

---

# **OSU DevOps BootCamp Documentation**

***Release 0.0.1***

**OSU OSL  
OSU LUG**

November 19, 2016



<b>1</b>	<b>Ready to Learn DevOps? <i>Lesson 0: Start Here</i></b>	<b>3</b>
<b>2</b>	<b>Schedule</b>	<b>5</b>
2.1	Fall . . . . .	5
2.2	Winter . . . . .	5
2.3	Spring . . . . .	5
<b>3</b>	<b>Donate</b>	<b>7</b>
3.1	Lesson 0: Start Here . . . . .	7
3.2	Lesson 1: First Steps . . . . .	10
3.3	Lesson 2: Operating Systems . . . . .	14
3.4	Lesson 3: Shell Navigation . . . . .	17
3.5	Lesson 4: Users, Groups, Permissions . . . . .	20
3.6	Lesson 5: Files . . . . .	23
3.7	Lesson 6: Packages, Software, Libraries . . . . .	26
3.8	Lesson 7: Questions, Answers, Docs . . . . .	28
3.9	Lesson 8: Version Control . . . . .	33
3.10	Lesson 9: Programming . . . . .	38
3.11	Lesson 10: Frameworks . . . . .	45
3.12	Lesson 11: Testing . . . . .	47
3.13	Lesson 12: Continuous Integration . . . . .	50
3.14	Lesson 13: Security . . . . .	52
3.15	Lesson 14: Databases . . . . .	56
3.16	Lesson 15: Dev Processes & Tools . . . . .	63
3.17	Lesson 16: DNS . . . . .	67
3.18	Lesson 17: Configuration Management . . . . .	72
3.19	Lesson 18: Application Isolation . . . . .	78
3.20	Lesson 19: Cloud Infrastructure . . . . .	81
3.21	Lesson 20: Contributing to Open Source . . . . .	82
3.22	About . . . . .	86
3.23	Schedule . . . . .	87
3.24	Running DOBC . . . . .	88



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!

---





---

## 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.

### 2.1 Fall

Lesson	Date/Time	Location	Description
DevOps Daycamp	Oct 1, 10am-3pm	<a href="#">OSU KEC 1001</a>	DevOps DayCamp (DOBC Kickoff)
Fall Meeting 2	Nov 5, 11am-3pm	<a href="#">OSU KEC 1001</a>	Files, Version Control, Programming
Fall Meeting 3	Dec 3, 11am-3pm	<a href="#">OSU KEC 1001</a>	Frameworks, Testing and CI

### 2.2 Winter

Lesson	Date/Time	Location	Description

### 2.3 Spring

Lesson	Date/Time	Location	Description

---



---

**Donate**

---

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

## 3.1 Lesson 0: Start Here

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

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

### 3.1.1 About the Program



devopsbootcamp.osuosl.org

#### *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

#### 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!

## 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.

### 3.1.2 Getting Involved

#### Where To Ask Questions

- Internet Relay Chat
- Mailing lists
- 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.

### 3.1.3 The OSU Open Source Lab is Hiring

*For more information check the [OSL Hiring Page](#) regularly.*



## 3.2 Lesson 1: First Steps

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

**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](#).

### 3.2.1 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.

#### 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.
- **Containers:** Not virtual machines, but basically virtual machines.

### TODO: What Vocabulary Do *You* Know?

*What other vocabulary can you think of related to DevOps?*

*What about Silicon Valley, Programming, System Administration, etc?*

---

**Note:** This is a **TODO**. It’s basically an exercise or activity but with a cheeky name. Try them out if you don’t feel confident in a topic.

---

## 3.2.2 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!')
```

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

## TODO: Reading Examples

*Trick question: how would you read this*

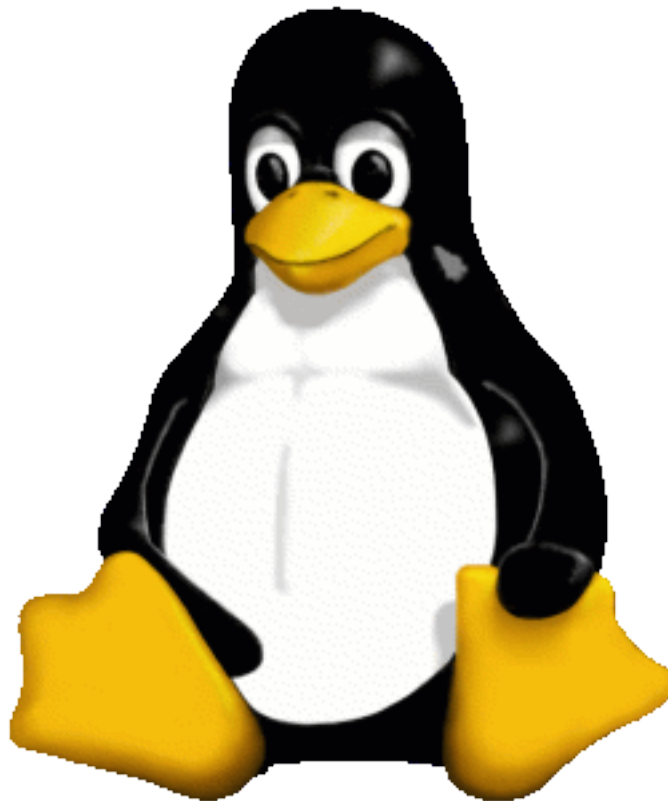
```
#!/bin/python
dogs = ['$BREED_ONE', '$BREED_TWO', '$BREED_THREE']
for breed in dogs:
    print(breed)
```

## Answer: Reading Examples

Replace the \$BREED\_N with actual dog breeds.

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

## 3.2.3 Getting Setup on Linux



### Lecture Setup

1. Get login credentials from your lecturer.
  - You will be provided a username, password, host, and port.



**Linux/Mac:**

2. Open a terminal and verify you have `ssh` installed by entering the command `ssh --version`.
3. Run `ssh -p <port> <username>@<host>` and enter the password when prompted (it will hide your password in the terminal).

**Windows:**

2. Install an SSH Client ([install Putty](#))
3. Log into your remote Linux environment using the credentials given to you.
  - (a) Under Host Name (or IP address) enter `<user>@<host>`, under Port enter `<port>`.
  - (b) You will be prompted for your password in new window, it will hide the password as you type it.

**Home Setup**

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

You may also need to [install VirtualBox](#), a tool necessary for Vagrant to function.

**TODO: 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.

**3.2.4 Further Reading**

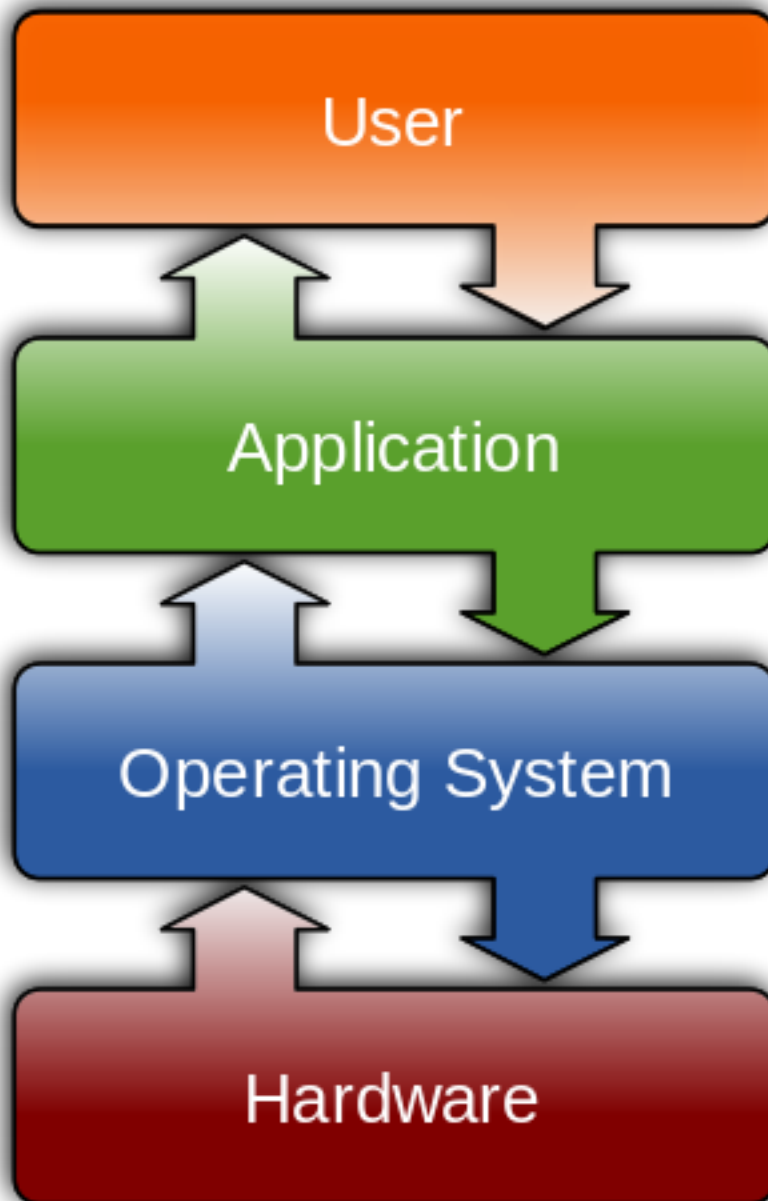
- More information on [Virtual Machines](#).
- [Install Putty](#) if you want to access a remote Linux box.
- [Install Vagrant](#) if you want to run a local Linux Virtual machine.
- [Install VirtualBox](#) in addition to Vagrant for local virtual machines.

## 3.3 Lesson 2: Operating Systems

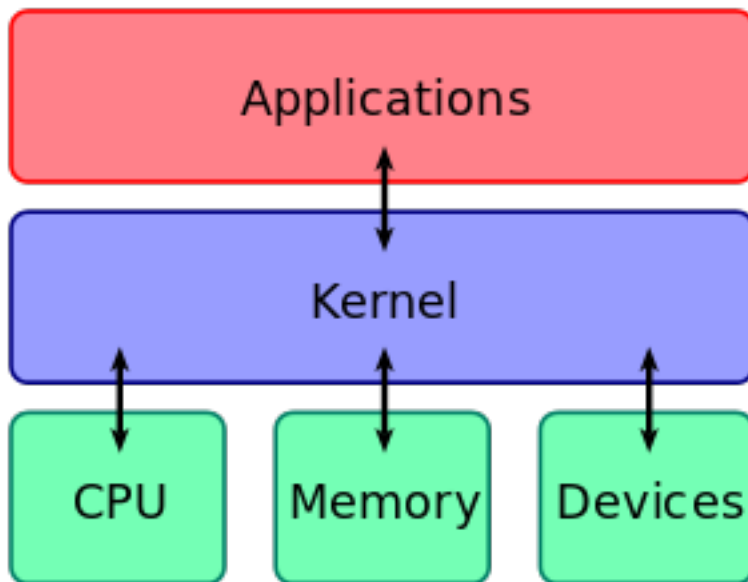
<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

**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](#).

### 3.3.1 What an Operating System is



### 3.3.2 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.

### 3.3.3 Types of Operating Systems

#### 3.3.4 Popular Operating Systems

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

#### 3.3.5 GNU/Linux

*Welcome to the Family*



### 3.3.6 Flavors of Linux

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

### 3.3.7 TODO: 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?

### 3.3.8 Further Reading

#### 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

##### 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](http://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.

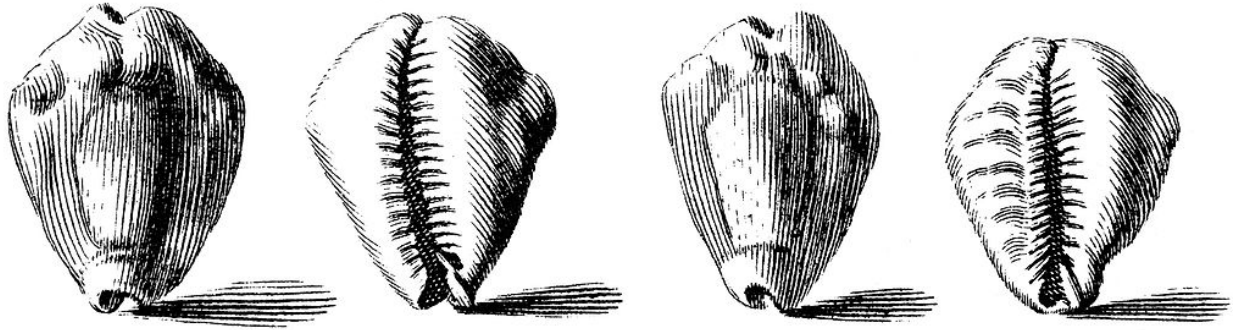
## 3.4 Lesson 3: Shell Navigation

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

**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](#).

### 3.4.1 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

**fish** Useful but not **sh** compliant shell.

**zsh** The *hip new shell on the block*.

## 3.4.2 Navigation Concepts

### 3.4.3 Basic Shell Commands

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

### 3.4.4 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:

```
$ chmod +x about_me.sh # Tell Linux that this can be run as a program.
$ ./about_me.sh        # Invoke the script.
```

### 3.4.5 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
```

### 3.4.6 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 (`\`).

### 3.4.7 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
```

### 3.4.8 TODO

### 3.4.9 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.

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

## 3.5 Lesson 4: Users, Groups, Permissions

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

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

### 3.5.1 The User

You... ish.

```
$ 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
```

Sometimes robots are users too: Apache, Mailman, ntp.

#### What a User has

/etc/passwd:

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

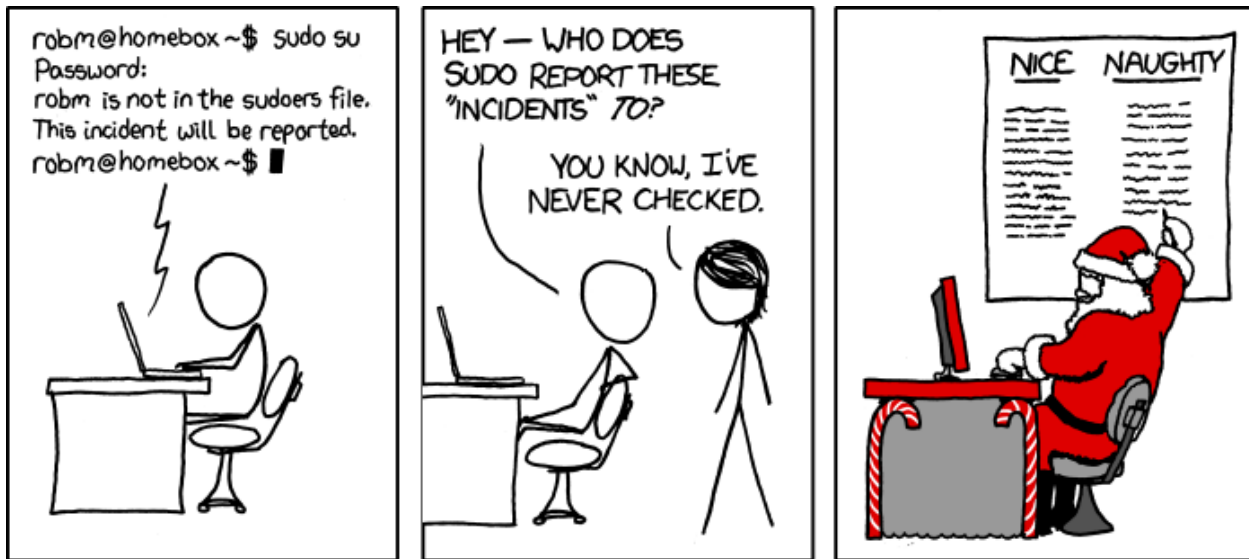
#### What Users Can Do

- Change Passwords with the `passwd` command.
- Act as Another user with `su`.

```
$ su $USER      # become user, with THEIR password
$ su            # become root, with root's password
$ sudo su -     # use your password instead of root's
$ sudo su $USER # become $USER with your password
```

- Act as themselves.
  - `ls -l` to see file permissions.
  - Check the file's group and user.
  - Check the file's read, write, and execute bits.



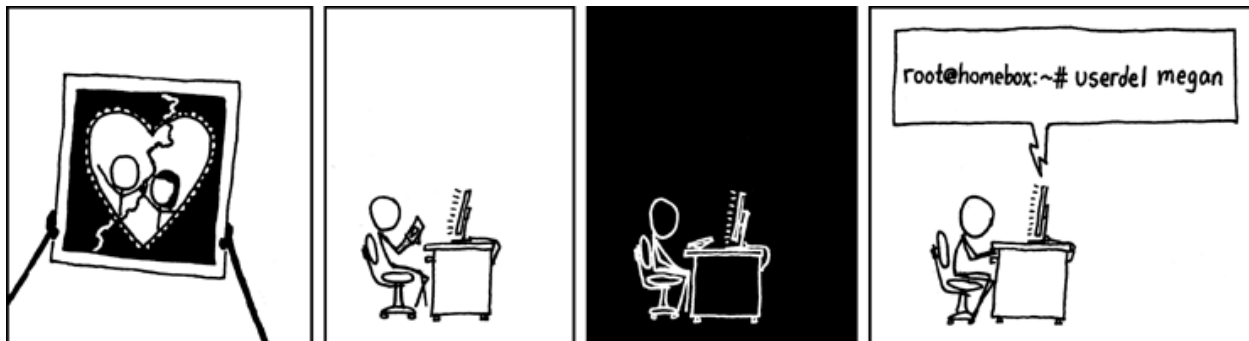


### 3.5.2 Managing Groups and Users

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

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

$ groupadd
$ usermod
$ groupmod
$ cat /etc/group
root:x:0:
```



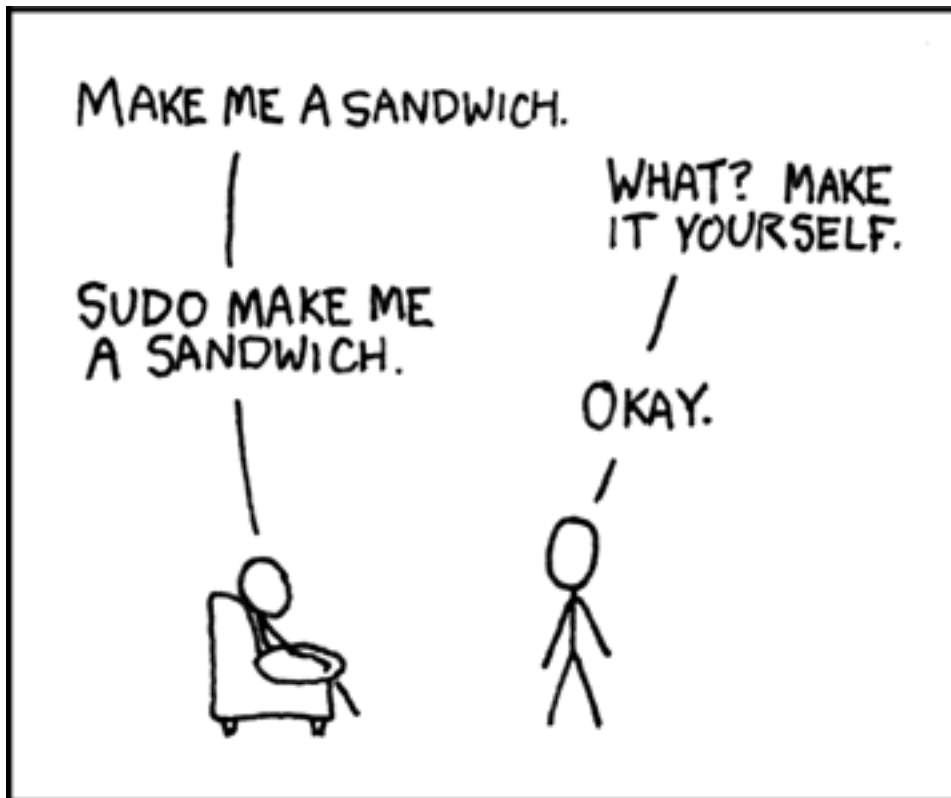
### 3.5.3 Examples of Non-Human Users

- mailman: For the mailing list program.
- apache: For the HTTP Server.
- postfix: For the other mail program.

### 3.5.4 Root and Sudo

**Root:** Basically god on Linux.

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



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

### 3.5.5 TODO

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

### 3.5.6 Further Reading

- [Understanding Linux File Permissions](#)

## 3.6 Lesson 5: Files

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

**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](#).

### 3.6.1 About Files

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

Files have:

- Owners
- Permissions (what different people can do with it)
- An inode (a low-level description of the file)
- Size
- 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
```

### 3.6.2 Everything is a file!?

Yes. Except the things that aren't..

```
int read_medical_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;
}
```

### 3.6.3 File Extensions

.jpg, .txt, .py

Not necessary, more of a recommendation.

```
$ ls
some_text_file  squirrel

$ file some_text_file
some_text_file: ASCII text

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

### 3.6.4 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
```

### 3.6.5 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
+-----								+-----	File Permissions

### 3.6.6 Editing Metadata

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

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

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

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

### 3.6.7 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

### 3.6.8 Executing a File?

For instance:

```
$ ls -alh 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!
```

### 3.6.9 Types of Files

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

### 3.6.10 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

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

### 3.6.11 TODO: Messing with Files

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

- Create an empty file in `/home/$yourusername/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.

### 3.6.12 Further Reading

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

## 3.7 Lesson 6: Packages, Software, Libraries

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

**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](#).

### 3.7.1 Software

Everything that isn't hardware.

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

### 3.7.2 Libraries

- Often used to make development easier.
- Rarely run on it's own.

- Shared code.

### 3.7.3 Package Management

- Automatically manage software and libraries on your system.
- Examples:
  - Android Play Store
  - Apple App store
  - Steam

### 3.7.4 Core Package Management Functionality

*TLDR: To take care of installation, removal, and updates of software.*

### 3.7.5 Popular Linux System Package Managers

Popular Linux Package Managers:

**Apt** ( `.deb`, `dpkg` ) Used by default on the Debian, Ubuntu, Linux Mint operating systems.

**Yum** ( `.rpm`, `rpm` ) Used by default on the RedHat, CentOS, Fedora operating systems.

### 3.7.6 Programming Language Package Managers

Examples:

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

### 3.7.7 Other Package Managers

**Portage** The Source-based package manager for Gentoo.

**Yaourt** An Arch User Repository wrapper for Pacman, the 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.

### 3.7.8 Installation from Source

#### How to install a package from source:

Using `grep` as an example:

```
$ wget http://mirrors.kernel.org/gnu/grep/grep-2.25.tar.xz
$ tar -Jxvf grep-2.25.tar.xz
$ cd grep-2.25
$ ./configure --prefix=$HOME/bin/
$ make
$ make install
```

### 3.7.9 TODO: Install `sl`

- Install the `git`, `gcc`, `make`, `ncurses-bin`, `ncurses-base`, `libncurses5-dev`, and `libncurses5-dev` packages via package manager.

```
$ sudo apt install git gcc make ncurses-bin ncurses-base libncurses5-dev libncurses5-dev
[...]
```

- Install `sl` from source into the directory `~/bin/`.

```
$ git clone https://github.com/mtoyoda/sl.git
[...]
```

```
$ cd sl
$ make
gcc -O -o sl sl.c -lncurses
$ mkdir ~/bin
$ ln sl ~/bin/
$ echo "export PATH=$PATH:$HOME/bin" >> ~/.bashrc
$ source ~/.bashrc
$ whereis sl
sl: /home/username/bin/sl
$ sl
```

### 3.7.10 Further Reading

- [More about APT](#)

## 3.8 Lesson 7: Questions, Answers, Docs

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

**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](#).

### 3.8.1 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.

### 3.8.2 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

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.

---

### 3.8.3 Project Docs

Where to look:

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

### 3.8.4 Questions and Answers

- Stack Overflow
- Forums
- Mailing Lists
- Blogs

NEVER HAVE I FELT SO  
CLOSE TO ANOTHER SOUL  
AND YET SO HELPLESSLY ALONE  
AS WHEN I GOOGLE AN ERROR  
AND THERE'S ONE RESULT  
A THREAD BY SOMEONE  
WITH THE SAME PROBLEM  
AND NO ANSWER  
LAST POSTED TO IN 2003



### 3.8.5 How to Talk to People

- Chatrooms
- Meetups
- Face to Face (!?)

### 3.8.6 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!

#### TODO 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
```

#### Connecting and Setup

In the IRC client run these commands

```
/connect irc.freenode.net

/nick <myawesomenickname>
/msg nickserv register <password> <email>

/nick <myawesomenickname>
/msg nickserv identify <password>

/join #devopsbootcamp
```

## Commands and Tips

- `/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>`
- 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

# Jargon

**(noun)** specialist vocabulary for a particular subject or profession.

- **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 eachother.

- **kick/ban/k-line:** GTFO

### 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 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

### 3.8.7 Further Reading


- **About `info`:** `info` is an alternative to `man` that some distros use instead.

## 3.9 Lesson 8: Version Control

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

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

### 3.9.1 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)

### 3.9.2 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

### 3.9.3 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

```
$ sudo yum install 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:

```
$ git branch      # Shows your branches and current branch
* master
$ git checkout -b <new-branch>  # Switches to new branch '<branch name>'
$ git branch
master
* new-branch
$ git checkout master  # Switches to existing branch '<branch name>'
```

### TODO: Working With a Git Repository

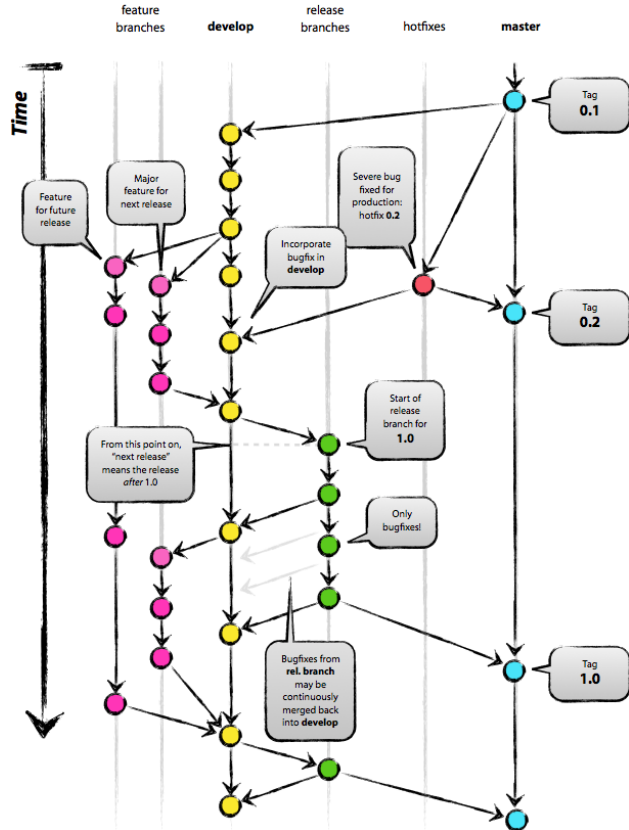
- Checkout a new feature branch on your repository.
- Create/Edit files on the new branch.
- Create a diff between the two.
- Locally merge the changes from your new branch into Master.



### 3.9.4 What not to do with Git

### 3.9.5 Workflow(s)

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



### 3.9.6 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 feature than the previous three. Open Source, useful for learning the nitty-gritty Git *really* works.

### Cloning a Repository

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

## TODO: Cloning Exercise

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

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

```
$ cd tiny-flask-app
$ virtualenv venv
$ pip install -r requirements.txt
$ python script.py
```

Now if you go to <your ip address>:<http port> in your web-browser to see a live version of the app!

## 3.9.7 Further Reading

**The Online Git Docs** This is a portal to all of the official docs on [git-scm.com](http://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.

## 3.10 Lesson 9: Programming

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

**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](#).

### 3.10.1 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(nebraska))
>>> -> 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"
```

## 3.10.2 TODO: Write Pseudo-Code

Write pseudo-code to do the following tasks:

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

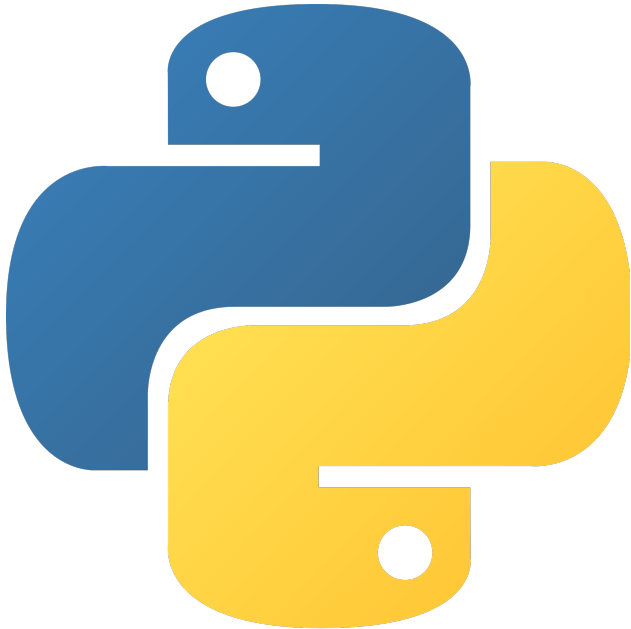
- Generate prime numbers.

**Hints:**

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

### 3.10.3 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!
```

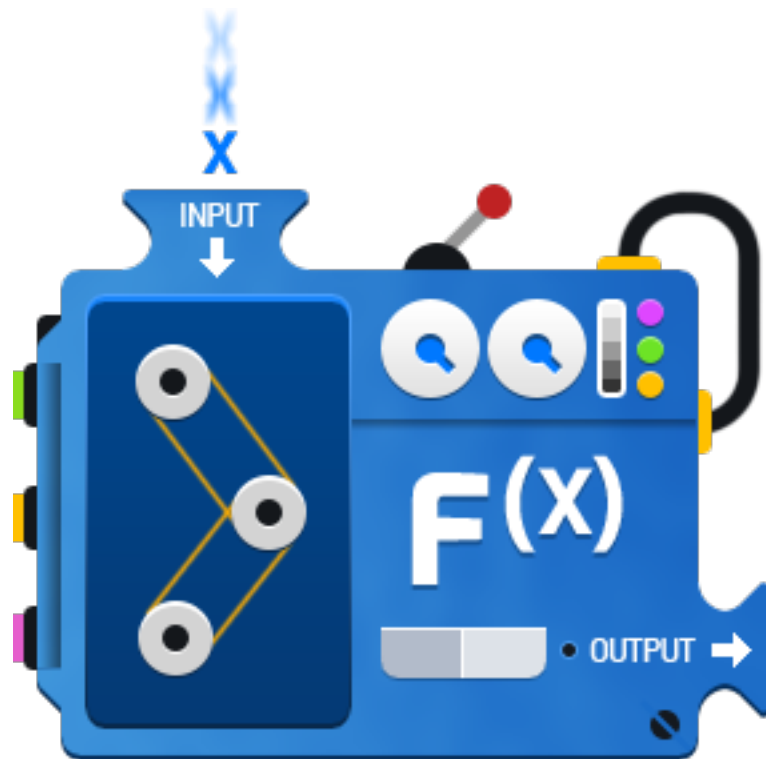
## Python Control Flow

```
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](http://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
```



### 3.10.4 TODO: Practicing Python

Formalize the last TODO by writing them in Python.

Prove the program works by running the code!

### 3.10.5 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.



## 3.11 Lesson 10: Frameworks

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

**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](#).

### 3.11.1 Frameworks

- Web-development frameworks.
- Game-development 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.

#### Types of Frameworks

- Testing Frameworks
- Web-app Frameworks
- Game Frameworks

### 3.11.2 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

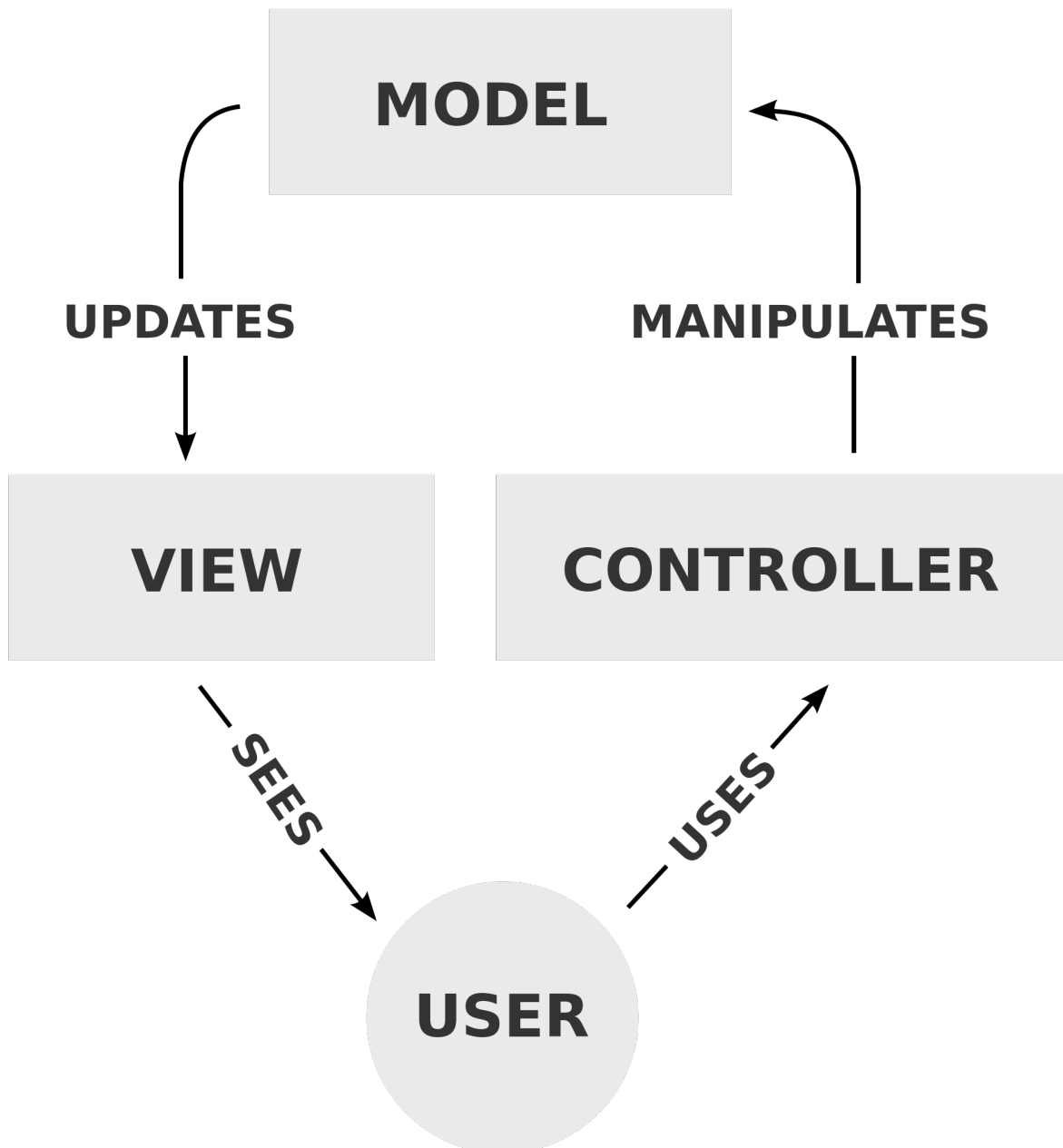
**Rails** Arguably the most popular web-framework out there. Similar to Django in it's features out of the box.

**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.

**Note: Model, View, Controller**



## URL Routing

```
app.route('/delete', delete_account)

def delete_account():
    if username was passed in the query parameters
        and password was passed in the query parameters
        and the user is in the database
        and passed password is correct

        database.remove_user(username)
        return success
    else
        return failure
```

## Templating Engines (mad-libs!)

```
data = {"animal": "Cat", "number": 5}
template.render("You have {{ number }} {{ animal }}s! That's crazy.", data)
```

You have 5 cats! That's crazy.

## TODO: Dynamic Website

Read the documentation on [Flask](#), a simple Python Web-Framework and build a simple “Display the Time in each Timezone” Application.

When a user goes to our website they will see the server's time and when they go to `app-url/<timezone code>` they will see the timezone in that area of the world.( <http://flask.pocoo.org> )

### 3.11.3 Further Reading

## 3.12 Lesson 11: Testing

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

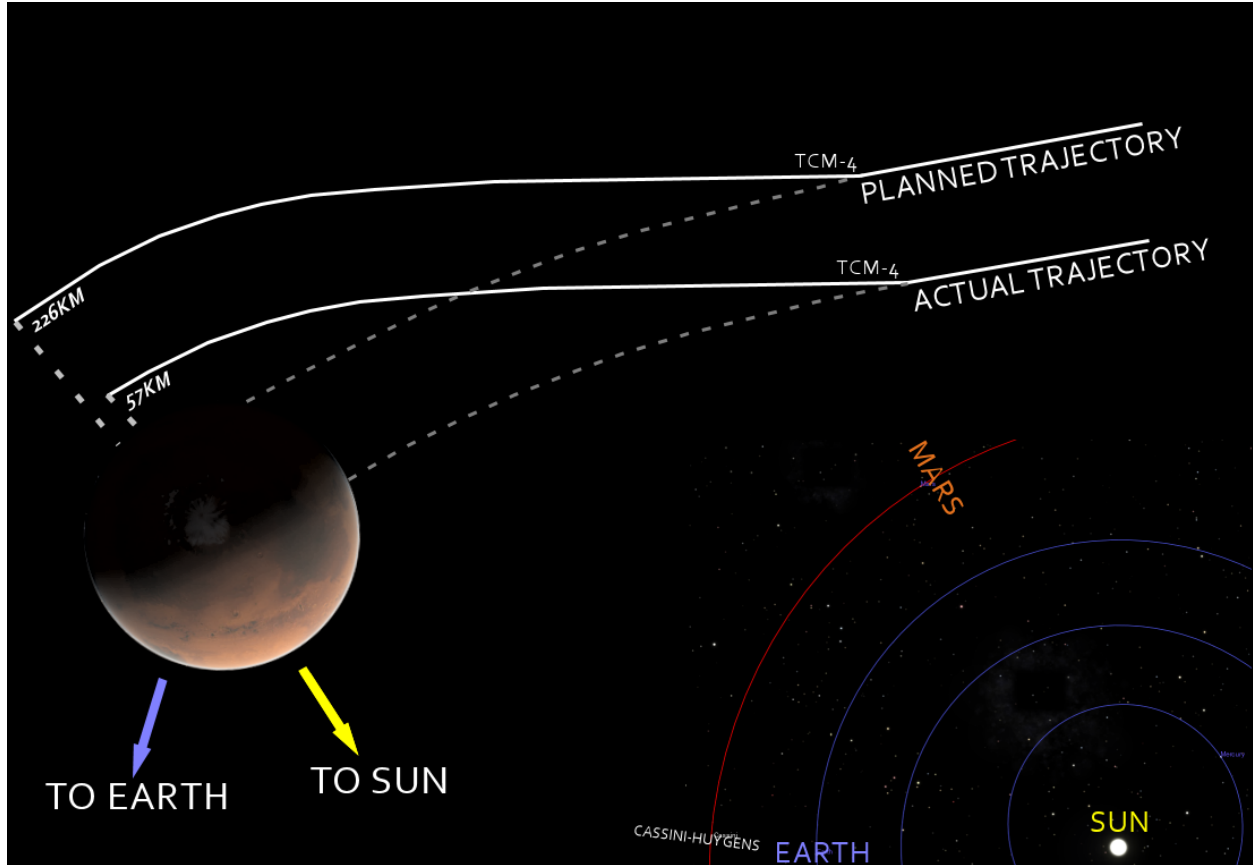
**Warning:** This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

### 3.12.1 Testing

```
def add_double(x, y):
    return 2*(x+y)

def test_add_double():
    expect(add_double(1, 2) == 6)
```

### 3.12.2 Why Testing Matters



### 3.12.3 Structure of a Test

Most test are consist of the same general structure:

### 3.12.4 Types of Testing

### 3.12.5 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.

### 3.12.6 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:

- populate a test database
- write and delete files
- or anything 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 automaically
    use data in database
    asset something is true
    teardown is run automatically
```

### 3.12.7 TODO: Using Python’s unittest

### 3.12.8 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.

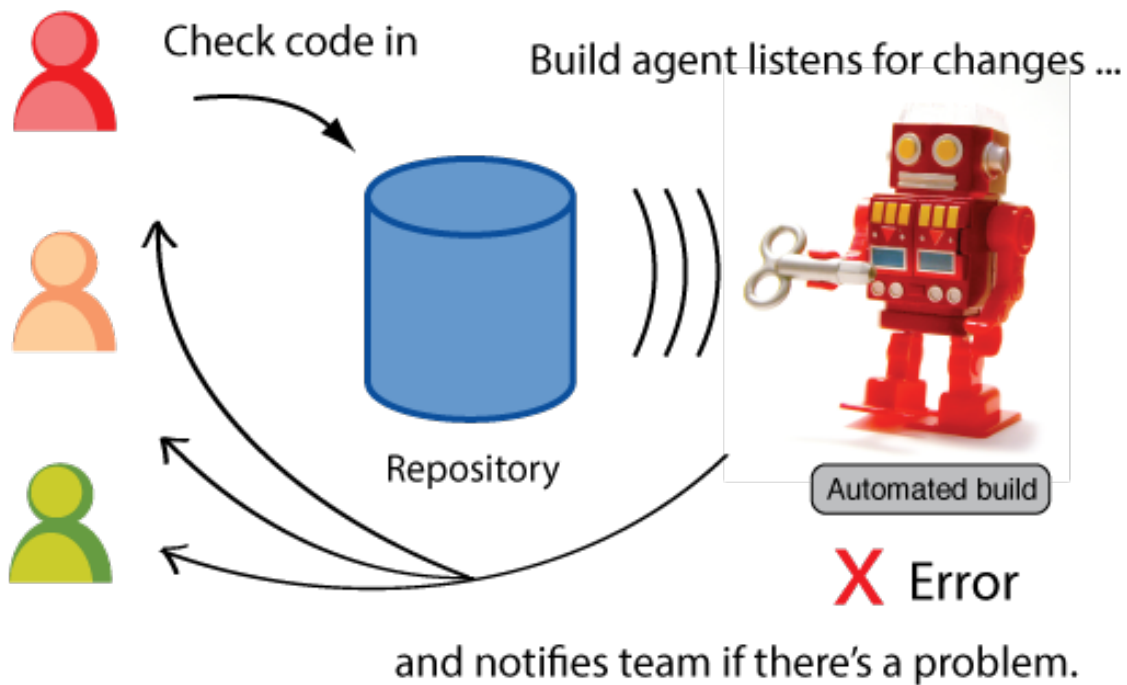
## 3.13 Lesson 12: Continuous Integration

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

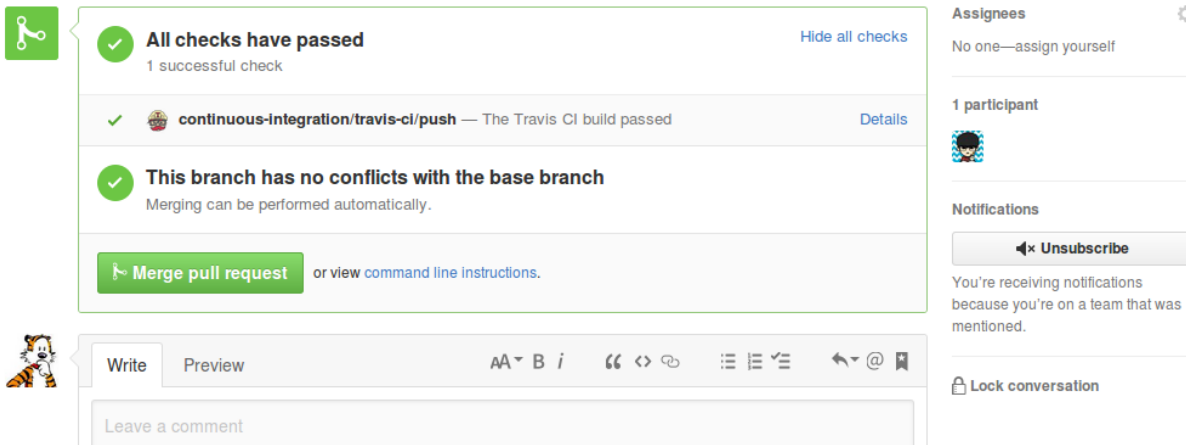
**Warning:** This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

### 3.13.1 Continuous Integration

#### Developers



### 3.13.2 Automated Testing



The screenshot shows a GitHub pull request interface. At the top, a green box indicates "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. Another green box states "This branch has no conflicts with the base branch". A green button labeled "Merge pull request" is visible. On the right, the "Assignees" section shows no one assigned, and the "Notifications" section has an "Unsubscribe" button. At the bottom, a comment box with a "Write" tab and a "Preview" tab is shown, with a "Leave a comment" placeholder.

### 3.13.3 Tool: Travis CI

```

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](#)!*

### 3.13.4 Tool: 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!

### 3.13.5 TODO: Setup Travis on a GH Repo

### 3.13.6 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.

## 3.14 Lesson 13: Security

<a href="#">Homepage</a>	<a href="#">Content</a>	<a href="#">Slides</a>	<a href="#">Video</a>
--------------------------	-------------------------	------------------------	-----------------------

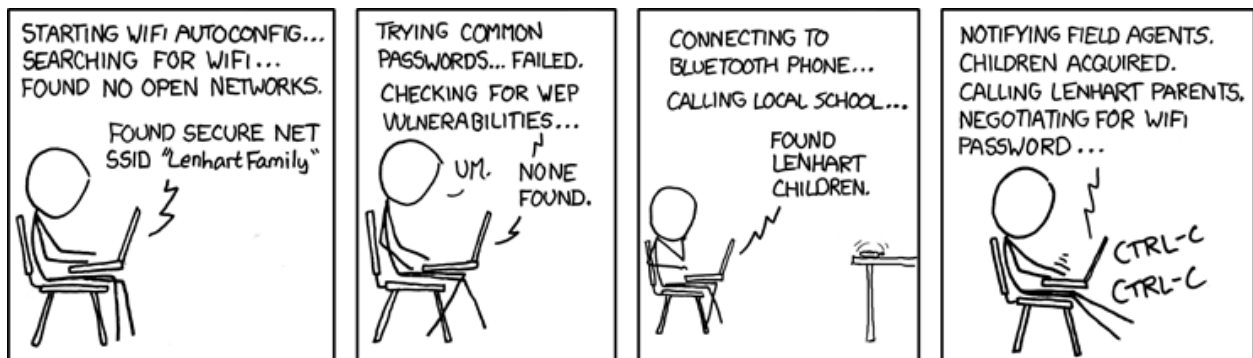
**Warning:** This lesson is under construction. Learn from it at your own risk. If you have any feedback, please fill out our [General Feedback Survey](#).

### 3.14.1 Security

**se·cu·ri·ty** ( *sikyoorit/* ) [ **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.

### 3.14.2 Types of Security



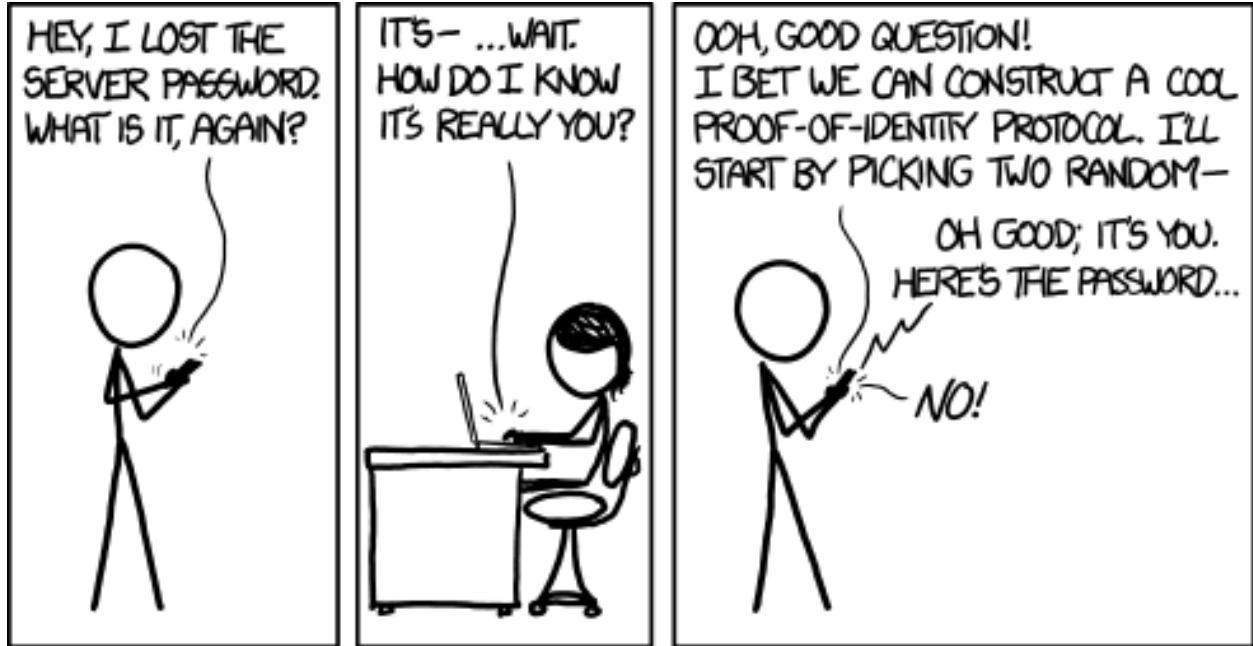
There are three main types of security in computing:

### 3.14.3 Threat Models

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



### 3.14.4 Authentication, Authorization, Identity



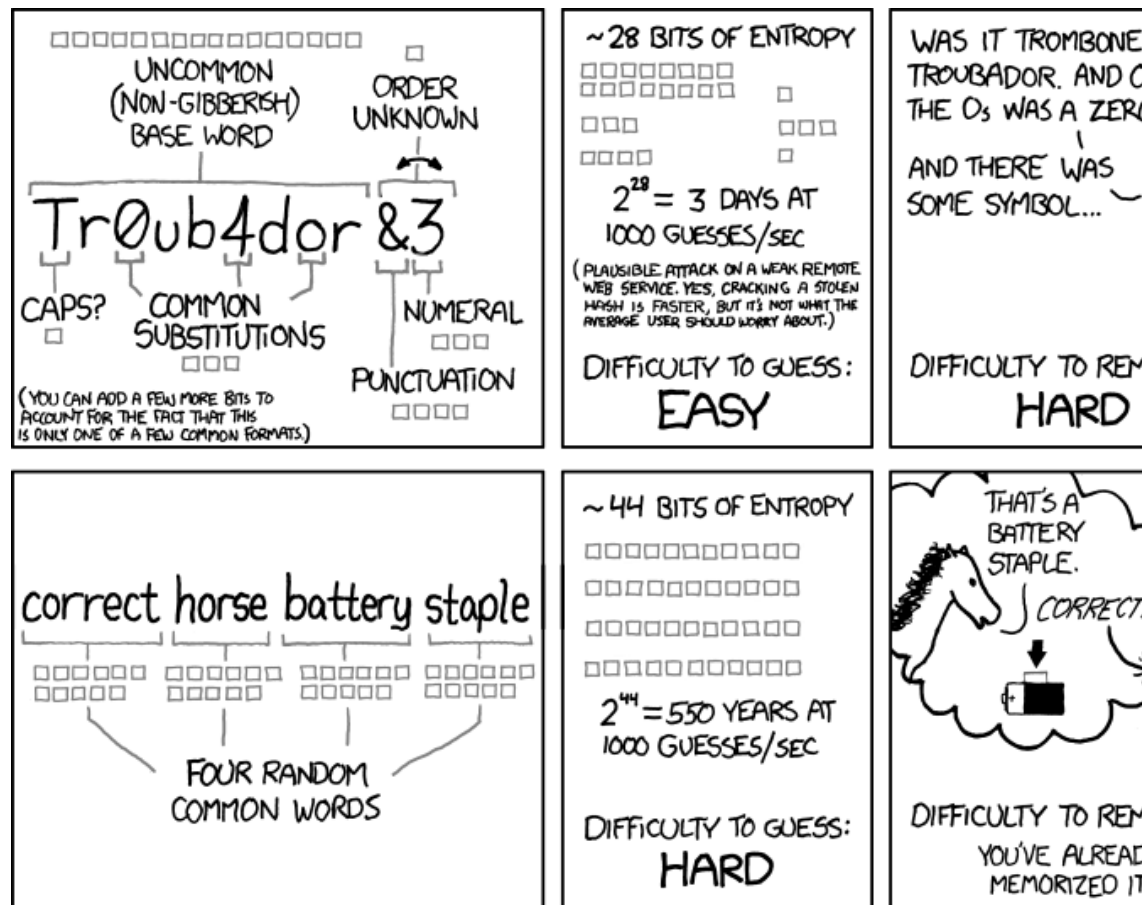
### 3.14.5 Passwords / Passphrases

#### Problems with Passwords

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

## Solutions for Passwords

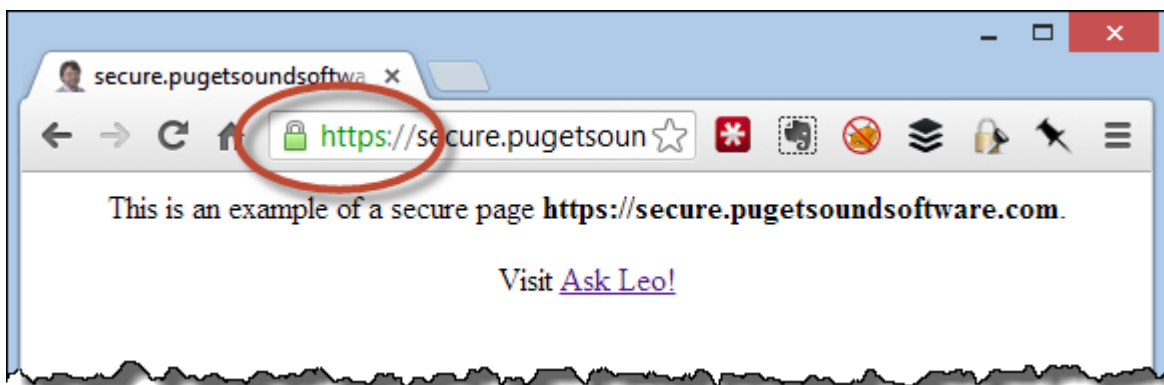
### Choosing Pass-phrases



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

### 3.14.6 Certificates and HTTPS

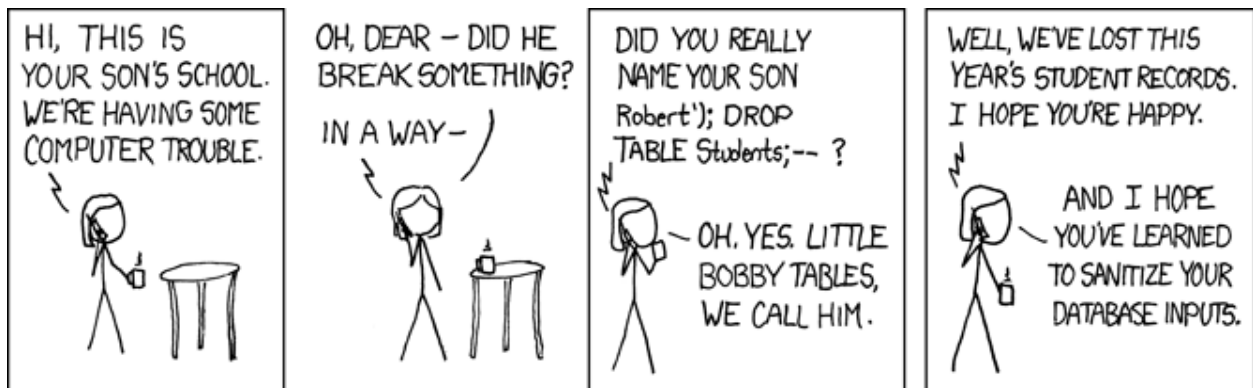


### 3.14.7 Types of Attacks

According the security vendor Cenzic, the top vulnerabilities in March 2012 include:<sup>[9]</sup>

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 Attacks

```
Homepage</a> | <a href="#">Content</a> | <a href="#">Slides</a> | <a href="#">Video</a> |
|--------------------------|-------------------------|------------------------|-----------------------|

**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](#).

### 3.15.1 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

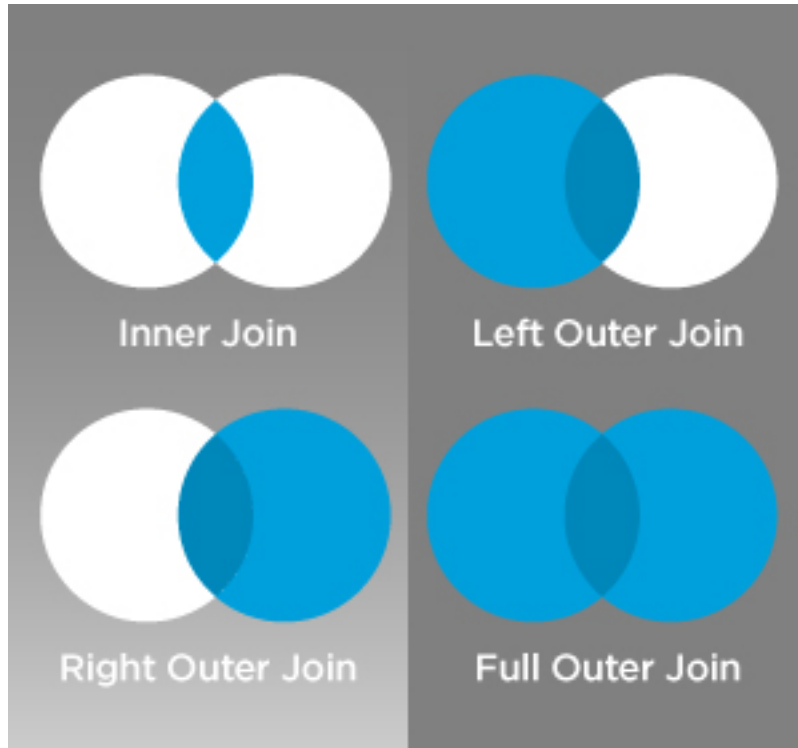
<Table 1>

| <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



### 3.15.2 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

#### Concurrent Read/Writes

### 3.15.3 When *not* to use a Database

### 3.15.4 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

### 3.15.5 Database Concepts

#### Schemas

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

#### Migrations

### 3.15.6 Raw SQL Syntax

#### SELECT

Example:

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

## INSERT

Example:

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

## UPDATE

Example:

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

## DELETE

Example:

```
DELETE FROM
  nobel
WHERE
  yr = 1989 AND subject = 'peace';
```

### 3.15.7 TODO: Crafting Queries!

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

- Who won the prize for Medicine in 1952?
- How many people were awarded the 1903 Nobel in Physics?
- How many prizes 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!=n1.1 or n0.subject!=n1.subject); #(16)
```

### 3.15.8 TODO: Using a *Real* Database

#### Installing MySQL

```
# Install mysql -- hit 'enter' to name your user root, and then enter
# again for password
$ sudo yum install mysql-server

$ sudo service mysqld start # Start the service

$ mysql_secure_installation # Use this to set the root password

# There is no current password
# Hit 'yes' or 'y' for all options
# Add a sensible password which you will remember
# DO NOT MAKE IT YOUR USUAL PASSWORD.

$ sudo service mysqld status # Make sure service is running

$ mysqladmin -u root -p ping # Ping the database

$ mysqladmin -u root -p create nobel # Create a table for Nobel prizes
```

#### Configuration

1. Open and edit /etc/my.cnf.
2. Add default\_storage\_engine = InnoDB to your file.

#### Users

Login to the mysql shell with your root user credentials:

```
$ sudo mysql -p

mysql> CREATE USER 'me'@'localhost'
        IDENTIFIED BY 'password';

mysql> GRANT ALL PRIVILEGES ON nobel.*
        TO 'me'@'localhost'
        WITH GRANT OPTION;

mysql> exit
```

#### Importing Data

```
# Get the database from the osl server
$ wget http://osl.io/nobel -O nobel.sql
# put the database in a file called nobel.sql
```

```
$ sudo mysql -p nobel < nobel.sql
# Open up mysql shell to execute queries
$ sudo mysql -p nobel

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

### 3.15.9 Ways to Use a Database

#### Raw Queries

#### Native Queries

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

db = ("localhost", "testuser", "test123", "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)
```

### 3.15.10 TODO: Use an ORM

### 3.15.11 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!

## 3.16 Lesson 15: Dev Processes & Tools

|                          |                         |                        |                       |
|--------------------------|-------------------------|------------------------|-----------------------|
| <a href="#">Homepage</a> | <a href="#">Content</a> | <a href="#">Slides</a> | <a href="#">Video</a> |
|--------------------------|-------------------------|------------------------|-----------------------|

**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](#).

### 3.16.1 Code Analysis

(How to( find( the missing parenthesis)).

**Static** These tools look at your code *files* and never actually run the program.

Includes Linting and Type Checking.

**Dynamic** Dynamic Code Analysis tools actually run your code and make verifications by watching how your program acts, where it fails, what it does in memory, and if your tests are adequate enough.

Includes Debuggers, Memory Profiling Tools, and Code Coverage.

### 3.16.2 Debugging Tools

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

#### CLI Tools

##### C/C++

- GDB: Used to inspect and debug everything in a C program from variables to why it encountered a fatal failure. Generally called the swiss-army-knife of debugging, a great tool to use in programming, allows you to set '*break points*' where the program stops mid-run and you can see what it's doing.
- Valgrind: Used specifically to inspect and debug memory issues.

```
$ valgrind ./tests/bin/lexer_tests
...
==6703== Conditional jump or move depends on uninitialised value(s)
...
==6703==    by 0x4018C1: print_token (lexer.c:36)
==6703==    by 0x4011F7: test_get_tok (lexer_tests.c:54)
==6703==    by 0x4008CD: main (lexer_tests.c:8)
...
==6703== LEAK SUMMARY:
==6703==    definitely lost: 192 bytes in 8 blocks
==6703==    indirectly lost: 162 bytes in 10 blocks
```

##### Python

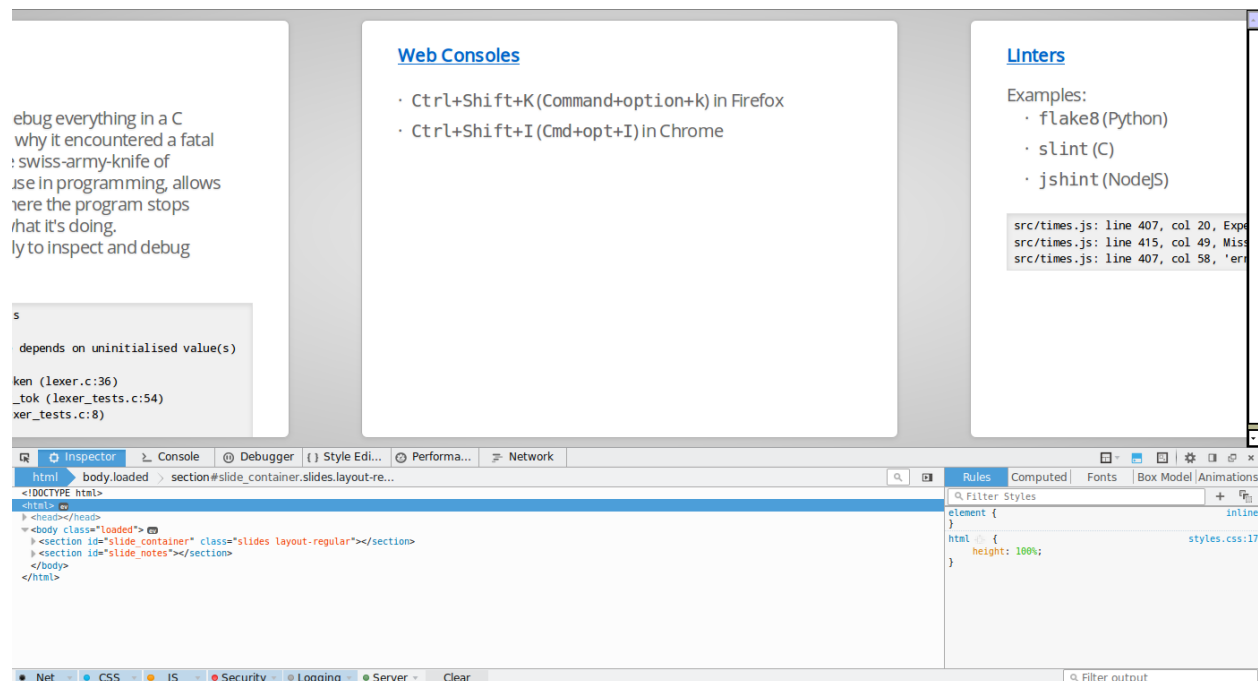
- PDB: The GDB tool for python.
- Stack Traces: A feature built into Python. Explains what function calls triggered a fatal failure. Not interactive but very useful for debugging in development.

```
$ python some_script.py
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in f
TypeError: unsupported operand type(s) for * : 'int' and 'NoneType'
```

## NodeJS

- node debug: Built into NodeJS used to debug programs in the language.
- Node Inspector: External tool for debugging NodeJS programs.

## Web Consoles



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

## Linters

### Examples:

- flake8 (Python)
- slint (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.
```

**TODO: Lint the Code!**

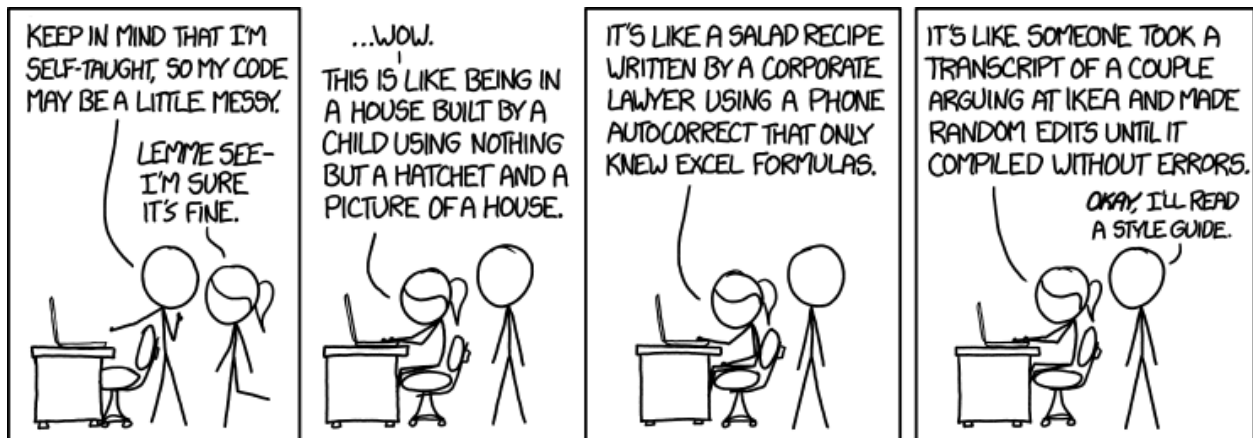
### 3.16.3 Code Coverage

### 3.16.4 Integrated Development Environments (IDE)

**Examples:**

- Netbeans: Java
- Visual Studio: .NET
- PyCharm: Python
- Eclipse: Misc

### 3.16.5 Style Guides



#### Example: Linux Kernel Standards

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.

[ <https://www.kernel.org/doc/Documentation/CodingStyle> ]

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.

[ [http://lars-lab.jpl.nasa.gov/JPL\\_Coding\\_Standard\\_C.pdf](http://lars-lab.jpl.nasa.gov/JPL_Coding_Standard_C.pdf) ]

## Style Guides Enforced by Linters

### 3.16.6 Dependency Isolation

#### Python: Virtualenv

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==6.6
Flask==0.11.1
itsdangerous==0.24
Jinja2==2.8
MarkupSafe==0.23
Werkzeug==0.11.11
```

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.

### 3.16.7 Development Servers

*A Carbon Copy of the Production Environment(s)*

### 3.16.8 TODO

### 3.16.9 Further Reading

- The [Community Ruby Style Guide](#) is a good resource for anybody learning Ruby. It's the style guide that `Rubocop` enforces.

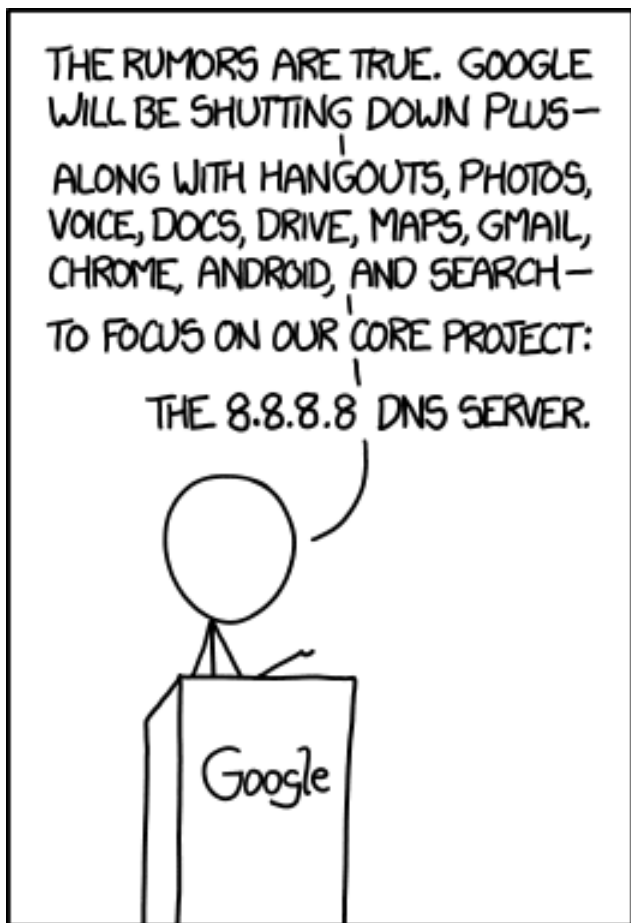
- The [Official Python Style Guide](#) is a well respected style guide for Python and is commonly accepted as *the* python style guide.

## 3.17 Lesson 16: DNS

|                          |                         |                        |                       |
|--------------------------|-------------------------|------------------------|-----------------------|
| <a href="#">Homepage</a> | <a href="#">Content</a> | <a href="#">Slides</a> | <a href="#">Video</a> |
|--------------------------|-------------------------|------------------------|-----------------------|

**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](#).

### 3.17.1 Problems DNS Solves



The Domain Name System (DNS) translates human-readable URLs ([devopsbootcamp.osuosl.org](http://devopsbootcamp.osuosl.org)) into computer IP addresses (140.211.15.183).

It works by storing records in a distributed tree-like hierarchy. It was designed like this because it scales well.

### Obligatory History Lesson

|      |   |
|------|---|
| MIT  | 1 |
| Yale | 2 |

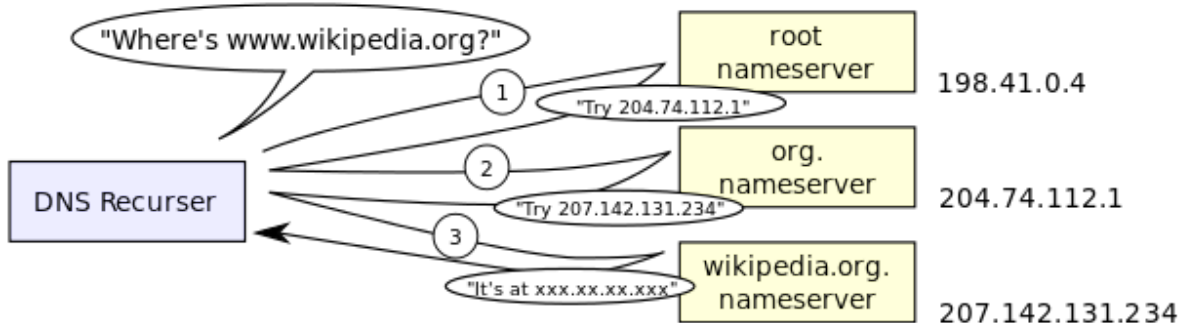
```
Harvard      3
ATT          4
...
joeBillson   14895
susan-gill   15832
```

### 3.17.2 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.

### 3.17.3 A DNS Request

1. A computer makes a request for `http://osuosl.org..`
2. This request gets sent to the root (`.`) of the DNS tree.
3. The root sends it off to the `org` (top level domain) branch.
4. The `org` node sends it off to the `osuosl` (domain) branch.
5. The `osuosl` node sends it to the `devopsbootcamp` (subdomain) branch.





### 3.17.4 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

```
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

```
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

- A DNS server is authoritative if it has a Start of Authority (SOA) record for a domain
- The root-servers contain SOA records for the TLDs and gTLDs
- The NS servers for each (g)TLD contain SOA records for each registered domain
- ... and so on...

#### CNAME Records

#### NXDOMAIN Records

Tells you there is no answer to a query:

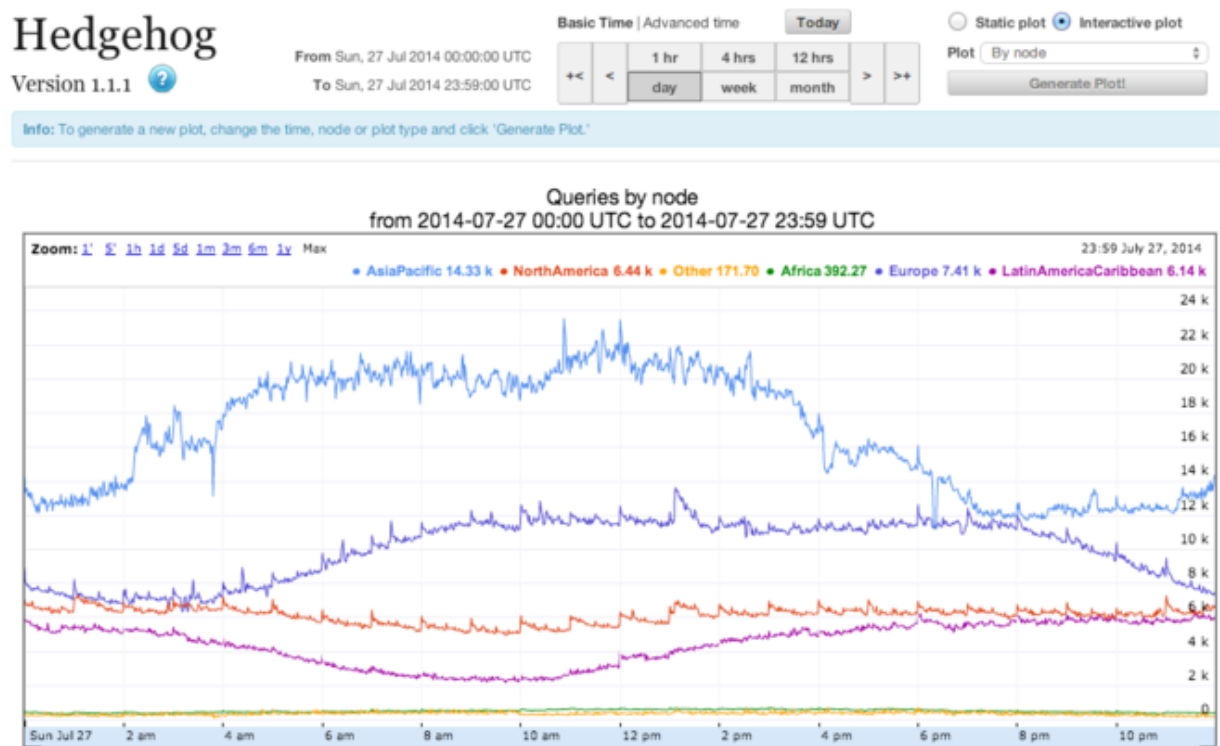
```
Host something.invalid.osuosl.org not found: 3 (NXDOMAIN)
```

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

### 3.17.5 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



### 3.17.6 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.
.          518400  IN  NS  d.root-servers.net.
.          518400  IN  NS  k.root-servers.net.
.          518400  IN  NS  g.root-servers.net.
.          518400  IN  NS  h.root-servers.net.
.          518400  IN  NS  m.root-servers.net.
.          518400  IN  NS  e.root-servers.net.
.          518400  IN  NS  c.root-servers.net.
.          518400  IN  NS  j.root-servers.net.
```

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.afiliias-nst.info.
org.                  172800 IN  NS  a2.org.afiliias-nst.info.
org.                  172800 IN  NS  b0.org.afiliias-nst.org.
org.                  172800 IN  NS  b2.org.afiliias-nst.org.
org.                  172800 IN  NS  c0.org.afiliias-nst.info.
org.                  172800 IN  NS  d0.org.afiliias-nst.org.

;; ADDITIONAL SECTION:
a0.org.afiliias-nst.info. 172800 IN  A   199.19.56.1
a2.org.afiliias-nst.info. 172800 IN  A   199.249.112.1
b0.org.afiliias-nst.org.  172800 IN  A   199.19.54.1
b2.org.afiliias-nst.org.  172800 IN  A   199.249.120.1
<truncated>
```

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
```

### 3.17.7 TODO: Traverse the DNS Tree with `dig`

### 3.17.8 TODO: Run a DNS Server

### 3.17.9 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.

## 3.18 Lesson 17: Configuration Management

|                          |                         |                        |                       |
|--------------------------|-------------------------|------------------------|-----------------------|
| <a href="#">Homepage</a> | <a href="#">Content</a> | <a href="#">Slides</a> | <a href="#">Video</a> |
|--------------------------|-------------------------|------------------------|-----------------------|

**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](#).

### 3.18.1 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

### 3.18.2 Concept: Infrastructure as Code

Infrastructure as code is the act of describing what you want your servers to look like *once*, and using that to *provision* many machines to look the same.

It turns *pets* into *cattle*. (more on this difference later)

- CM enables System Operation to define their infrastructure in 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.

### 3.18.3 Pull vs Push Models

#### **Pull Model**

- When the server being provisioned (node) runs an agent (daemon) that asks a central authority (master) if/when it has any updates that it should run.
- Requires a daemon to be installed on all machines *and* a central authority to be setup.
- Scales well, difficult to manage.

#### **Push Model**

- A central server contacts the nodes and sends updates as they are needed.
- When a change is made to the infrastructure (code) each node is alerted of this and they run the changes.
- Simple to manage and setup (usually uses prevalent SSH protocol), not scalable.

### 3.18.4 Tools

#### Puppet



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

[ [Puppet Site](#) ]

## Chef



# CHEF™

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

[ [Chef Site](#) ]

## CFEngine



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

[ [CFEngine Site](#) ]

## Ansible



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



[ [Ansible Site](#) ]

## SaltStack

- Easy to use.
- Hard to get started.

[ [SaltStack Site](#) ]

### 3.18.5 Delcaration 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 it's own language for Configuration management you are more limited in what you can express with Puppet, but this isn't always a bad thing. It's feature rich and can do pretty much anything 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.

---

### 3.18.6 TODO: Use Ansible to Provision localhost

```
$ pip install ansible
```

### 3.18.7 Further Reading

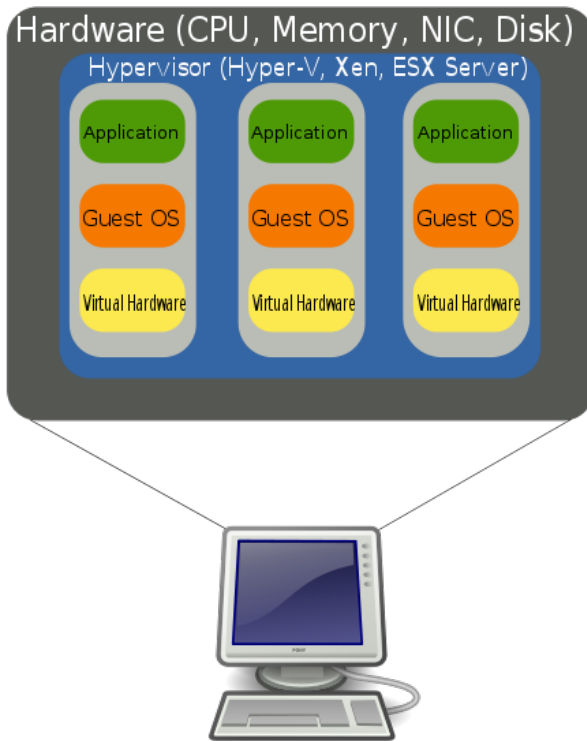
## 3.19 Lesson 18: Application Isolation

|                          |                         |                        |                       |
|--------------------------|-------------------------|------------------------|-----------------------|
| <a href="#">Homepage</a> | <a href="#">Content</a> | <a href="#">Slides</a> | <a href="#">Video</a> |
|--------------------------|-------------------------|------------------------|-----------------------|

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

### 3.19.1 Application Isolation

### 3.19.2 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

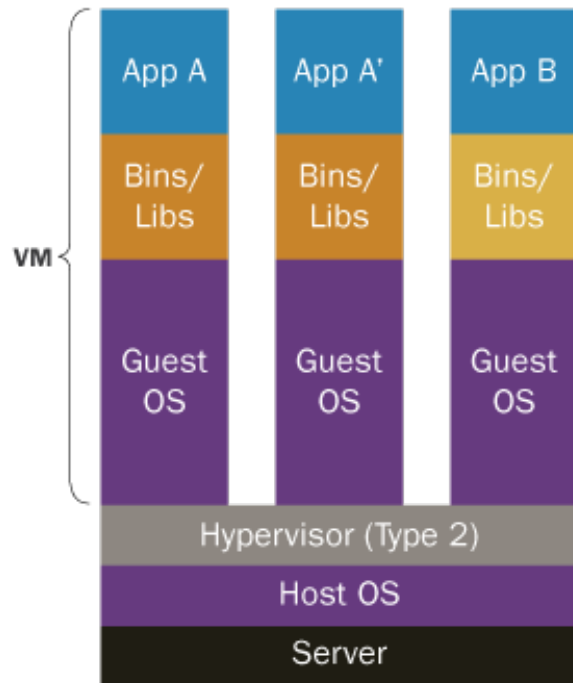
### 3.19.3 Containers

```
$ ps aux # Lists all processes running on an OS
PID  USER      TIME  COMMAND
```

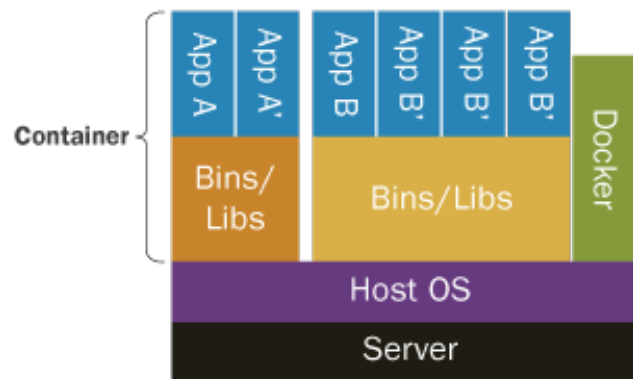
```
1 root      0:00 sh
6 root      0:00 ps aux
```

## Not a Virtual Machine

# Containers vs. VMs



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



## CGroups + Systemd

### 3.19.4 Pros

| Virtual Machines                              | Containers                      |
|-----------------------------------------------|---------------------------------|
| Complete process isolation<br>'Battle Tested' | Fast startup<br>Little overhead |

### 3.19.5 Cons

| Virtual Machines                                                | Containers                                       |
|-----------------------------------------------------------------|--------------------------------------------------|
| Slightly more overhead.<br>Slow startup.<br>Cross-OS emulation. | Security concerns.<br>No cross-kernel emulation. |

### 3.19.6 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 defacto 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 isolation.

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

### 3.19.7 TODO

### 3.19.8 Further Reading

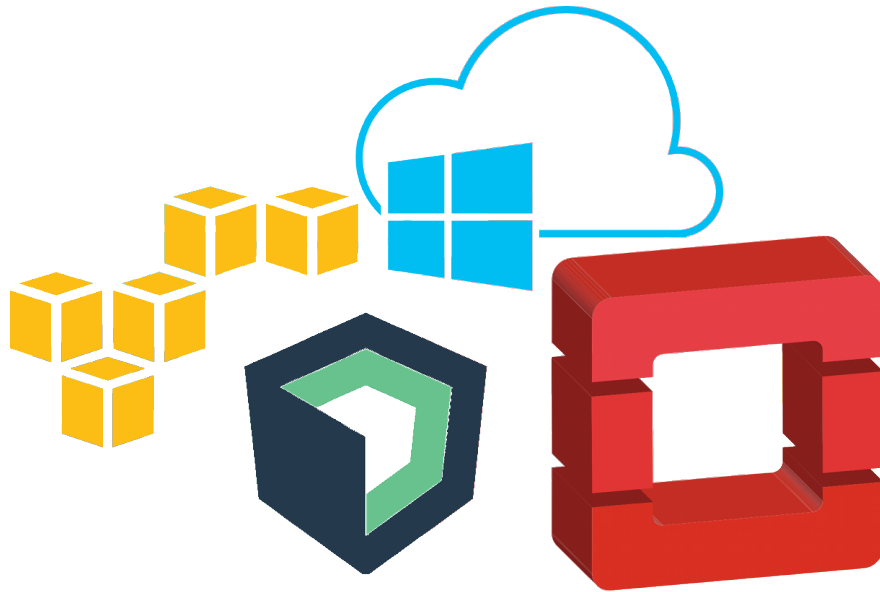
## 3.20 Lesson 19: Cloud Infrastructure

|                          |                         |                        |                       |
|--------------------------|-------------------------|------------------------|-----------------------|
| <a href="#">Homepage</a> | <a href="#">Content</a> | <a href="#">Slides</a> | <a href="#">Video</a> |
|--------------------------|-------------------------|------------------------|-----------------------|

**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](#).

### 3.20.1 What the Cloud Looks Like

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



### 3.20.2 Advantages over Bare Hardware

- **Ephemeral:** Creating and Destroying operating systems is quick and painless.
- **Cost effective:** Pay for what you use.
- **Low startup cost:** Initial investment is cheap, <\$100 as opposed to >\$1,000+. (unless you are running a private cloud, more on that in a second).

### 3.20.3 Private Clouds

### 3.20.4 Public Clouds

### 3.20.5 Cloud + Configuration Management

### 3.20.6 Cattle VS Pets

Advantages

Disadvantages

### 3.20.7 TODO

### 3.20.8 Further Reading

## 3.21 Lesson 20: Contributing to Open Source

|                          |                         |                        |                       |
|--------------------------|-------------------------|------------------------|-----------------------|
| <a href="#">Homepage</a> | <a href="#">Content</a> | <a href="#">Slides</a> | <a href="#">Video</a> |
|--------------------------|-------------------------|------------------------|-----------------------|

**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](#).

### 3.21.1 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.

[gnu.org/philosophy/free-sw](https://gnu.org/philosophy/free-sw)

### 3.21.2 Accessing 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.

### 3.21.3 How to Get Involved

### 3.21.4 Finding a Project

In order of perceived usefulness:

- [Openhatch](#)
- [24 pull requests](#)
- [BugsAhoy](#)
- [Showcased github projects](#)
- [Trending github projects](#)
- Choose a company, search "<Company Name> Open Source"
- Easy bugs
- GSOC submitters who didn't get enough interns
- Search by language
- Search by project type – find something that interests you (web dev? bioinformatics? video games?)
- Your immediate payment for contributions will be satisfaction, so pick something satisfying

### I Can't Find a Project I Like!

*That's okay.*

Sometimes you find the project, sometimes the project finds you.

### 3.21.5 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

#### 3.21.6 Know your Licenses



- **MIT:** A very lax license permitting any (free and non-free) use of the software.
- **Apache:** A little more precise, gives more rights to the developers.
- **AGPL/GPL/LGPL:** For when you love Open Source and want to spread the love.
- **Creative Commons:** For when you're not writing code.
- <http://choosealicense.com/>

### 3.21.7 TODO: Find a FOSS Project

### 3.21.8 Further Reading

## 3.22 About

### 3.22.1 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.

### 3.22.2 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.

### 3.22.3 Policies

#### Attendance

Attendance is not mandatory but highly suggested to get the most out of DBOC; 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.

### 3.22.4 Get Involved

#### Mailing list

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

#### 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](mailto: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.

## 3.23 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.

### 3.23.1 Fall

| Lesson         | Date/Time       | Location     | Description                         |
|----------------|-----------------|--------------|-------------------------------------|
| DevOps Daycamp | Oct 1, 10am-3pm | OSU KEC 1001 | DevOps DayCamp (DOBC Kickoff)       |
| Fall Meeting 2 | Nov 5, 11am-3pm | OSU KEC 1001 | Files, Version Control, Programming |
| Fall Meeting 3 | Dec 3, 11am-3pm | OSU KEC 1001 | Frameworks, Testing and CI          |

### 3.23.2 Winter

| Lesson | Date/Time | Location | Description |
|--------|-----------|----------|-------------|
|        |           |          |             |

### 3.23.3 Spring

| Lesson | Date/Time | Location | Description |
|--------|-----------|----------|-------------|
|        |           |          |             |

## 3.24 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.

### 3.24.1 Before You Begin

### 3.24.2 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.

### 3.24.3 Online Engagement

### 3.24.4 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.