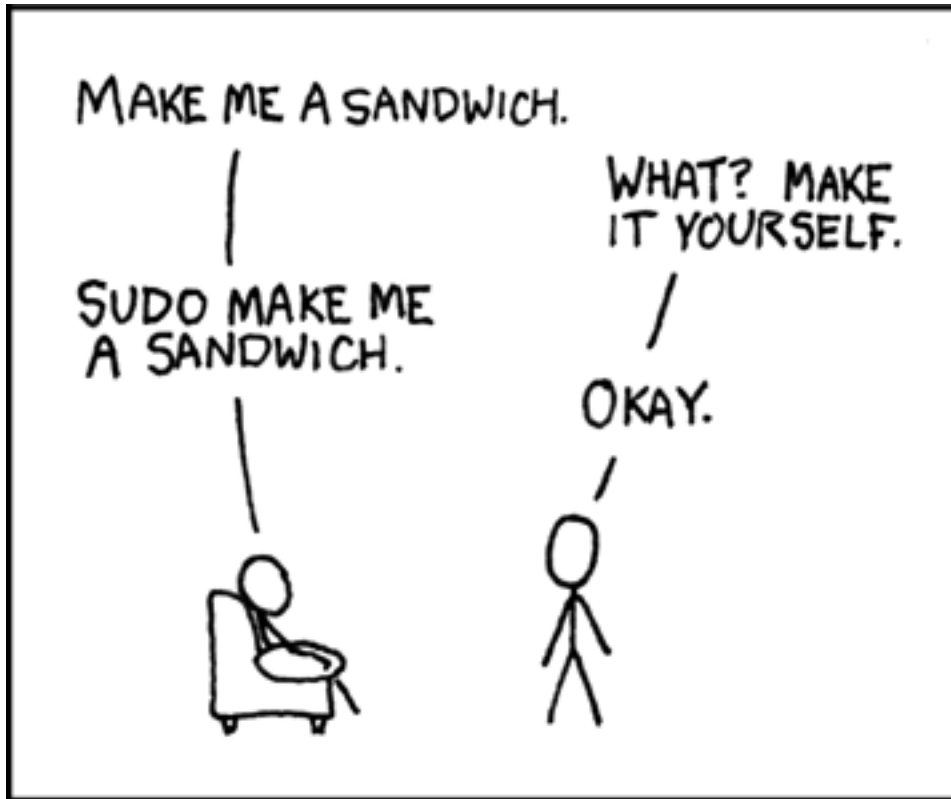
```
$ sudo su -
$ cat /etc/shadow
daemon*:15630:0:99999:7:::
bin*:15630:0:99999:7:::
sys*:15630:0:99999:7:::
mail*:15630:0:99999:7:::
```

```
# name:hash:time last changed: min days between changes: max days
#   between changes:days to wait before expiry or disabling:day of
#   account expiry
```

```
$ chage # change when a user's password expires
```

Root/Superuser

- UID 0
- sudo



Warning: Acting as root is dangerous! You can accidentally delete your filesystem, forcing you to completely re-install your OS! **Type carefully.**

Sudo

Consult `man 5 sudoers` for more information:

```
# User alias specification
User_Alias DOBC_ADMIN = lance, teacher
User_Alias DOBC_STUDENT = john, jane

# Runas alias specification
Runas_Alias ADMIN = root, sysadmin
Runas_Alias STUDENT = httpd

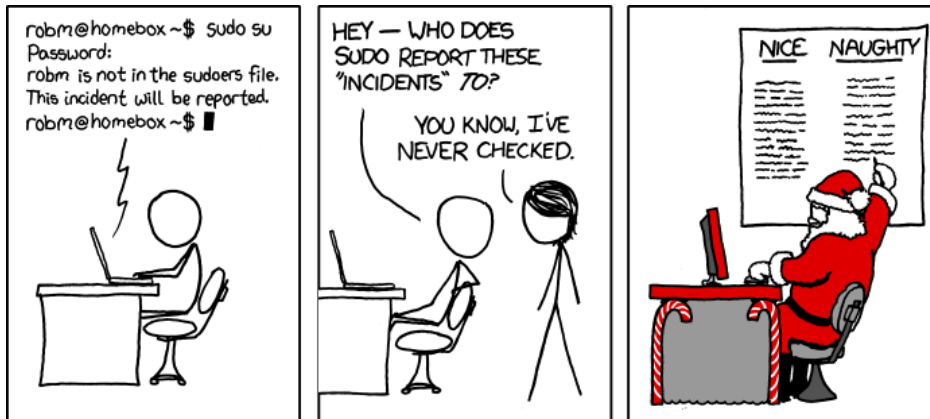
# Host alias specification
Host_Alias OSU_NET = 128.193.0.0/16
Host_Alias SERVERS = www, db

# Cmnd alias specification
Cmnd_Alias KILL = /bin/kill
Cmnd_Alias SU = /bin/su

# User privilege specification
root    ALL = (ALL) ALL
DOBC_ADMIN    ALL = NOPASSWD: ALL
DOBC_STUDENT OSU_NET = (STUDENT) KILL, SU
```

Acting as another user

```
$ su joe           # become user joe, with THEIR password
$ su              # become root, with root's password
$ sudo su -       # become root, with your password
$ sudo su joe     # become user joe with your password
```



A dash after `su` provides an environment similar to what the user would expect. Typically a good practice to always use `su -`

Super users

Trying to run commands which require root permissions as a regular user can be a problem. However, `sudo` authorizes you to do commands based on your permissions. For example:

```
[dobc@dobc ~]$ yum install httpd # Runs command as 'dobc' user.
Loaded plugins: fastestmirror, ovl
ovl: Error while doing RPMdb copy-up:
[Errno 13] Permission denied: '/var/lib/rpm/__db.002'
You need to be root to perform this command.
```

```
[dobc@dobc ~]$ sudo yum install httpd # Runs command as 'root' user.
password:
Loaded plugins: fastestmirror, ovl
[... installs correctly ...]
```

Exercises

1. Create a user on your system for yourself, with your preferred username.
2. Give your user `sudo` powers.
3. Change your password.
4. Use `su` to get into your user account.
5. Create a directory called `bootcamp` in your home directory.
6. Create a group called `devops`.

Exercise Answer Key

```
$ sudo su -
$ useradd lance
# better to use visudo instead
$ echo "lance ALL = (ALL) ALL" >> /etc/sudoers
$ passwd lance
Changing password for user lance.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
$ su - lance
$ mkdir bootcamp
$ sudo groupadd devops
```

We trust you have received the usual lecture from the local System Administrator. It usually boils down to these three things:

- #1) Respect the privacy of others.
- #2) Think before you type.
- #3) With great power comes great responsibility.

[sudo] password for lance:

Further Reading

- [Understanding Linux File Permissions](#)

Next: *Lesson 6: Files*

Lesson 6: Files

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

What are files?

Everything in Linux is a file... except the things that aren't.

Files have:

Owner	atime, ctime, mtime
Group	POSIX ACLs
Permissions	Spinlock
Inode	i_ino
Size	read, write and link count
Filename	

```
$ ls -il
total 8
2884381 drwxrwxr-x 5 test test 4096 Nov  6 11:46 Documents
```

```
2629156 -rw-rw-r-- 1 test test    0 Nov 13 14:09 file.txt
2884382 drwxrwxr-x 2 test test 4096 Nov  6 13:22 Pictures
```

Everything is a file?

Yes. Except the things that aren't...

This functionality isn't just limited to the shell! Let's say you're programming an interface for a device that **streams** data from a sensor. Using the “*Everything is a file*” philosophy, we could read data from the device like so:

```
int read_device_data(int device_file_pointer) {
    // Open a connection to the device
    int * stream = open(device_file_pointer);
    // Write the stream of data to the screen
    write(STDOUT, stream);
    // Do some other stuff with that data
    // Close the data stream
    close(stream);

    return EXIT_SUCCESS;
}
```

More file metadata

```
$ ls -l
crw-rw-rw- 1 root  tty    5, 0 Jan  6 13:45 /dev/tty
brw-rw---- 1 root  disk   8, 0 Dec 21 14:12 /dev/sda
srw-rw-rw- 1 root  root   0   Dec 21 14:13 /var/run/acpid.socket
prw----- 1 lance lance  0   Jan  5 17:44 /var/run/screen/S-lance/12138.ramereth
lrwxrwxrwx 1 root  root   4   Nov 25 09:26 /var/run -> /run

$ stat /etc/services
  File: '/etc/services'
  Size: 19303      Blocks: 40          IO Block: 4096   regular file
Device: fc00h/64512d Inode: 525111     Links: 1
Access: (0644/-rw-r--r--)  Uid: (   0/   root)   Gid: (   0/   root)
Access: 2015-01-07 08:22:43.768316048 -0800
Modify: 2012-05-03 09:01:30.934310452 -0700
Change: 2012-05-03 09:01:30.982310456 -0700
 Birth: -
```

File Extensions

.jpg, .txt, .py

Not necessary, more of a recommendation.

File contains information about its encoding

```
$ ls
some_text_file  squirrel

$ file some_text_file
some_text_file: ASCII text
```



```
$ file squirrel
squirrel: JPEG image data, JFIF standard 1.01
```

Hidden Files

Any file starting with `.` is called a **hidden file** and is not listed by default.

Adding the `-a` flag to `ls` command includes hidden files in your output.

```
$ ls
Documents  file.txt  Pictures

$ ls -a
.  ..  Documents  file.txt  .hidden_file  Pictures  .vimrc
```

Finding Metadata with ls -l

```
$ ls -l
drwxrwxr-x   5 test    test      4096 Nov  6 11:46 Documents
-rw-rw-r--   1 test    test        0 Nov 13 14:09 file.txt
drwxrwxr-x   2 test    test      4096 Nov  6 13:22 Pictures
```

							File Name
						+---	Modification Time
					+-----		Size (in bytes)
					+-----		Group
					+-----		Owner
					+-----		References Count
					+-----		File Permissions & Type

Editing Metadata

You can edit the metadata of a file with various commands, but some of the most useful commands are `chown`, `chmod`, and `chgrp` commands. These commands allow you to edit the owner, the read/write/execute, and the group permissions of a file respectively.

```
# Change the owner of myfile to "root".
$ chown root myfile

# Change the owner of myfile to "root" and group to "staff".
$ chown root:staff myfile

# Change the owner of /mydir and subfiles to "root".
$ chown -hR root /mydir

# Make the group devops own the bootcamp dir
$ chgrp -R devops /home/$yourusername/bootcamp
```

chmod and Octal Permissions

rwX	Binary	Octal
---	000	0
--x	001	1
-w-	010	2
-wx	011	3
r--	100	4
r-x	101	5
rw-	110	6
rwx	111	7

- u, g, o for user, group, other
- -, +, = for remove, add, set
- r, w, x for read, write, execute

Example:

```
$ chmod ug+x my_script.sh
# Adds the permission to execute the file to its
# owner user and owner group.

$ chmod o-w myfile.txt
# Removes the permission to write to the file
# from users other than its owners.
```

Executing a File?

For instance:

```
$ ls -lh my-script
-r-xr-xr-x 1 username username 1.9K Sep 27 09:44 my-script

$ cat my-script
#!/bin/bash
# The above line tells Linux how to invoke the script on my behalf.
echo 'This is a script being run without using bash!'

$ ./my-script # my-script is invoked just like a compiled binary!
This is a script being run without using bash!
```

Types of Files

- - is a normal file
- d is a directory
- b is a block device
- l is a symlink

Directories

Directories are also files!

- `+r` allows you to read the contents of the directory.
- `+w` allows you to add files to the directory.
- `+x` allows you to use the directory at all.

```
$ ls -alh | grep foobarbaz
drw-rw-rw- 2 voigte voigte 4.0K Sep 29 10:47 foobarbaz

# Below is the literal output, not pseudo-output
$ ls -alh foobarbaz
ls: cannot access foobarbaz/.: Permission denied
ls: cannot access foobarbaz/..: Permission denied
total 0
d???????? ? ? ? ?      ? .
d???????? ? ? ? ?      ? ..
```

Exercise: Messing with Files

```
# create empty file called foo
$ touch foo
```

- Create an empty file in `/home/dobc/bootcamp`.
- Who can do what to the file?
- Change the group to `devops`.
- Make a file called `allperms` and give user, group, and world `+rwx`.
- Make more files and practice changing their permissions.

Exercise Answer Key

```
$ touch bootcamp/emptyfile
$ ls -alh bootcamp/emptyfile
-rw-rw-r-- 1 dobc dobc 0 Nov  3 22:38 bootcamp/emptyfile
# You may need to create the devops group.
$ sudo chown dobc:devops bootcamp/emptyfile
# Alternatively, you can also do the following
$ sudo chgrp devops bootcamp/emptyfile
$ touch allperms
$ chmod ugo+rwx allperms
$ ls -l allperms
-rwxrwxrwx 1 dobc dobc 0 Nov  3 22:39 allperms
```

Bonus: What's another way of giving a file all permissions?

Further Reading

- [Permission Mishaps](#)
- [Access the Linux kernel using the /proc filesystem](#)

Next: *Lesson 7: Packages, Software, Libraries*

Lesson 7: Packages, Software, Libraries

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

Software

Everything that isn't hardware.

- Code that is run on a Computer.
- Binaries.
- Scripts.
- Packages

Libraries

- Often used to make development easier.
- Rarely run on it's own.
- Shared code.
- Binaries are linked dynamically to libraries (kind of like DLL's in Windows)

```
$ ldd /usr/bin/nano
linux-vdso.so.1 => (0x00007ffclfdcd000)
libncursesw.so.5 => /lib64/libncursesw.so.5 (0x00007ff2cf8cfaee000)
libtinfo.so.5 => /lib64/libtinfo.so.5 (0x00007ff2cf8c4000)
libc.so.6 => /lib64/libc.so.6 (0x00007ff2cf500000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007ff2cf2fc000)
/lib64/ld-linux-x86-64.so.2 (0x000055e46c4b4000)
```

Package Management

- Automagically manage software and libraries on your system.
- Examples:
 - Android Play Store
 - Apple App store
 - Steam
 - apt (Debian/Ubuntu)
 - yum (CentOS/Fedora/RHEL)

Package Management

Take care of installation and removal of software

Popular Linux System Package Managers

.rpm

- yum - RPM Package manager with repo support
- rpm - low level package manager tool used by yum
- Used by RedHat, CentOS, Fedora and others

.deb

- apt - Debian package manager with repo support
- dpkg - low level package manager tool used by apt
- Used by Debian, Ubuntu, Linux Mint and others

Yum vs. Apt

Yum

- XML repository format
- Automatic metadata syncing
- Supports a plugin module system to make it extensible
- Checks all dependencies before downloading

Apt

- Upgrade and Dist-Upgrade
 - Dist-Upgrade applies intelligent upgrading decisions during a major system upgrade
- Can completely remove all files including config files

Programming Language Package Managers

Examples:

- Python: pip
- Ruby: gem, rubygems
- Haskell: cabal
- NodeJS: npm
- ... and so on forever ...

Other Package Managers

Portage The Source-based package manager for Gentoo.

Pacman The Simple Arch Linux Package manager.

Nix A ‘Fully Functional/Transactional’ package manager.

Brew An *Open Source* package manager for OSX.

Chocolatey A package manager for Windows.

Installation from Source

How to install a package from source:

Exercise: Install `sl`

1. Install the `git`, `gcc`, `make` and `ncurses-devel` packages via package manager.
2. Clone <https://github.com/mtoyoda/sl.git> using `git`
3. Build the software using `make`
4. Copy the compiled `sl` binary into the directory `~/local/bin/`.
5. Update your `$PATH` to include `$HOME/local/bin`
6. Run `'whereis sl'` to ensure it's in your path
7. Run `sl` and see what happens!

Answer: Install `sl`

```
$ sudo yum install git gcc make ncurses-devel
$ git clone https://github.com/mtoyoda/sl.git
$ cd sl
$ make
gcc -O -o sl sl.c -lncurses
$ mkdir -p ~/local/bin
$ cp sl ~/local/bin/
$ echo "export PATH=$HOME/local/bin:$PATH" >> ~/.bashrc
$ source ~/.bashrc
$ whereis sl
sl: /home/dobc/local/bin/sl
$ sl
```

Exercise: Install `grep`

1. Check the current version of `grep`
2. Double check it's location using `which`
3. Download the latest tarball: <http://mirrors.kernel.org/gnu/grep/grep-3.1.tar.xz>
4. Unpack using `tar`
5. `cd` into the unpacked folder

6. Run `./configure --prefix=$HOME/local/`, `make` and then `make install`
7. Run `hash -r` to ensure your environment knows about the new binary
8. Check the current version of `grep` (it should be 3.1 now!)
9. Double check it's location using `which`

Answer: Install grep

```
$ grep --version
grep (GNU grep) 2.20
$ which grep
alias grep='grep --color=auto'
/usr/bin/grep
$ wget http://mirrors.kernel.org/gnu/grep/grep-3.1.tar.xz
$ tar -Jxvf grep-3.1.tar.xz
$ cd grep-3.1
$ ./configure --prefix=$HOME/local/
$ make
$ make install
$ hash -r
$ grep --version
grep (GNU grep) 3.1
$ which grep
alias grep='grep --color=auto'
~/local/bin/grep
```

Further Reading

Lesson 8: Version Control

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

Version Control Systems

VCS is how one tracks changes, modifications, and updates to source files over time. Creating a **history of changes** for a project over time.

Used for:

- Documentation
- Code
- Configuration
- Collaboration

Other Names Include:

- Source Control Management (SCM)

- Version Control Software
- Revision Control Software

What VCS Solves

Version control solves a lot of problems:

- *I have changes I want to integrate (merge) into the main project.*
- *I want to track the state of this project over time.*
- *I want to make some changes without possibly breaking what I have.*
- ... and much more.

Principles of VCS

Types of VCS

Git

Git is a Free and Open Source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. (<https://git-scm.com>)



Setting up Git

```
$ git config --global user.name "My Name"
$ git config --global user.email "myself@gmail.com"
$ git config --global core.editor "nano"
```

TODO: Use Git Locally

Create a project with Git:


```
$ mkdir my-project
$ cd my-project      # Always run 'git init' inside of a project folder!
$ git init           # Never inside of your home directory.
```

Add and commit a file to your project with Git:

```
$ touch newfile.txt
$ git add newfile.txt
$ git commit # Edit message in Nano, save the file, exit to commit.
```

To see which files are staged, unstaged, or untracked:

```
$ git status
```

To look through your repository history:

```
$ git log
```

To create and checkout a branch:

```
#Note the '*' which indicates the current branch
$ git checkout -b "new-branch"
$ git branch
master
* new-branch
```

TODO: Working With a Git Repository

Checkout a new feature branch on your repository.

```
$ git checkout -b "add-awesome-feature"
```

Create/Edit files on the new branch.

```
$ echo "Some awesome text" > awesomefile.txt
$ git status
# On branch add-awesome-feature
# Untracked files:
...
#      awesomefile.txt
...

$ git add awesomefile.txt
$ git commit -m "Short awesome commit message"
```

View the diff between the two.

```
$ git diff master
diff --git a/awesomefile.txt b/awesomefile.txt
new file mode 100644
index 0000000..08cec7f
--- /dev/null
+++ b/awesomefile.txt
@@ -0,0 +1 @@
+Some awesome text
```

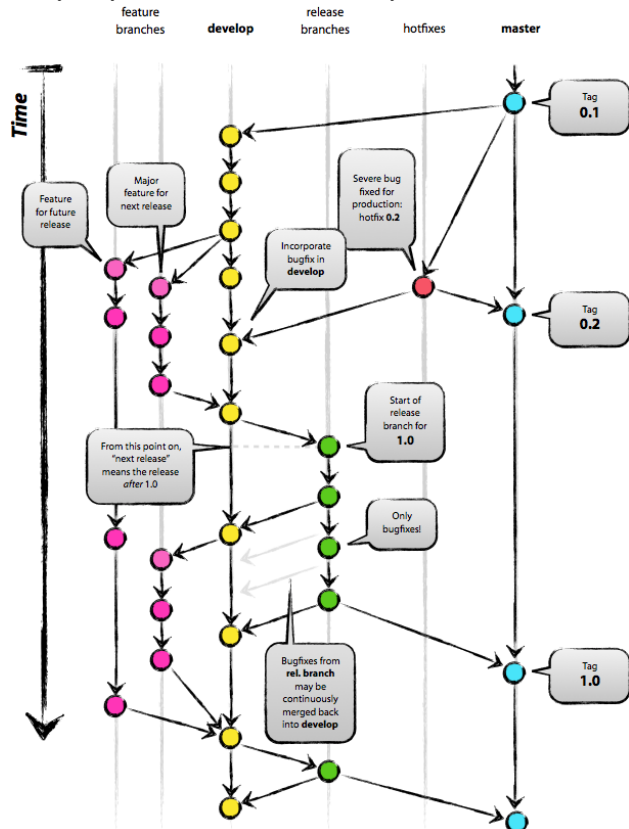
Locally merge the changes from your new branch into Master.

```
$ git checkout master
$ git merge add-awesome-feature
Updating 459de26..5c4ca48
Fast-forward
 awesomefile.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 awesomefile.txt
```

What not to do with Git

Workflow(s)

Everybody uses VCS differently. Choose the workflow that works best for everybody involved.



Centralizing Git

Gitlab Open Source, free to run, feature rich.

Github Very popular. Not Open Source but free for Open Source projects.

Bitbucket Also popular, similar to Github, unlimited free private and public repositories.

Gitolite *Bare-bones.* Fewer features than the previous three. Open Source, useful for learning the nitty-gritty on how Git *really* works.

Cloning a Repository

To contribute to someone else's repository you first need to *clone* the repo.

```
$ cd /path/to/my/projects
$ git clone <some git url>
$ cd <new repo directory>
$ ls
```

Once you clone a repository you can make as many local changes as you want without affecting the original (central) copy. You can experiment and work without the original owner even knowing what you're doing!

TODO: Cloning Exercise

```
$ cd ~
$ git clone https://github.com/DevOpsBootcamp/tinsy-flask-app.git
$ cd tinsy-flask-app
```

See <http://git.io/vcVmB> for more details about the `tinsy-flask-app` repository.

```
#Setup python virtual environment
$ virtualenv venv
$ source venv/bin/activate
(venv) $ pip install -r requirements.txt
#Run server
(venv) $ python script.py
#When finished, deactivate virtual environment
(venv) $ deactivate
$
```

Further Reading

The Online Git Docs This is a portal to all of the official docs on git-scm.com. It includes everything from *Getting Started* to *Git Internals*. Check it out!

Git workflow tutorial This is the tutorial provided on <https://git-scm.com/about/distributed>. It is a good high-level overview of some common git workflows.

A successful Git branching model This blogpost describes a git workflow (git-flow) that the Open Source Lab bases their workflow on.

Lesson 9: Programming

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

Paradigms

Programming is a big topic.

Note: Pseudo-code

```
function f(x):
    # This line is a comment, not run by the computer.
    # Comments are only for human eyes.
    if x is less than 5
        print "x is less than 5"
    else if x is less than 10
        print "x is greater than five and less than 10"
    else
        print "x is greater than 10"
```

Variables & Constants

```
>>> x = "value"
>>> print(x)
value
>>> x = "different value"
>>> print(x)
different value
```

Data Types

Data types dictate how a piece of data should be handled within a program.

Flow Control

Flow Control allows you to execute code only if certain conditions are met.

Conditionals: If / Else If / Else Conditionals are used to tell the program when to execute commands.

In pseudocode, they usually look something like

```
if some conditional statement is true
    do something
else if some other conditional
    do something else
else
    do a final thing
```

Loops: For / While / Do While Loops are used to do multiple things, usually an *indefinite* number of things.

For instance:

```
for every element, let's call it "foo", in a list "my_list"
    if foo is greater than five
        print(foo)
    else
        print(foo + " is too small")
```

While loops execute indefinitely (while something continues to be true).

For loops iterate over a list (array) of elements or to a specific number.

Input & Output

```
>>> user_input = get_input("Where would you like to go today? ")
>>> -> Where would you like to go today? Nebraska
>>> print(user_input)
>>> -> nebraska
>>> print(reverse(user_input))
>>> -> aksarben
```

Functions

```
function read_file(x):
    # Also check that it exists! How convenient!
    if file_exists(x)
        v = read_file_to_string(x)
        return v
    else
        print("file does not exist")
        return Null
```

Structs

```
struct dog {
    breed: String
    height: Float
    color: String
    age: Integer
}

spot = struct dog      # Create a new variable of type 'struct dog'
spot.breed = "corgie"  # Assign each member a variable.
spot.height = 1.5
spot.color = "Blond"
spot.age = 1
print(spot.breed, spot.height, spot.color, spot.age)
```

Objects

```
class chair():
    function init(material):
        self.material = material

    function rock():
        print("The ", self.material, " chair rocks slowly.")

>>> my_chair = chair.init("plastic")
>>> my_chair.rock()
>>> -> The plastic chair rocks slowly.
```

Libraries

```
import math_lib

print(math_lib.pi, math_lib.pow(2, 5), math_lib.tan(79.3))
# prints out "3.14 32 .951"
```

TODO: Write Pseudo-Code

Write pseudo-code to do the following tasks:

- Count to 20 (hint: `for` loop).
- Get user input and print it.
- Generate prime numbers.

Hints:

- Break the problem down to the simplest steps.
- Don't worry about the details.
- This is pseudo-code! Get creative.

Python

```
$ sudo <apt or yum> install python
```



Python Datatypes

- You don't need to declare the type of your variables, Python will assume the type of your variable and type it for you.

- Python is a duckly-typed language. If it walks like a duck and quacks like a duck, then Python treats it like a duck. As long as an object implements the proper *interfaces*, it can act like any type it wants.

Type	Example
boolean	True
integer	7
long	18,446,744,073,709,551,615
float	12.4
string	"Hello World!"
list	['first', 'second']
dict (map)	{'key1': 'value', 'key2', 'value2'}
tuple	('value', 'paired value')
object	anObjects.variable == <value>
None	

Python Variables

```
# This is a comment
boolean = True # boolean
name = "Lucy" # string
age = 20 # integer
pi = 3.14159 # float
alphabet = ['a', 'b', 'c']
dictionary = {"pi":3.14159, "sqrt 1":1}
winter = ('December', 'January', 'February', 'March')

print(name + " is " + str(age+1) + " this " winter[3])
```

REPL: Try it out

Open a REPL (Read Evaluate Print Loop):

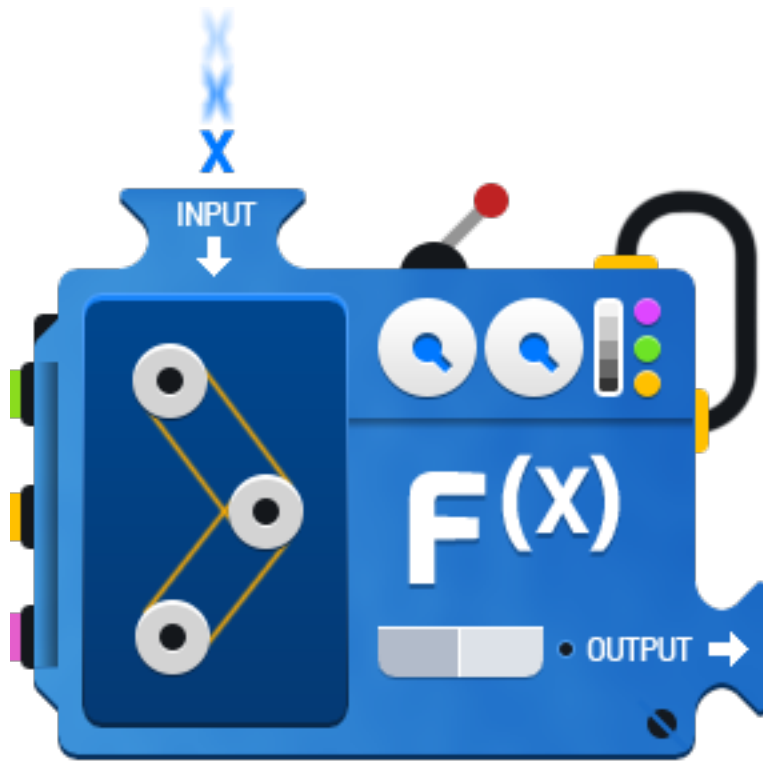
```
$ python
>>> print("I'm in a REPL!")
>>> name =      # <Your name>
>>> age =       # <Your age>
>>> print(name + " is " + str(age))
>>> # We need to convert age from int to string so it can print!
```

Python Control Flow

```
if name == "Lucy":
    for month in winter:
        print name + " doesn't like " + month
else:
    print "My name isn't Lucy!"
```

Python Functions

```
def myfunction(arg1, arg2):  
    return arg1 + arg2  
  
print myfunction(1, 5)
```



WWW.MATHWAREHOUSE.COM

Python Libraries

There are a few ways to use other code in your code:

```
from math import pi  
x = pi  
  
from math import *  
x = pi
```

There are **hundreds** of Python libraries. If you're trying to do something and think "This has probably been solved...", Google it!

Some libraries to know:

- sys
- os
- dateutil
- future

- And more

Python (Virtual) Environments

```
$ sudo apt-get install python-virtualenv
$ sudo yum install

# In each project you work on, you'll want to run
$ virtualenv venv
$ source venv/bin/activate
(venv)$ pip install <package>
(venv)$ deactivate
```



TODO: Practicing Python

Formalize the last TODO by writing them in Python.

Prove the program works by running the code!

Further Reading

Python on Learnpython.org The Python programming language's website offers some good (free) tutorials and reference documentation.

Python on Codecademy Codecademy is a great resource for learning many programming languages and offers a good (free) beginner's guide to Python.

CS 160, 161, 162 These OSU courses focus on programming fundamentals covered in this lesson in greater detail. Python is used in CS 160 and C/C++ is used in CS 161 and CS 162.

Lesson 10: Frameworks

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

Frameworks

Frameworks are collections of classes, functions, and constants designed to make completing a task easier.

Types of frameworks include:

- Web frameworks
- Game frameworks
- GUI frameworks

The job of a framework

To take care of the boring stuff.

Why and When to use a Framework

Use a framework if you are making a *cookie cutter* application.

If a framework exists for what you're doing, consider using it.

Looking for Frameworks

Things to keep in mind when looking for a framework:

- Good frameworks usually have:
 - Good documentation
 - Active developers
 - A helpful community

Web Frameworks



Static vs Dynamic Sites

There are two types of websites: Static and Dynamic.

Static Site Rarely changes, looks the same for all visitors (Blog, News, Document)

Dynamic Site Changes based on who you are and what you do. (Search Engine, Login)

Popular Web Frameworks

Python

Django Offers many feature out of the box: Admin page, easy database management, simple templating, convenient URL routing. Well documented too.

Flask Sparsely featured, offers very little out of the box and lets you build *up* the features you need. Well supported with community libraries and add-ons.

Ruby

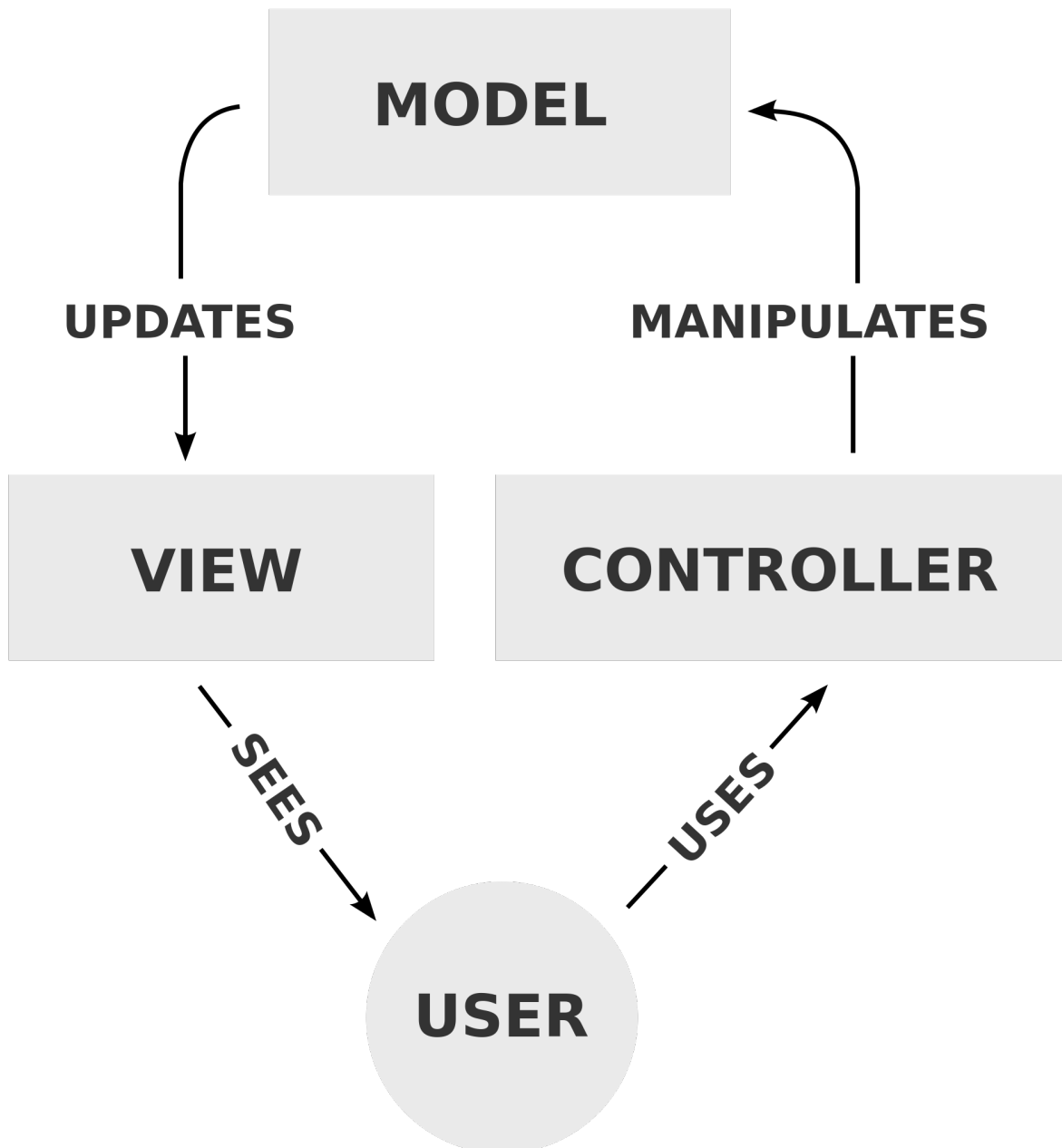
Rails Arguably the most popular web-framework out there. Similar to Django in it's features out of the box.

Sinatra Analogous to Flask on the Python side, very simple and easy to start with, encourages building *up* the features you need.

Node.js

ExpressJS A bare-bones NodeJS application, similar again to Flask.

The Model-View-Controller Pattern



URL Routing

```
@app.route('/accounts/<account_name>', methods=['DELETE'])
def delete_account(account_name):
    if authenticated() and authorized():
        database.remove_account(account_name)
        return 'Success', 200
```

```
else
    return 'Failure', 401
```

Templating Engines (mad-libs!)

```
<!--DOCTYPE HTML-->
<html>

  <head>
    <title>Template Example</title>
  </head>

  <body>
    <p>Your lucky number today is {{ number }}!</p>
  </body>

</html>

render_template("template.html", number=random.randint(0, 99))

Your lucky number today is 42!

...
<body>
{% for message in messages %}
  <p>{{ message }}</p>
{% endfor %}
</body>
...

messages = ["Welcome!", "Test Message", "Vim > Emacs"]

render_template("template2.html", messages=messages)

Welcome!
Test Message
Vim > Emacs
```

HTTP

```
GET http://web.site/page.html HTTP/1.1

HTTP/1.1 200 OK
Content-Type: text/html
...
<!--DOCTYPE HTML-->
...
```

HTTP Methods

REST

- Servers are stateless
- Resources are self-contained

- HTTP methods have predictable side-effects

TODO: Dynamic Website

Part One: Writing The Views

Part Two: Writing The Templates

Further Reading

The Flask Microframework Flask is a web framework that is simple enough for beginners to use but configurable enough to allow more advanced users to have full control over their application. It has a very active community and fantastic documentation.

Intro to HTTP and REST HTTP is the protocol that web clients and web servers use to communicate with each other, and REST is a set of web design guidelines that takes advantage of HTTP's features and allows different applications to easily communicate with each other.

Lesson 11: Testing

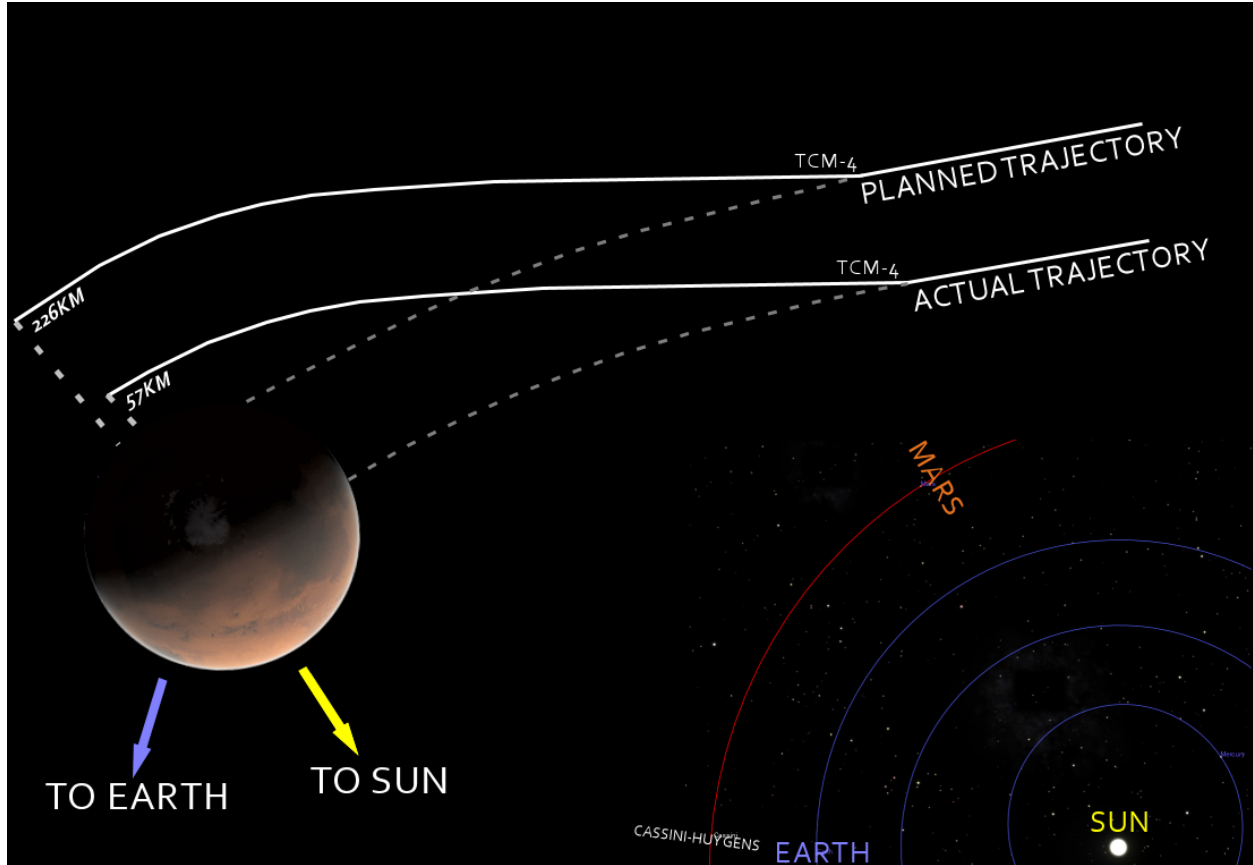
Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

Testing

```
def add_double(x, y):  
    return 2 * (x+y)  
  
def test_add_double():  
    expect(add_double(1, 2) == 6)
```

Why Testing Matters



Structure of a Test

Most tests consist of the same general structure:

Types of Testing

Concept: Mocking

Simulating behavior external to a program so your tests can run independently of other platforms.

You're testing **your** program, not somebody else's. Mock other people's stuff, not your own.

Testing Frameworks

```
$ run tests
Finding tests...
Running tests in tests/foo.ext
Running tests in tests/bar.ext
Running tests in misc/test_baz.ext
```

Frameworks vs ‘The Hard Way’

While you *can* write tests the hard way:

```
var = some_function(x)
if var == expected_output:
    continue
else
    print("Test X failed!")

$ run test
Test 5 failed!
```

It’s usually easier to use a framework.

```
def simple_test():
    expect(some_function(x), expected_output)

$ run tests
....X....
Test 5 failed.
Debug information:
...
```

Teardown and Setup

Useful for:

- populating a test database
- writing and deleting files
- or anything else you want!

```
def tests_setup():
    connect to database
    populate database with test data

def tests_teardown():
    delete all data from test database
    disconnect from database

def some_test()
    setup is called automatically
    use data in database
    assert something is true
    teardown is run automatically
```

TODO: Using Python’s unittest

Let’s suppose that we want to add a new view to the Flask app we created in the Frameworks lesson’s TODO. When the user enters the url /hello/<name>, where “name” is any string of the user’s choice, the view should return “Hello <name>!!” BEFORE you actually write this view, write a test that will test the desired functionality first– i.e., test that your hello.py returns “Hello bob!!” when “bob” is provided as the name variable. AFTERWARDS, implement the actual view to make your test(s) pass.

Unittesting in Flask Check out the official Flask docs for help with syntax.

Further Reading

CS 362 This OSU Course covers testing *very* in depth and even covers types of testing including *Random* testing and testing analysis.

Python Unittest Documentation A good reference for using Python's built-in unit-testing module.

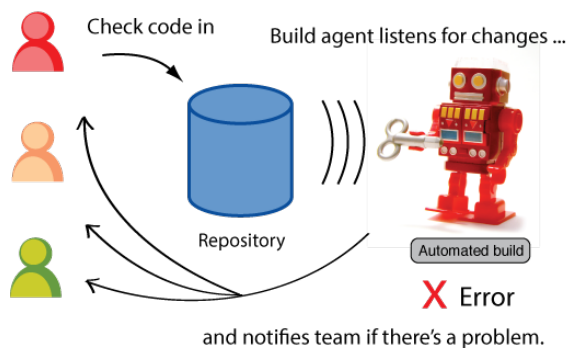
Lesson 12: Continuous Integration

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

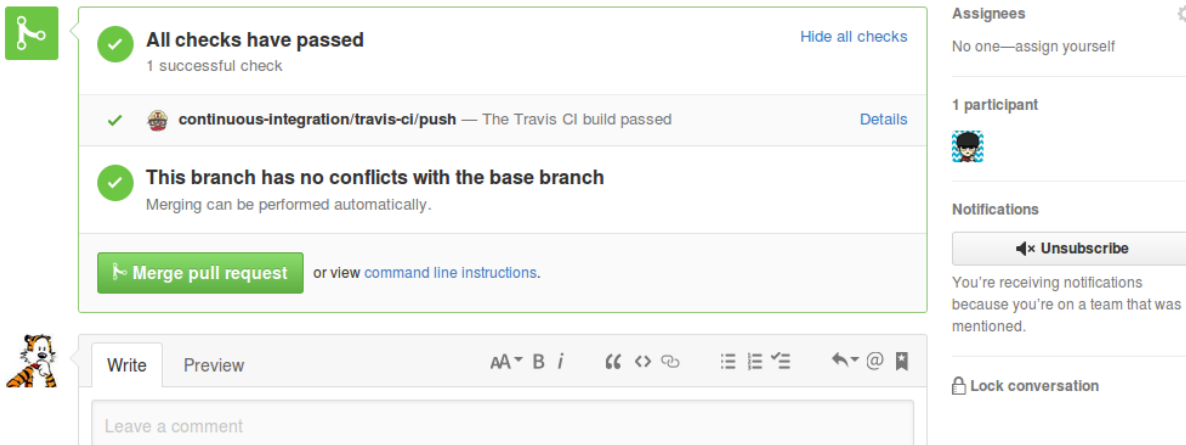
Continuous Integration

Continuous Integration is a name for any kind of automated tool that performs the following tasks:
Developers



1. Detects changes to your project
2. Runs a suite of tests on the changed code
3. Alerts the people that care if something good/bad happened

Automated Testing



The screenshot shows a GitHub pull request interface. On the left, a green box indicates that all checks have passed, with a link to 'Hide all checks'. Below this, a specific check for 'continuous-integration/travis-ci/push' is shown as passed, with a link to 'Details'. Another green box states 'This branch has no conflicts with the base branch', noting that merging can be performed automatically. A green button labeled 'Merge pull request' is visible, along with a link to 'view command line instructions'. On the right side, the 'Assignees' section shows 'No one—assign yourself'. The '1 participant' section shows a user profile. The 'Notifications' section includes an 'Unsubscribe' button and a note that the user is receiving notifications because they are on a team that was mentioned. At the bottom right, there is a 'Lock conversation' option.

Tool: Travis CI

Travis CI is very popular among Github users because it is easy to setup with Github projects and integrates well with Github workflows. It's also free for Open Source projects, although the service itself is not Open Source.

```

925 validateUUID
926 ✓ returns true for a valid UUID
927 ✓ returns false for an invalid UUID
928 ✓ returns false for a non-string UUID
929 ✓ returns false for null
930
931
932 213 passing (6s)
933
934
935 The command "npm run test_pg" exited with 0.
936 $ npm run linter
937
938 > timesync@0.0.0 linter /home/travis/build/osuosl/timesync-node
939 > jshint ./src ./tests ./scripts && eslint ./src ./tests ./scripts
940
941 The react/jsx-quotes rule is deprecated. Please use the jsx-quotes rule instead.
942
943 The command "npm run linter" exited with 0.
944
945 Done. Your build exited with 0.

```

Runs test suites for:

- C / C++
- Java
- Javascript
- Python
- Ruby
- *Many* more on [the Travis CI docs!](#)

Tool: Jenkins

Jenkins is a more powerful version of Travis, but as a result is more complicated to use and set up. While you can pay to use a public instance of Jenkins, it is more common to run your own instance of Jenkins.

- Does pretty much anything you can tell a computer to do, automatically.
- Builds and uploads binaries (releases).

- Runs tests.
- Reports build successes/failures.
- Also has plugins!

TODO: Setup Travis on a GH Repo

Further Reading

Jenkins Documentation The Jenkins project documentation. If you need a broad overview read the *Getting Started with...* docs.

TravisCI Documentation If you end up working on a large project on GitHub you're going to interface with TravisCI sooner or later.

CircleCI Documentation CircleCI is a tool we didn't get to touch on. It is very similar to Travis.

Lesson 13: Security

[Homepage](#) [Content](#) [Slides](#) [Video](#)

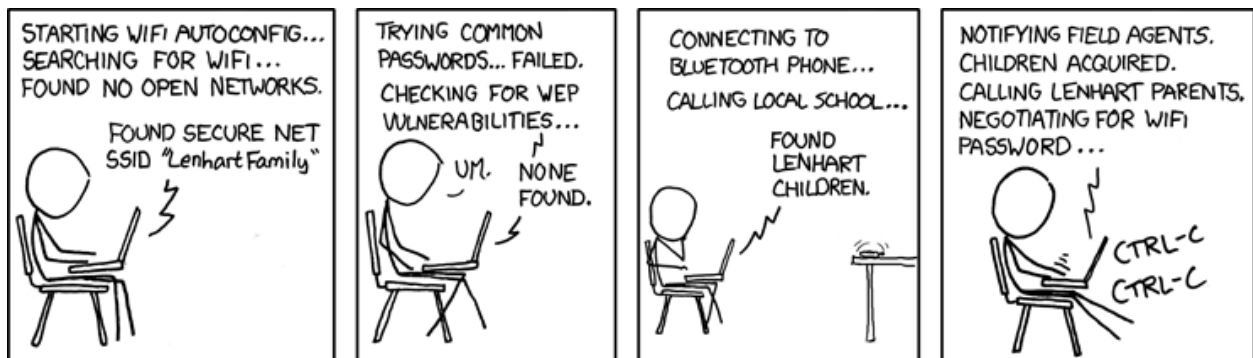
Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

Security

se·cu·ri·ty (*sikyooritē* /) [**noun**] The state of being free from danger or threat.

The safety of a state or organization against criminal activity such as terrorism, theft, or espionage.

Types of Security



There are three main types of security in computing:

Physical Security Use physical barriers to prevent unauthorized access to data

Software Security Fix flaws in your application that could grant attackers unwanted levels of access to your systems

Network Security Security pertaining to networked services (websites, databases, etc).

- **Active:** in which an intruder initiates commands to disrupt the network's normal operation (Denial-of-Service, Ping of Death)
- **Passive:** a network intruder intercepts data traveling through the network. (Man-in-the-Middle, Wiretapping, Idle Scan)

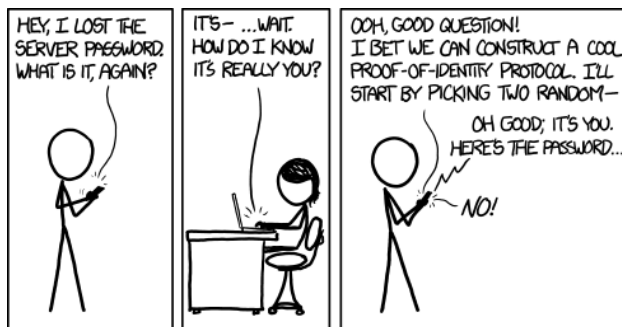
Each of these encompasses a field of computer security unto itself. We will at least mention each of them in more detail, but we will focus on network security in this course.

Threat Models

Threat models allow you to focus and limit your security resources on what is *necessary* instead of what is *possible*.

Threat models are the assessment of which attacker you are protecting against. This is so you don't spend too much time in a panic attack trying to protect your tiny webapp from the NSA.

Access Control



- **Identification:** Who is this person?
- **Authentication:** Is this person who they say they are?
- **Authorization:** Is this person allowed to perform this action?

Access Control is a framework for controlling who has access to what resources on a system. There are many ways to implement Access Control, but the three basic principles of Access Control are *Identification*, *Authentication*, and *Authorization*.

Passwords / Passphrases

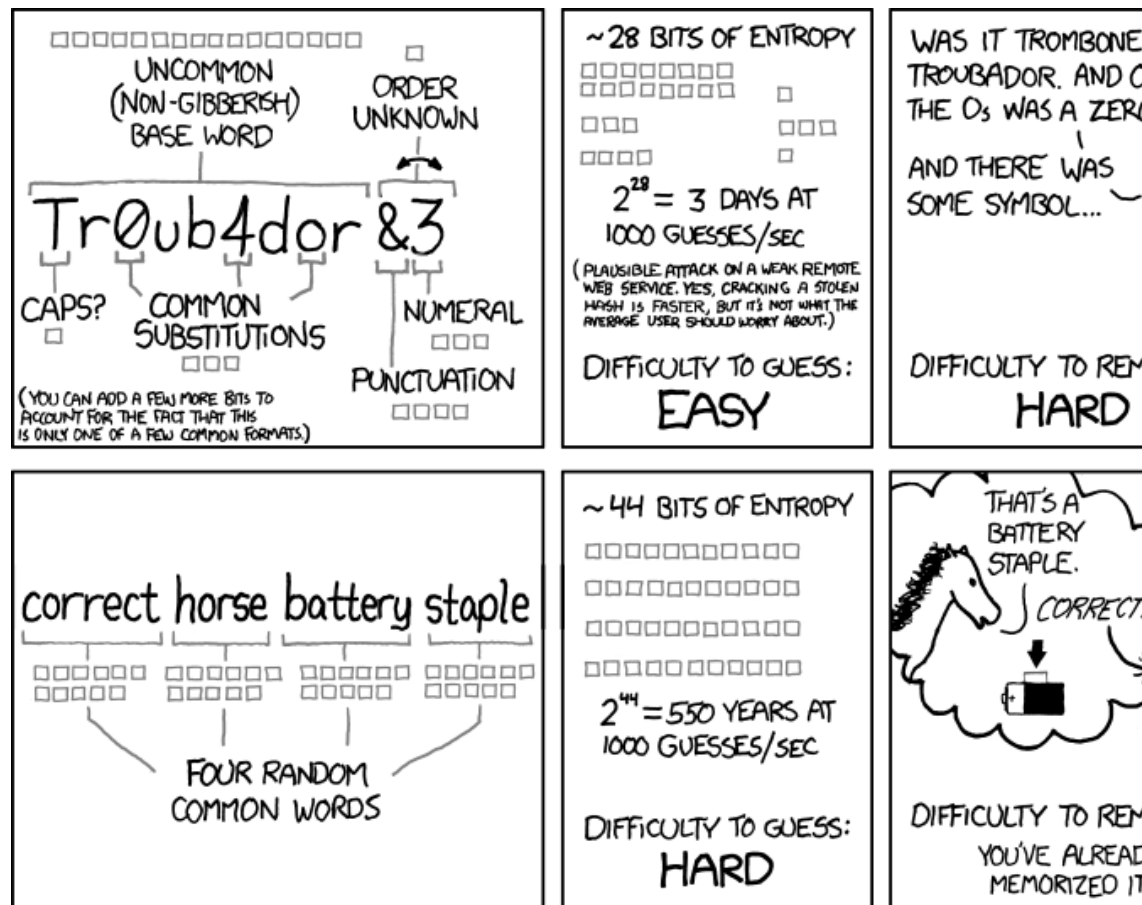
Problems with Passwords

Passwords are a necessary part of security. They aren't great though for a few reasons.

- People repeat passwords.
- Many passwords are easy to guess.
- Passwords are hard to remember.

Solutions for Passwords

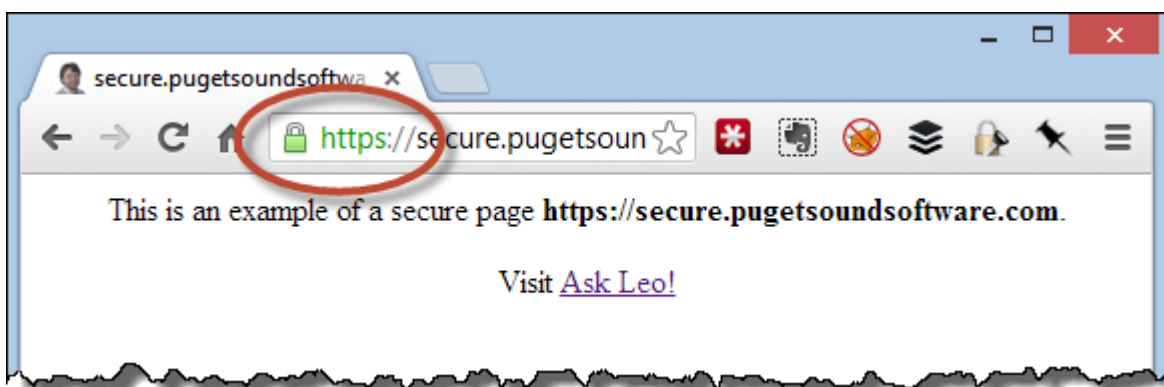
Choosing Pass-phrases



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Relevant funny bash.org post

Certificates and HTTPS



Types of Attacks

According the security vendor Cenzic, the top vulnerabilities in March 2012 include:^[9]

37%	Cross-site scripting
16%	SQL injection
5%	Path disclosure
5%	Denial-of-service attack
4%	Arbitrary code execution
4%	Memory corruption
4%	Cross-site request forgery
3%	Data breach (information disclosure)
3%	Arbitrary file inclusion
2%	Local file inclusion
1%	Remote file inclusion
1%	Buffer overflow
15%	Other, including code injection (PHP/JavaScript), etc.

Code Injection



Code Injection is the act of inserting code into a running process (website, webapp, word processor, etc.) with malicious intention.

Code Injection Attacks

SQL Injection: SQL Injection is when you take advantage of the fact that a form input is inserted directly into a SQL query. You write some password and then write a new SQL query which drops all tables, or returns all data,

exploiting an easy security hole.

```
+-----+-----+
| username: | admin |
+-----+-----+
| password: | pass' || true); DROP TABLE STUDENTS;-- |
+-----+-----+
```

Cross-Site Scripting (XSS): Cross-Site Scripting is when a malicious script is sent to, and run on, a person's computer. This tends to take advantage of the fact that your browser blindly runs any JavaScript you tell it to.

```
<img onerror=alert("Tracking your IP with a GUI interface!");>
```

Cross-Site Request Forgery (CSRF): CSRF is when one website on your browser tries to carry out an action *as you* on a different website. For instance you're an admin of some big social media website, you get an email, embedded in the email is a CSRF script which tries to *delete all user accounts* on your website. Since you've got your credentials cached your browser doesn't know better and can run that command because it looks like any other command.

```

```

Code Injection Defenses

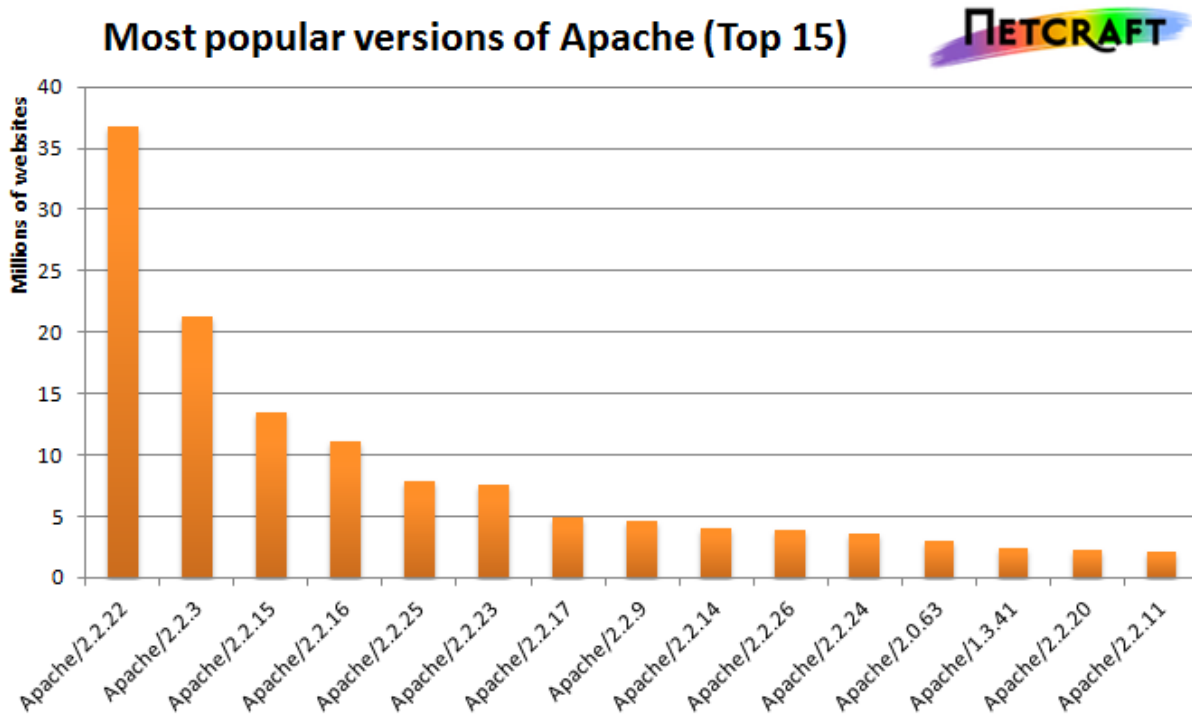
- Sanitize User Inputs
- Use CSRF Tokens

Some of these attacks are very hard to fight against, but they all have industry-tested solutions that are easy enough to implement in an application of your own.

Sanitize Inputs Input sanitation is when your code sniffs a piece of input to see if it looks like a SQL or code of any kind. If it does look like code it's probably malicious so your program errors out and tells the user to enter a *real* input.

CSRF Tokens A CSRF token is a unique string that has to be tied to each request you send to a server. You don't need to log back in each time you get a new one but the application won't complete your action unless the token is included in your query. This means only the website you're logged into can send a real query because only that website knows the CSRF token.

Web Server Attacks



Web Server attacks take advantage in vulnerabilities of specific versions or default configurations of web servers.

Discovering Vulnerabilities

1. **Test and document the bug to verify it exists.** If you think you encountered a bug, make sure you can replicate it. If you can't how can you expect the developers to recreate it?
2. **Disclose it privately to those responsible for fixing it.** Provide examples – it's basically a bug report, but through private channels (not public tracker yet!)
3. **Give them time to release a patch before announcing it.** Google waits 90 days to announce a bug after informing the developers.

Further Reading

codebashing.com/sql_demo Try your hand at *actual* SQL Injection attacks

OverTheWire Wargames Learn the basics of offensive security by solving challenges and using exploits to gain access to the password for the next level.

Lesson 14: Databases

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

Databases

Relating Data

Imagine a kitchen cupboard program that stores food currently in stock, where it is, recipes using it, expiration dates, etc.

Databases and Structure

Structure SQL databases are based on around Relational Algebra

<Table 1>

<Primary key>	<Field 1>	<Field 2>
1	value	value'
...

<Table 2>

<Primary key>	<Field 1>	<Foreign key to Table 1>
1	val	7
...

Concept: Relational Algebra

<Table 1>

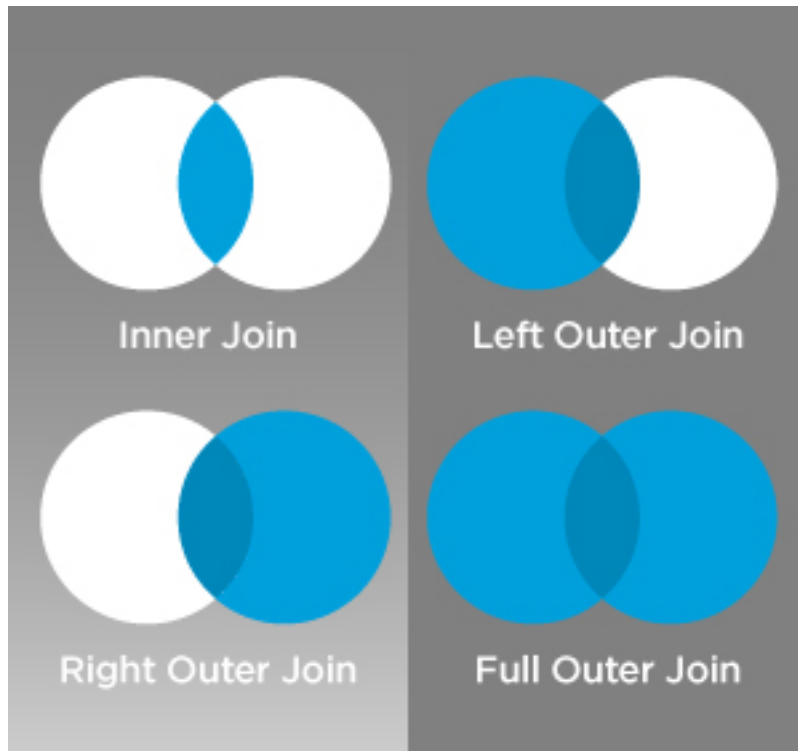
<Name>	<Major>
Linus Torvalds	Computer Science
Richard Stallman	Computer Science

<Table 2>

<Major>	<School>	<Advisor Name>
Computer Science	Engineering	Dennis Ritchie

<Table 1> JOIN <Table 2>

<Name>	<Major>	<School>	<Advisor Name>
Linus Torvalds	Computer Science	Engineering	Dennis Ritchie
Richard Stallman	Computer Science	Engineering	Dennis Ritchie



When to use a Database

When you have to work with a lot of well structured data.

Databases are useful for two situations:

1. Lots of data.
2. High throughput.

Lots of Data

Global Internet traffic by year			
Year	IP Traffic (PB/month)	Fixed Internet traffic (PB/month)	Mobile Internet traffic (PB/month)
1990	0.001	0.001	n/a
1991	0.002	0.002	n/a
1992	0.005	0.004	n/a
1993	0.01	0.01	n/a
1994	0.02	0.02	n/a
1995	0.18	0.17	n/a
1996	1.9	1.8	n/a
1997	5.4	5.0	n/a
1998	12	11	n/a
1999	28	26	n/a
2000	84	75	n/a
2001	197	175	n/a
2002	405	356	n/a
2003	784	681	n/a
2004	1,477	1,267	n/a
2005	2,426	2,055	0.9
2006	3,992	3,339	4
2007	6,430	5,219	15
2008 ^[17]	10,174	8,140	33
2009 ^[18]	14,686	10,942	91
2010 ^[19]	20,151	14,955	237
2011 ^[20]	30,734	23,288	597
2012 ^[21] ^[22]	43,570	31,339	885
2013 ^[23]	51,168	34,952	1,480
2014 ^[24]	59,848	39,909	2,514
2015 ^[25]	72,521	49,494	3,685

Note: 1 PB = 1,000,000 GB

Concurrent Read/Writes

Atomicity: Either the entire transaction succeeds or it fails completely

Consistency: Transactions always leave the database in a valid state

Isolation: Concurrent operations look like they took place sequentially

Durability: Transactions are permanent after they're committed

When *not* to use a Database

Databases might not be particularly useful for:

- Storing content for a website that rarely updates
 - **Alternative:** Use a static site generator such as Pelican or Jekyll
- Hosting large individual files
 - **Alternative:** Store the files on disk

Types of Databases

There are two broad types of databases.

- **SQL:** Stores data in tables organized by column and field.
- **NoSQL:** Stores data differently than an SQL database.
- **NewSQL:** A middle-ground between SQL and NoSQL

SQL

Examples:

- MySQL/MariaDB
- PostgreSQL
- SQLite

NoSQL

Examples:

- MongoDB
- Apache Cassandra
- Dynamo
- Redis

Database Concepts

Schemas

Schemas are how you define what a table looks like, what data will populate it, and what each field will be called. The schema also defines relationships between tables; more or less the blueprint of your database.

```
CREATE TABLE nobel (  
  id int(11)  
    NOT NULL  
    AUTO_INCREMENT,  
  yr int(11),  
  subject varchar(15),  
  winner varchar(50)  
)  
ENGINE = InnoDB;
```

Migrations

Migrations are the process of updating tables and fields in your database. Since databases *might* need to change in the future (you never know!) you can create and run a migrations to modify your schema as needed.

```
from django.db import migrations, models

class Migration(migrations.Migration):
    dependencies = [
        ('app', '0001_initial')
    ]

    operations = [
        migrations.AddField("Nobel", "topic", models.CharField(80))
    ]
```

Raw SQL Syntax

There are many tools out there that allow you to *avoid* writing raw SQL, but it's always good to know the syntax. One day you may need to write raw SQL queries, and at the very least you'll need to *read* SQL for debugging purposes.

SELECT

Select statements **get** data from the database which matches the requirements you have.

```
SELECT
    yr, subject, winner
FROM
    nobel
WHERE
    yr = 1960 AND subject='medicine';
```

```
+-----+-----+-----+
| yr   | subject | winner |
+-----+-----+-----+
| 1960 | "medicine" | "Sir Frank Macfarlane Burnet" |
| 1960 | "medicine" | "Sir Peter Brian Medawar" |
+-----+-----+-----+
```

INSERT

Insert statements create an entry into a table and populate the fields appropriately.

```
INSERT INTO
    nobel
VALUES
    ('2013', 'Literature', 'Herta Müller');
```

```
+-----+-----+-----+-----+
| id  | yr  | subject | winner |
+-----+-----+-----+-----+
| ... | ... | ...     | ...     |
| 873 | 2013 | "Literature" | "Herta Müller" |
| ... | ... | ...     | ...     |
+-----+-----+-----+-----+
```

UPDATE

Update statements modify an existing entry in a table.

UPDATE`nobel`**SET**`winner='Andrew Ryan'`**WHERE**`subject='Peace' AND yr='1951';`

id	yr	subject	winner
...
120	1951	"Peace"	"Andrew Ryan"
...

DELETE

Delete statements... You can guess what a delete statement does I bet.

DELETE FROM`nobel`**WHERE**`yr = 1989 AND subject = 'peace';`**TODO: Crafting Queries!**

Craft a query to get the following data out of our Nobel table:

- Who won the prize for Medicine in 1952?
- Who won the 1903 Nobel in Physics?
- Which prize(s) were awarded to Linus Pauling?
- How many people have won more than once? (Difficult)

Don't worry about getting it exactly right! Craft pseudo-SQL!

Answers

```
SELECT winner FROM nobel
```

```
WHERE yr=1952 AND subject='medicine'; #(Selman A. Wksman)
```

```
SELECT * FROM nobel
```

```
WHERE yr=1903 AND subject='physics'; #(3)
```

```
SELECT * FROM nobel
```

```
WHERE winner='Linus Pauling'; #(2)
```

```
SELECT COUNT(*) FROM nobel
```

```
AS n0 INNER JOIN nobel AS n1 on n0.winner=n1.winner
AND (n0.yr!=n1.yr or n0.subject!=n1.subject); #(16)
```

TODO: Using a *Real* Database

Now that we have belabored the *theory* of databases and SQL, lets actually start *doing* work with databases.

Throughout this exercise we will load it up with some data (`nobel.sql.gz`) and learn to interact with it via the command line interface.

Importing Data

```
# Create a table for Nobel prizes
$ mysqladmin -u root create nobel
# Get the database from the osl server
$ wget http://osl.io/nobel -O nobel.sql.gz
# Gunzip the file and import it into the nobel db
$ gunzip nobel.sql.gz
$ mysql nobel < nobel.sql
# OR do it in one step!
$ zcat nobel.sql.gz | mysql nobel
# Open up mysql shell to execute queries
$ mysql nobel

# List all the tables
SHOW TABLES;
# Print the layout of the database to the screen
DESCRIBE nobel;
```

Ways to Use a Database

Now that you have a working database you have a few options for how you want to use it.

- Raw SQL Queries
- Native Queries
- ORMs

Raw Queries

We've already done this in the previous exercise. You use your choice of program to interact with the database exclusively via SQL and run the queries you want. This is rarely the way to go and isn't very useful for most applications. The SQL language is only good for doing database *stuff*.

```
mysql> SELECT subject, yr, winner FROM nobel
WHERE yr=1960;
```

```
+-----+-----+-----+
| yr    | subject      | winner                    |
+-----+-----+-----+
| 1960  | Chemistry    | Willard F. Libby          |
| 1960  | Literature    | Saint-John Perse          |
| ...   | ...          | ...                       |
+-----+-----+-----+
```

Native Queries

See `nobel.py`

```
#!/usr/bin/python
import MySQLdb
import os

db = MySQLdb.connect(
    os.environ['MYSQL_PORT_3306_TCP_ADDR'],
    'root',
    os.environ['MYSQL_ENV_MYSQL_ROOT_PASSWORD'],
    "nobel"
)

cursor = db.cursor()
cursor.execute("SELECT subject, yr, winner FROM nobel WHERE yr = 1960")
data = cursor.fetchall()

for winner in data:
    print "%s winner in %s: %s " % (winner[0], winner[1], winner[2])

db.close()
```

Object Relational Mappers

- Maps an Object in an application to a database table or relationship.
- Talks SQL to the database, your favorite language to you.
- Lets you point to different databases with the same syntax.
- Intelligently manages transactions to the database.

```
# SELECT * FROM nobel WHERE yr = 1960
for subject, yr, winner in session.query(Nobel).filter_by(yr=1960):
    print "%s winner in %s: %s " % (subject, yr, winner)
```

Further Reading

CS 340 The CS 340 course at OSU (titled “Databases”) is a great introduction to this topic. If you have the option to take it you should!

Lesson 15: Dev Processes & Tools

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

<p>Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our General Feedback Survey.</p>
--

Code Analysis

Code analysis tools are some of the most important tools in a developer's arsenal when it comes to finding and fixing bugs. Code analysis tools come in two flavors:

Debugging Tools

Debuggers are interactive dynamic analysis tools that are used to inspect your code as it runs.

- Print (broken) variables.
- Read and reports error messages.
- Highlight (incorrect) syntax.

CLI Debugging Tools

C/C++ Tools

- GDB
- Valgrind

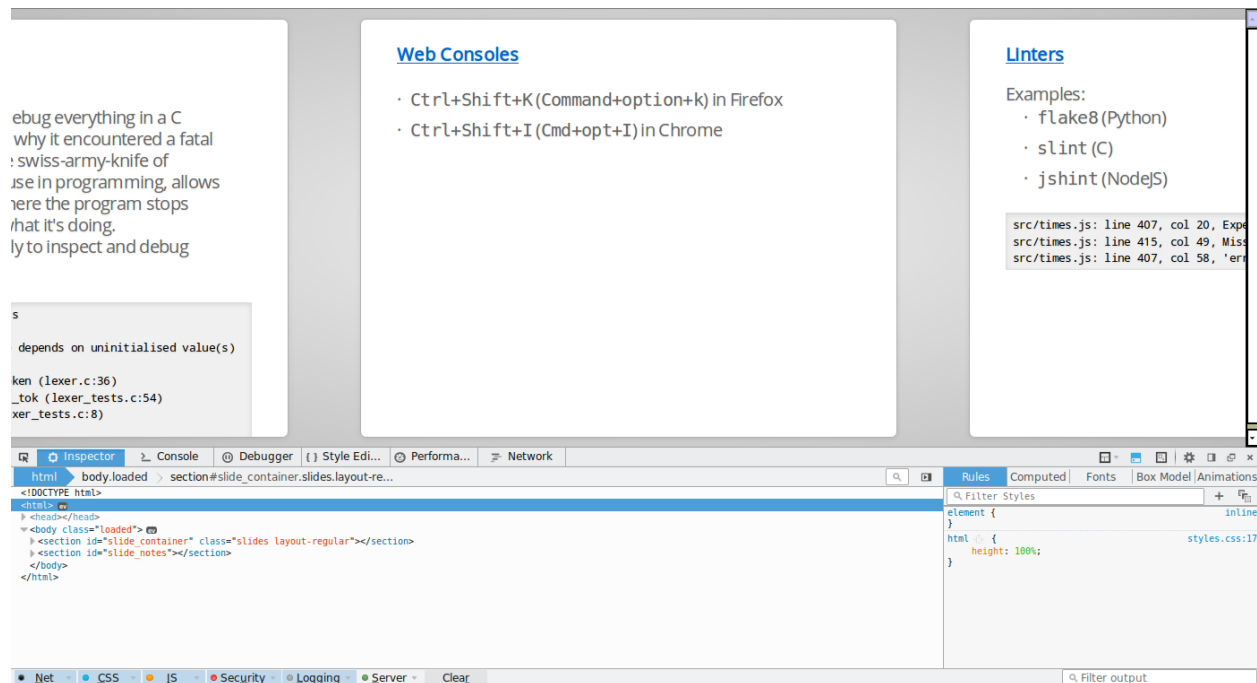
Python Tools

- PDB

NodeJS Tools

- node debug
- Node Inspector

Web Consoles



- Ctrl+Shift+K (Command+option+k) in Firefox
- Ctrl+Shift+I (Cmd+opt+I) in Chrome

Linters

Linters inspect your code and flags suspicious usage. This can be to enforce a style guide or to flag code which will probably not compile or break the program when it is running.

Examples:

- flake8 (Python)
- rubocop (Ruby)
- splint (C)
- jshint (NodeJS)

```
src/times.js: line 407, col 20, Expected '{' and instead saw 'return'.
src/times.js: line 415, col 49, Missing semicolon.
src/times.js: line 407, col 58, 'error' is not defined.
```

Code Coverage

Name	Stmts	Miss	Cover	Missing
my_program.py	20	4	80%	33-35, 39
my_other_module.py	56	6	89%	17-23
TOTAL	76	10	87%	

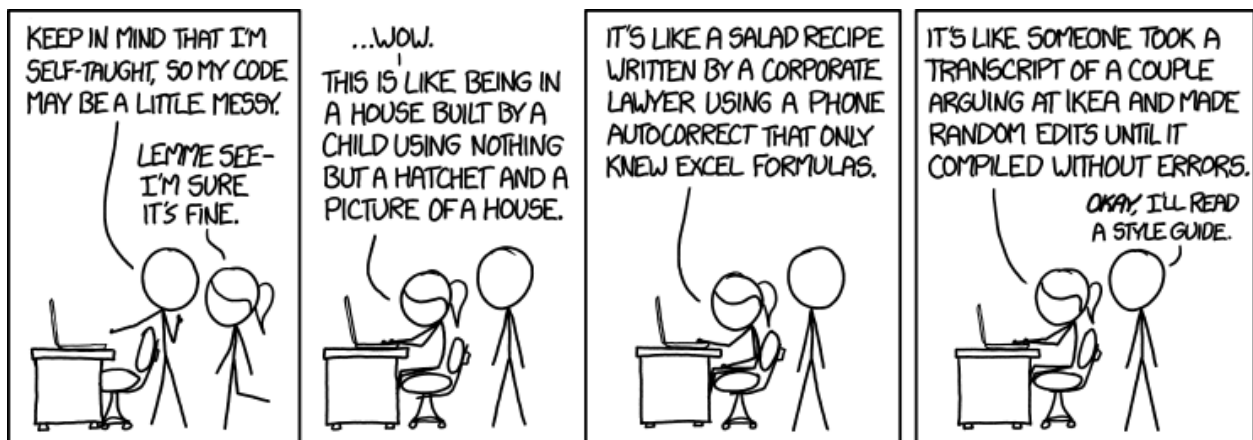
Integrated Development Environments (IDE)

IDEs are programs used to help developers get their job done by integrating many essential tools into one ecosystem.

Examples:

- **Netbeans** (Java)
- **Visual Studio** (.NET)
- **PyCharm** (Python)
- **Eclipse** (Many)
- **Atom** (Many)

Style Guides



Example: Real-World Style Guides

The Linux kernel style guidelines are actually fun to read:

“First off, I’d suggest printing out a copy of the GNU coding standards, and NOT read it. Burn them, it’s a great symbolic gesture.”

—Linux Kernel Coding Style

NASA’s Jet Propulsion Laboratory style guidelines are very short and are concerned with automated tooling to do code analysis:

“All loops shall have a statically determinable upper-bound on the maximum number of loop iterations.”

—JPL Coding Standard

Dependency Isolation

Dependency isolation is the process of – wait for it – isolating the dependencies of a project. This is a surprisingly hard problem and many consider it largely unsolved.

TODO: Python Virtualenvs

Setup and *enter* the virtual environment.

```
$ virtualenv <virtualenv name>
New python executable in /path/to/<venv name>/bin/python
Installing setuptools, pip, wheel...
done.
$ source <venv name>/bin/activate
```

Install a package. This installs it in the current working directory and so does not ask for root permissions.

```
(<venv name>) $ pip install flask
[...]
```

To list all packages in the venv:

```
(<venv name>) $ pip freeze
click==6.7
Flask==1.0.2
itsdangerous==0.24
Jinja2==2.10
MarkupSafe==1.0
Werkzeug==0.14.1
```

Deactivate (leave) the venv.

```
(<venv name>) $ deactivate
$
```

Other Examples

Node.js: Creates a `node_modules` directory and tracks dependencies in `package.json`.

Go: Dependencies are tracked via git repositories and using the `go get` command.

Rust: Dependencies and versions are specified in `Cargo.toml`. All compiled code (and dependencies) are stored in a `target` directory.

Development Servers

A Carbon Copy of the Production Environment(s)

Development servers are used to test that your code works in a real environment, with a real server, and real data. You shouldn't throw your code up on a *production* website to see if it works, so a development server is as close to the real thing as you can get.

Further Reading

- The [Community Ruby Style Guide](#) is a good resource for anybody learning Ruby. It's the style guide that [Rubocop](#) enforces.
- The [Official Python Style Guide](#) (PEP8) is a well respected style guide for Python and is commonly accepted as *the* python style guide.

Lesson 16: DNS

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Use it for learning purposes at your own peril.
If you have any feedback, please fill out our [General Feedback Survey](#).

Problems DNS Solves



Obligatory History Lesson

HOSTS.TXT circa 1977:

```
MIT          1
Yale         2
Harvard      3
ATT          4
...
```

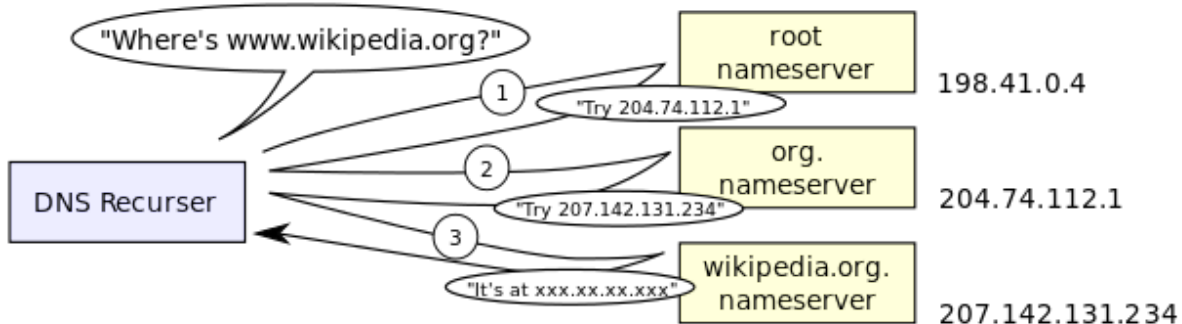
HOSTS.TXT a few years later:

```
...
joeBillson 14895
susan-gill 15832
...
```

How DNS Works

1. Computer **A** wants to fetch data from `devopsbootcamp.osuosl.org.` (notice the `.` at the end of the address).
2. Computer **A** checks the local cache.
3. If the address isn't in the cache, **A** contacts the DNS `root` server. (We're actually skipping a few layers of cache. Read up for more info on that.)
4. One of the `root` nodes tells **A** to check the `org` node.
5. The `org` node is contacted and tells **A** to check the `osuosl` node.
6. The `osuosl` node tells it to check the `devopsbootcamp` node.

A DNS Request



DNS Records

Acronym	Name
A, AAAA	IP Addresses
MX	SMTP Mail Exchangers
NS	Name Servers
SOA	DNS Zone Authority
PTR	Pointers for Reverse DNS Lookups
CNAME	Domain Name Aliases

A Records

The **A** record is used to map an IP address to a domain name. This is as close to a 'regular' record as you can get.

```
osuosl.org.      300 IN  A    140.211.15.183
```

MX Records

The MX record is for tracking mail servers.

```
osuosl.org.      3600   IN  MX   5 smtp3.osuosl.org.
osuosl.org.      3600   IN  MX   5 smtp4.osuosl.org.
osuosl.org.      3600   IN  MX   5 smtp1.osuosl.org.
osuosl.org.      3600   IN  MX   5 smtp2.osuosl.org.
```

NS Records

Servers with a NS record are allowed to speak with authority on a domain and DNS requests.

```
osuosl.org.      86258  IN  NS   ns1.auth.osuosl.org.
osuosl.org.      86258  IN  NS   ns2.auth.osuosl.org.
osuosl.org.      86258  IN  NS   ns3.auth.osuosl.org.
```

SOA (Authority) Records

SOA is the record for proving authority over a site or zone.

```
osuosl.org.      86400  IN  SOA  ns1.auth.osuosl.org. ...
```

CNAME Records

CNAME is an record for aliasing old names to redirect to new names.

```
bar.example.com. 86400  IN  CNAME foo.example.com
```

NXDOMAIN Records

Tells you there is no answer to a query:

```
Host something.invalid.osuosl.org not found: 3(NXDOMAIN)
```

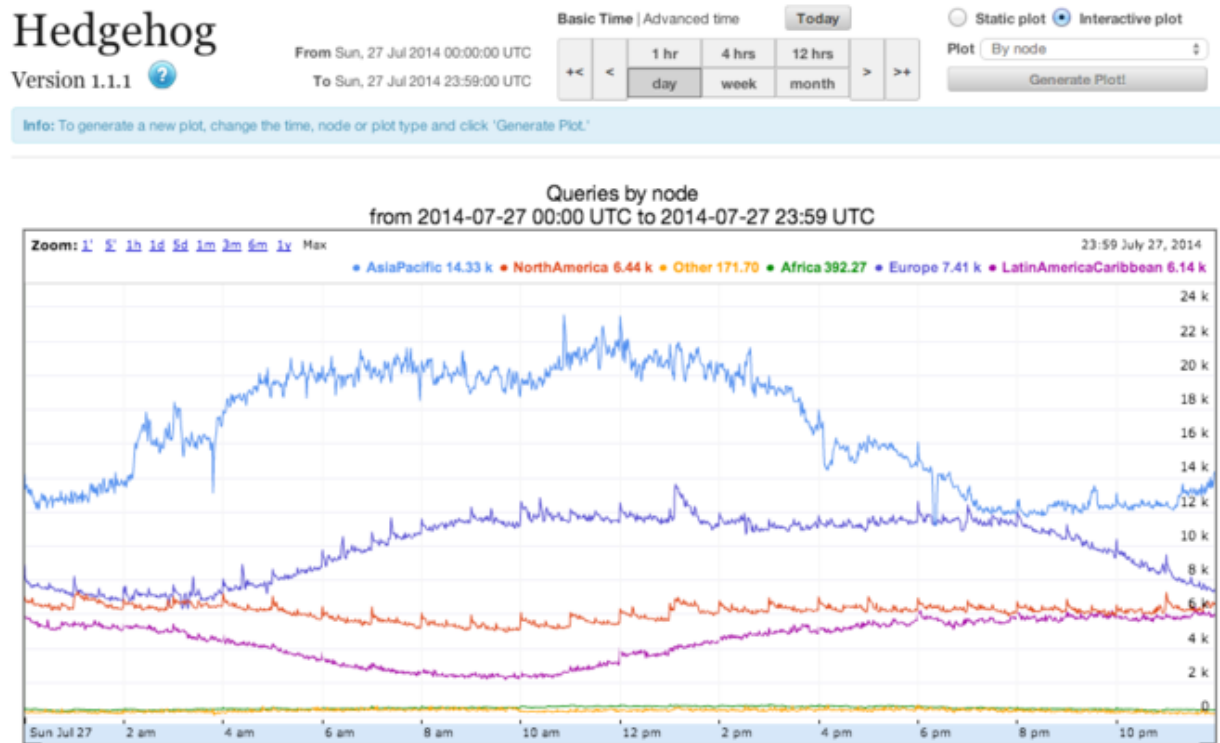
Some ISPs and others never serve NXDOMAINS, instead they point you at themselves.

The Root

```
$ dig ns .
;; ANSWER SECTION:
.          512297 IN  NS   i.root-servers.net.
.          512297 IN  NS   e.root-servers.net.
.          512297 IN  NS   d.root-servers.net.
.          512297 IN  NS   j.root-servers.net.
.          512297 IN  NS   b.root-servers.net.
.          512297 IN  NS   a.root-servers.net.
.          512297 IN  NS   f.root-servers.net.
.          512297 IN  NS   h.root-servers.net.
```

```
.      512297  IN  NS   g.root-servers.net.  
.      512297  IN  NS   c.root-servers.net.  
.      512297  IN  NS   m.root-servers.net.  
.      512297  IN  NS   k.root-servers.net.  
.      512297  IN  NS   l.root-servers.net.
```

The Thirteen



Tool: dig

dig is a command-line tool for performing DNS lookups.

Syntax:

```
dig @server name type
```

Examples:

```
dig @ns1.osuosl.org osuosl.org A
```

Example: Recursive Request

First we query a NS record for .:

```
$ dig ns .  
;; QUESTION SECTION:  
; .                IN  NS
```



```
;; ANSWER SECTION:
.           518400 IN NS i.root-servers.net.
.           518400 IN NS a.root-servers.net.
.           518400 IN NS l.root-servers.net.
.           518400 IN NS f.root-servers.net.
.           518400 IN NS b.root-servers.net.
```

etc...

Next we query NS for org.:

```
$ dig ns com. @a.root-servers.net
;; QUESTION SECTION:
;org.                IN NS

;; AUTHORITY SECTION:
org.                 172800 IN NS a0.org.afiliast.info.
org.                 172800 IN NS a2.org.afiliast.info.
```

etc...

```
;; ADDITIONAL SECTION:
a0.org.afiliast.info. 172800 IN A 199.19.56.1
```

etc...

Next we query NS for osuosl.org.:

```
$ dig ns osuosl.org. @199.19.56.1
;; QUESTION SECTION:
;osuosl.org.         IN NS

;; AUTHORITY SECTION:
osuosl.org.          86400 IN NS ns3.auth.osuosl.org.
osuosl.org.          86400 IN NS ns2.auth.osuosl.org.
osuosl.org.          86400 IN NS ns1.auth.osuosl.org.

;; ADDITIONAL SECTION:
ns1.auth.osuosl.org. 86400 IN A 140.211.166.140
ns2.auth.osuosl.org. 86400 IN A 140.211.166.141
ns3.auth.osuosl.org. 86400 IN A 216.165.191.53
```

Next we query A for osuosl.org.:

```
$ dig a osuosl.org. @140.211.166.140
;; QUESTION SECTION:
;osuosl.org.         IN A

;; ANSWER SECTION:
osuosl.org.          300 IN A 140.211.15.183

;; AUTHORITY SECTION:
osuosl.org.          86400 IN NS ns1.auth.osuosl.org.
osuosl.org.          86400 IN NS ns2.auth.osuosl.org.
osuosl.org.          86400 IN NS ns3.auth.osuosl.org.

;; ADDITIONAL SECTION:
ns1.auth.osuosl.org. 86400 IN A 140.211.166.140
ns2.auth.osuosl.org. 86400 IN A 140.211.166.141
```

```
ns3.auth.osuosl.org.      3600      IN      A      216.165.191.53
```

TODO: Traverse the DNS Tree with `dig`

Can you traverse the DNS tree to get to these websites? Give it a try!

- `github.com`
- `web.archive.org`
- `en.wikipedia.org`

Further Reading

- Try running `dig` on some of your favorite websites and see what you find.
- Read the manpage on `dig` and see what else you can find in the output.
- Try registering your own domain name and run a website using the [Github Student Pack](#) resources like Digital Ocean and DNSimple.

Lesson 17: Configuration Management

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

Configuration Management

“Configuration management is the process of standardizing resource configurations and enforcing their state across IT infrastructure in an automated yet agile manner.”

- Puppet Labs

```
user { 'audience':  
  ensure => present,  
}
```

Short History of CM

In the beginning there were no computers.

Then many years passed and eventually we built the first computer.

Then a few years after that we had more computers than we really had time to manage. Things got out of hand pretty quick.

Concept: Infrastructure as Code

- Install packages, configure software, start/stop services.
- Ensure/guarantee a specific state of a machine.
- Provide history of changes for a system.
- Repeatable way of rebuilding a system.
- Orchestrate a cluster of services together.

Pull vs Push Models

Pull Model Scales well but difficult to manage.

Push Model Simple to manage and setup but not scalable.

Tools

- Puppet
- Chef
- CFEngine
- Ansible
- Saltstack

Puppet



- Uses custom CM Language.
- Primary Push Model.
- Widely Adopted.
- Very stable.
- Difficult to get setup.

Chef



CHEF™

- Primarily Push Model.
- Code files are Ruby.
- Widely Adopted.
- Difficult to setup.

CFEngine

- Fast at execution, slow at adaptation.
- Very old.
- Stable.

Ansible



- Easy to use.
- Easy to setup.
- Does not scale well.

SaltStack



- Easy to use.
- Hard to get started.

Declaration Configuration

```
packages [nginx, python, vim]
  state installed
  update true

service nginx
  state enabled
  alert service myapp_daemon
```

Chef Example

- Install apache and start the service
- Configuration is called a ‘recipe’
- Written as pure Ruby code

```
package "apache" do
  package_name "httpd"
  action :install
end

service "apache" do
  action [:enable, :start]
end
```

Note: Since chef uses Ruby you can do loops and other cool Ruby-isms in your configuration management. This can be a gift and a curse.

Puppet Example

- Install apache and start the service
- Configuration is called a ‘manifest’
- Puppet DSL based on Ruby

```
package { "apache":
  name      => "httpd",
  ensure    => present,
}

service { "apache":
  name      => "apache",
  ensure    => running,
  enable    => true,
  require   => Package["apache"],
}
```

Note: Since Puppet designed its own language you are more limited in what you can express, but this isn’t always a bad thing. It’s feature rich and can do pretty much anything that Chef can.

Ansible Example

- Install apache and start the service
- Configuration is called a ‘playbook’
- Uses YAML file format for configuration

```
- hosts: all
  tasks:

    - name: Install Apache
      yum:
        name: httpd
```

```
state: present

- name: Start Apache Service
  service:
    name: httpd
    state: running
    enabled: yes
```

Note: Ansible’s *language* is Yaml, which is basically JSON but easier to read and write. This is similar to Puppet in it limits the possible functionality, but again: these tools all achieve the same result, they just get there in different ways.

Further Reading

- [Ansible’s Documentation](#) is comprehensive and contains an easy-to-follow “Getting Started” guide.
- [Kitchen-CI](#) is a Chef oriented testing system
- [Puppet Learning VM](#) is a prebuilt VM for learning Puppet’s fundamental concepts

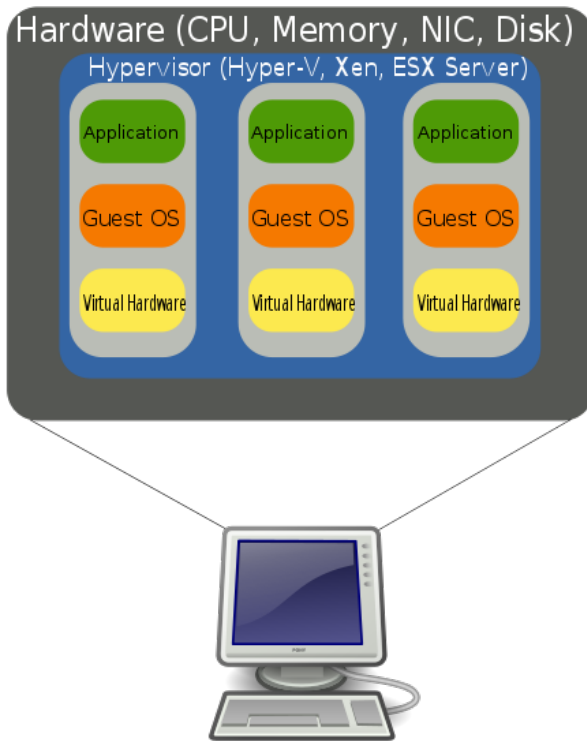
Lesson 18: Application Isolation

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

<p>Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our General Feedback Survey.</p>
--

Application Isolation

Virtual Machines



```
[vm] # ps aux
USER PID %CPU %MEM    VSZ   RSS TTY  STAT  START   TIME COMMAND
root   1   0.0   0.6 110564   3164 ?    Ss   2015   11:17 /lib/systemd/systemd --system --deserialize 15
root   2   0.0   0.0     0      0 ?    S    2015    0:00 [kthreadd]
root   3   0.0   0.0     0      0 ?    S    2015    3:55 [ksoftirqd/0]
root   5   0.0   0.0     0      0 ?    S<   2015    0:00 [kworker/0:0H]
[... 120+ more lines ...]
```

```
[host] # ps aux
USER  PID %CPU %MEM    VSZ   RSS TTY  STAT  START   TIME COMMAND
root   1   0.0   0.1 200328   5208 ?    Ss   Aug25   0:44 /sbin/init
root   2   0.0   0.0     0      0 ?    S    Aug25   0:00 [kthreadd]
root   3   0.0   0.0     0      0 ?    S    Aug25   0:05 [ksoftirqd/0]
root   5   0.0   0.0     0      0 ?    S<   Aug25   0:00 [kworker/0:0H]
[... 240+ more lines ...]
```

OS Emulation

Containers

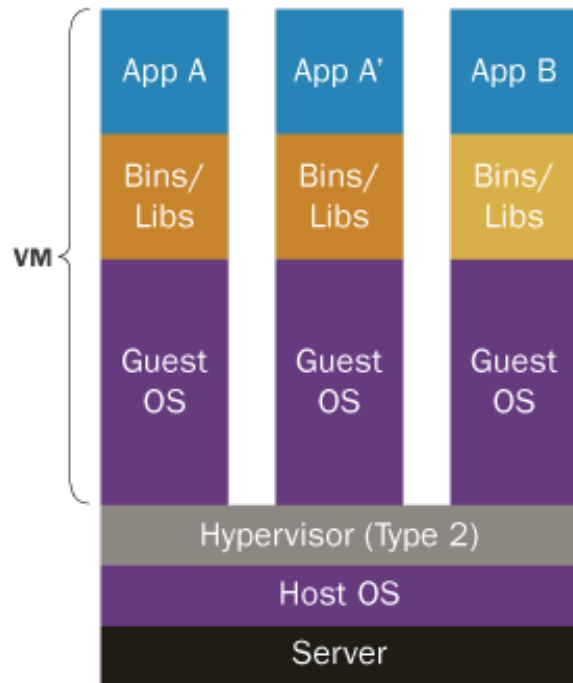
```
[container] $ ps aux
PID    USER      TIME  COMMAND
1      root      0:00  /bin/sh
```

```
1    root    0:00    sh
6    root    0:00    ps aux
```

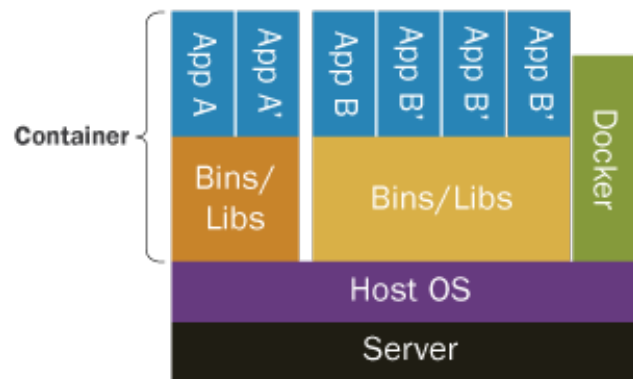
Container Technologies

Containers vs VMs

Containers vs. VMs



Containers are isolated, but share OS and, where appropriate, bins/libraries



Pros

Virtual Machines	Containers
Complete process isolation 'Battle Tested'	Fast startup Little overhead

Cons

Virtual Machines	Containers
Slightly more overhead. Slow startup. Cross-kernel emulation.	Security concerns. No cross-kernel emulation.

Tools

Virtual Machines	Containers
VirtualBox	Docker
VMWare	Rkt

Virtual Machines

VirtualBox An Open Source VM Manager.

Widely used and supported on Linux, Mac, and Windows.

VMWare A closed source VM Manager.

VMWare is a widely used and tends to have better performance than Virtual Box. While it can emulate Linux it does not work natively on Linux.

KVM The Kernel-based Virtual Machine.

Linux's native infrastructure for handling Virtual Machines and emulation. Usually used in a larger emulation program, not alone.

Containers

Docker The de facto CLI tool for creating and using containers.

Very popular and well integrated into other tools.

RKT A competitor to Docker created by CoreOS. Approaches container management from a different angle which has it's advantages and disadvantages.

chroot The *oldschool* way to use containers. Not a container in the modern sense, but achieves similar process isolation.

Jails The BSD Unix form of containerization. Offers a level of secure isolation not really possible in Linux.

TODO

Further Reading

Docker

RKT

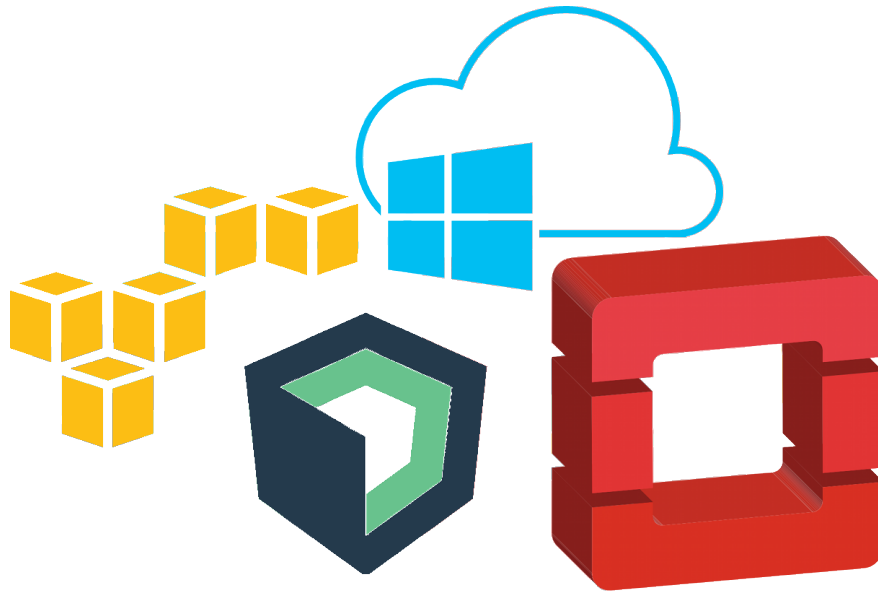
Lesson 19: Cloud Infrastructure

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

What the Cloud Looks Like

[...] a model for enabling ubiquitous, on-demand access to a shared pool of configurable computing resources.



Advantages over Bare Hardware

Private Clouds

Public Clouds

Cloud + Configuration Management

Advantages

Running your software on a cloud is:

- **Ephemeral**
- **Cost effective**
- **Low startup cost**

Disadvantages

Clouds can be great tools, but they have some disadvantages:

- **Central Point of Failure**

TODO

Further Reading

AWS provides a *lot* of services, not all of which are named very well. [This article](#) explains what each service does in plain English.

Lesson 20: Contributing to Open Source

Homepage	Content	Slides	Video
--------------------------	-------------------------	------------------------	-----------------------

Warning: This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

Open Source

- Learn lots of new things, and grow as developers.
- Give back to a community that has given you something.
- You have more to contribute than you may realize!
- Meet amazing people.
- Personal fulfillment.

Community Benefit

Share the Love (and the Code)

Personal Benefit

- ‘Learning the Ropes’ of a substantial code-base
- Working with others
- Getting code reviewed
- Documenting contributions
- Testing your changes

Free?

Free Software: *[Free Software] means that the users have the freedom to run, copy, distribute, study, change and improve the software.*

The Four Freedoms:

0. The freedom to run the program as you wish, for any purpose.
1. The freedom to study how the program works, and change it so it does your computing as you wish. Access to the source code is a precondition for this.

2. The freedom to redistribute copies so you can help your neighbor.
3. The freedom to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

Assessing a New Community

Elitism vs Nice-ism

Communication style

Documentation and Guides

Things to Look for

- When are the top pull requests time-stamped? Anything older than 3-4 months might not be ideal.
- Open / recent issues (especially with help wanted labels) are good.
- Many contributors means they're used to people helping out.

How to Get Involved

Finding a Project

In order of perceived usefulness:

- [Openhatch](#)
- [24 pull requests](#)
- [BugsAhoy](#)
- [Showcased github projects](#)
- [Trending github projects](#)

I Can't Find a Project I Like!

That's okay.

First Steps

0. Find a project
1. Read Contributing and Getting Started docs
2. Look at list of issues
3. Do a thing!
 - Write a test
 - Fix a typo
 - Deploy and update the installation docs

Know your Licenses



Licenses to use:

- MIT
- Apache 2.0
- AGPL/GPL/LGPL 2/3

Licenses to **not** use:

- Public Domain Dedication

TODO: Find an Open Source Project

Further Reading

[Choose A License](#)

About

What is DevOps?

DevOps is a hybrid of skills from both Software Development (Dev) and Computer Operations (Ops) intended to meet the unique demands of [cloud computing](#). *Software Developer* and *Systems Administrator* are no longer mutually exclusive job titles. Devs need more Ops knowledge to understand how their application will run in the real world. Admins need more Dev knowledge to design infrastructure that fit an app's needs efficiently and effectively. To top it off site reliability engineers and many modern security roles require at least a little background in both development and operations.

Purpose of DevOps BootCamp

DevOps BootCamp is an [OSU Open Source Lab](#) program dedicated to teaching core software development and systems operation skills. The program is free and open to any interested OSU students, community member, and online go-getter. DevOps BootCamp provides a comprehensive Open Source education that is outside the scope of regular Linux Users Group meetings and OSU Coursework.

What Students Get

- Mentorship from students and professionals with advanced skills in software development and systems administration.
- Professional connections in the software industry.
- A welcoming environment to start learning, for those who have always wanted to learn about software development and systems administration but were never sure where to start.
- An opportunity to fill in knowledge gaps for self-taught coders or sysadmins.
- The skills to build and deploy Open Source software, or contribute to existing projects

What the Open Source Lab Gets

- The OSL gets a larger pool of candidates to recommend to companies interested in recruiting students.
- The OSL gets to work with a wider variety of students, helping it contribute to the school of EECS.
- The Open Source Community gets more project contributors.

Target Audience

Our goal is to make the DevOps BootCamp program accessible to students and community members from all backgrounds. Students should:

- Want to learn.
- Be willing to ask questions
- Be open to setting apart time to play with the tools you'll be learning about in the class.

Policies

Attendance

Attendance is not mandatory but highly suggested to get the most out of DOBC; We will not spend class time reviewing material for those who skip a lecture and each classes curriculum will build on what you learned the previous session. All curriculum will be available online before and after class sessions to get caught up.

BootCamp mentors will be available at scheduled times outside of regular classes to help answer any questions about the training program's content. If you attend a lesson and don't understand something then you are encouraged to ask that question during the meeting since others are likely have the same question.

Laptops

As the course progresses, you will need a laptop. We hope and recommend that you decide to set up your laptop to dual-boot to Linux as the course progresses, but it is not required. If you don't own a laptop and are an OSU student you can check out a laptop from the OSU Library for at least 24 hours at a time.

As long as your laptop is new enough to boot from USB and connect to a wireless network the exact specifications do not matter. You will be provided with a remote virtual machine with which to do all class projects.

If you are not an OSU student and do not have access to a working laptop, contact the DevOps BootCamp ([email devopsbootcamp](mailto:devopsbootcamp)) organizers and they will see whether one can be loaned out to you.

Get Involved

Mailing list

Join the [mailing list](#) for updates.

Slack

Feel free to also join us on [Slack](#). If you [sign up](#) using your OSU email address, you'll automatically get added without any invitation. If you don't have an email address from the allowed domains, please let us know and we'll get you invited!

IRC

Join us on `irc.freenode.net` in `#devopsbootcamp` (students will be setting up an IRC network for the program early in the program).

Website & Curriculum

If you'd like to help edit this site, [email devopsbootcamp](#) or ping anyone in `#devopsbootcamp` on Freenode with your GitHub username to get access to the web site repo. You'll also want to learn the [ReStructured Text](#) markup language to edit the site, if you don't already know it.

Setting up SSH

Secure Shell (SSH) provides a secure channel to access a Linux machine remotely via command line.

Windows

Windows doesn't come with an ssh client natively, however you can download and install [PuTTY](#) to give you a nice SSH client.

Linux and Mac

If you're already running Linux or have a Mac laptop, you already have an ssh installed. For the Mac, simply open up the Terminal.app. For Linux, you can use something like Gnome Terminal, Terminator or xterm to name a few.

Setting up Docker

[Docker](#) is a software technology which provides the use of containers which is kind of a light form of virtual machines. It's used quite a bit in DevOps to setup development environments along with a variety of other uses. In addition to Docker, we're going to be using [Docker Compose](#) which is used for running multi-container environment. For DevOps Bootcamp, we're going to be using Docker in a variety of ways from acting as a simple Linux machine, to hosting applications.

Installing Docker

Docker can be installed on [Windows](#), [Mac](#) and a variety of Linux operating systems ([Ubuntu](#), [Debian](#), [CentOS](#), [Fedora](#)). Please be sure you read the installation instructions closely to ensure your system supports running Docker and has the needed BIOS features enabled. If you have any trouble getting it installed, feel free to ask in our Slack channel.

Installing Docker Compose

Docker Compose can be [installed](#) on Windows, Mac and a variety of Linux operating systems. Please read the installation instructions for your platform *carefully*.

Running the DOBC image

First you need you need to clone the [Bootcamp-Exercises](#) repository:

```
$ git clone https://github.com/DevOpsBootcamp/Bootcamp-Exercises.git
```

Once you have Docker and Docker Compose installed and running and also have the [Bootcamp-Exercises](#) repository cloned, you can spin up a Docker image we've created for DOBC by running the following from the root of the repository directory:

```
$ cd Bootcamp-Exercises
$ docker-compose up -d
$ docker-compose run -p 8080:8080 dobc bash
```

You can log out by typing `exit` and then enter which will stop the container.

Stopping the container

To stop the container, run the following:

```
$ docker-compose kill
$ docker-compose rm --all
```

Schedule

The DevOps BootCamp content is available for free but meet-space guided lectures are offered throughout the year. Check the schedule below for our in-person lectures; each lecture covers a different part of the curriculum covering the entire course during the OSU academic school year.

Warning: If you are working ahead be aware that the schedule and slides may be subject to change. Check back regularly.

Fall

Lessons Covered	Date/Time	Location	Description
0 - 7	Oct 27, 2018 9:30am-3:30pm	OSU KEC 1001	DevOps BootCamp Fall Kickoff

Open Office Labs

Each lab has two time slots (please choose one) to help assist with students being able to attend. All labs will in [Milne 224](#).

Description	Slot 1	Slot 2
Lab #1	Oct 31, 2018 10am-12pm	Nov 1, 2018 2-4pm
Lab #2	Nov 14, 2018 10am-12pm	Nov 15, 2018 2-4pm
Lab #3	Nov 28, 2018 10am-12pm	Nov 29, 2018 2-4pm

Running DOBC

If you're reading this it means you're interested in running DOBC yourself. It may have been passed on to you, or you may just like the curriculum and want to use it to start your own DOBC. Either way, thank you for reading this!

This page is a growing checklist, warning, notes, and fables from those teaching and contributing to DOBC. If you read this, heed it's warnings and take it's lessons to heart you will no doubt be on your way to success.

Before You Begin

Meet-Space Lectures

Practice. Just like any public speaking engagement, you should review what you're going to do and practice it in *real time*. This means you should say the things you're going to say and even do the activities the students will do.

Always have a buddy. Teaching alone can be done, but if at all possible try to have a teaching 'buddy'. This person is at least about as expert on the topics you're covering as you are. Your buddy can field questions, help with TODOs, and can even take over the lesson if you need them to (you might need to go to the bathroom, who knows).

Always be taking notes. As a lecturer you won't always teach perfectly. You won't get it perfect the first, second, third, or even last time – but you should always strive for perfection.

Take notes on what could have gone better, questions that were asked, and confusions students had. The DOBC curriculum can be very dense and sometimes it skims over important stuff. During each lesson be sure to improve the curriculum based on your notes. Whoever teaches it next time will thank you.

Online Engagement

Adding to Lessons

Attribute Images We do our best to attribute images by linking them to the website we got them from. If you add images to the website try to do the same.