

---

# Table of Contents

Introduction	1.1
Unit 1: Dev Environment	1.2
Install Ubuntu	1.2.1
Linux Basics	1.2.2
System Basics	1.2.3
LAMP Stack	1.2.4
Apache Basics	1.2.5
NPM	1.2.6
Git	1.2.7
Unit 2: BashScripting	1.3
Bash Scripting Basics	1.3.1
Unit 3: How Servers and the Web Work	1.4
The Anatomy of a URL	1.4.1
User Requests and Server Responses	1.4.2
Unit 4: HTML	1.5
HTML	1.5.1
HTML Forms With PHP	1.5.2
MailGun API	1.5.3
Unit 5: CSS	1.6
HTML	1.6.1
SASS	1.6.2
Unit 6: PHP	1.7
PHP Basics	1.7.1
PHP Control Structures	1.7.2
Form Validation	1.7.3
Mailgun API	1.7.4
Heredoc	1.7.5
Unit 7: JavaScript	1.8
JavaScript Basics	1.8.1
JavaScript Control Structures	1.8.2
Walking The Dom	1.8.3
Events	1.8.4
Canvas	1.8.5
jQuery	1.8.6
Unit 8: Front End Toolkits	1.9
Toolkits	1.9.1
Bootstrap	1.9.2

---

Yarn and Gulp	1.9.3
Unit 9: MySQL	1.10
SQL	1.10.1
MySQL	1.10.2
Working with MySQL	1.10.3
Data Models	1.10.4
Unit 10: CakePHP	1.11
CakePHP	1.11.1
Unit 11: MongoDB	1.12
MongoDB	1.12.1
Unit 12: Express	1.13
Unit 13: Angular	1.14
Unit 14: Apache Cordova	1.15
14-ApacheCordova	1.15.1
Course Syllabus	1.16

---

# MicroTrain's Dev Bootcamp

Resources for MicroTrain's Developer Bootcamp.

# Dev Environment

In this section you will create a dev environment by

- Installing Ubuntu Linux
- Installing a full LAMP Stack
- Some basic configuration
- Install basic development tools

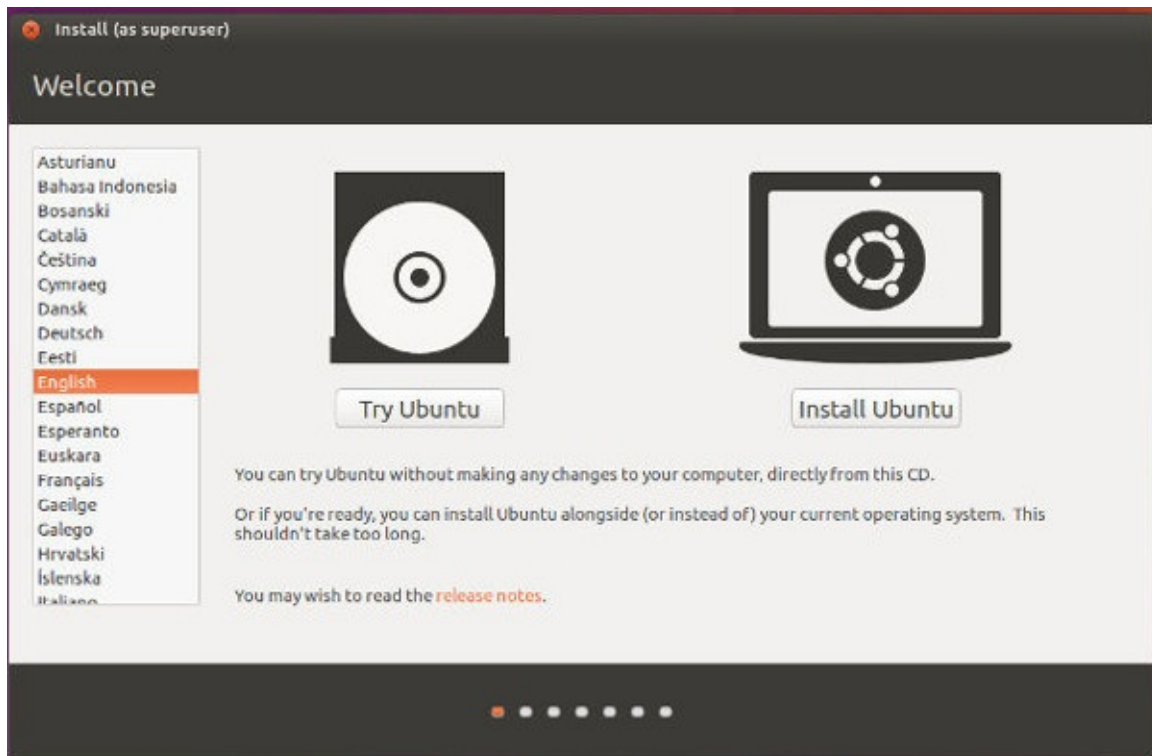
The goal of this section is to gain an understanding of the dev environemnt and gain some comfort on a Linux commandline.

## Install Ubuntu 16.04

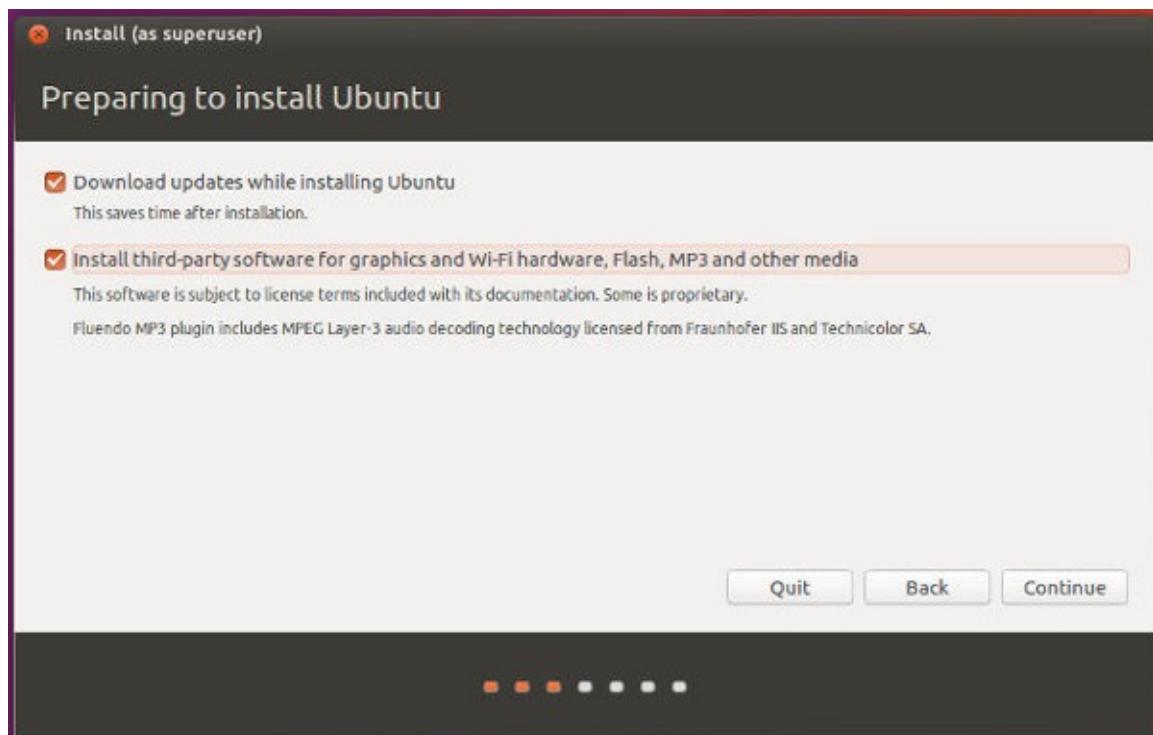
Your development (dev) environment is any system upon which you develop your applications. When possible, building your dev environment as close to production (prod) as possible is helpful. When this is not possible running tests in an emulator is a good option. This course will make use of both methods. Let's start by installing the operating system; Ubuntu Linux.

Insert the preloaded Ubuntu 16.04 dongle into an open USB port and power up the system. Press f12 when the Dell splash screen appears and choose *USB Storage Device* under legacy boot legacy.

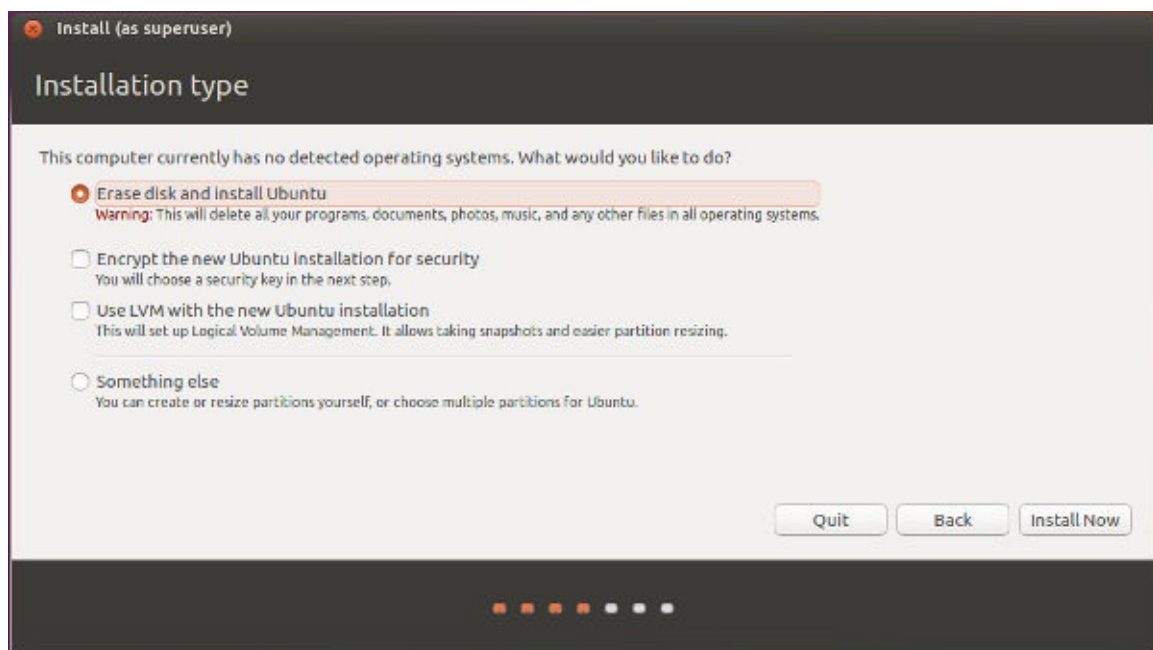
You will see a window titled *Install (as superuser)* from this window choose English and click the *Install Ubuntu* button.



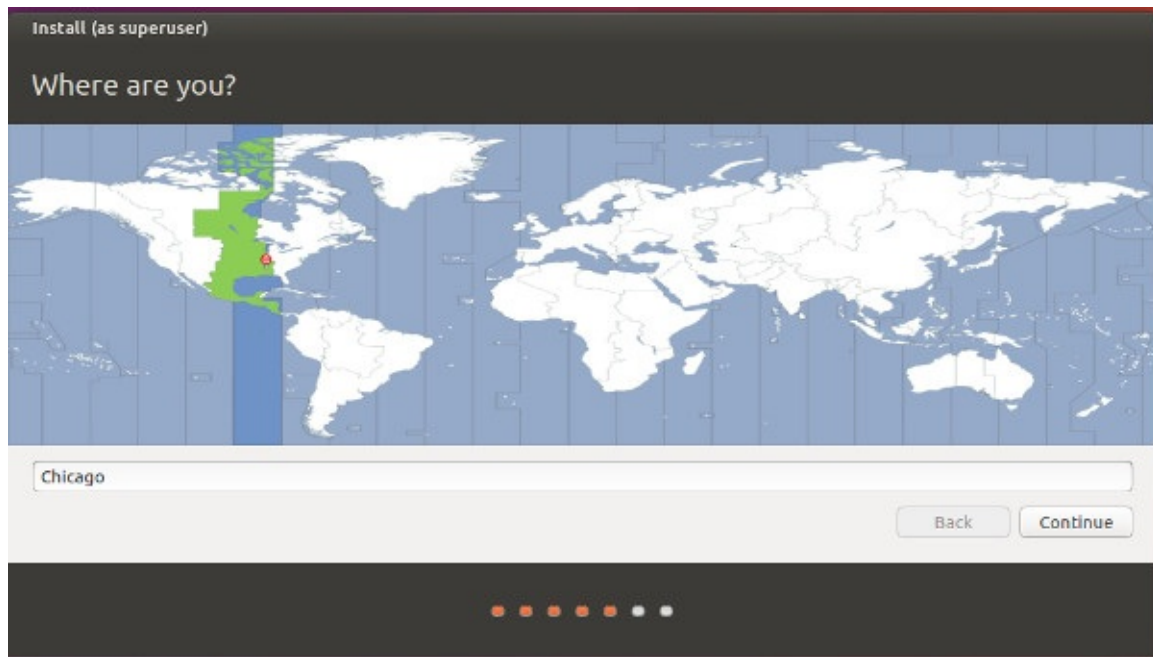
You will now be given two options *Download updates while installing Ubuntu* and *Install third-party software...* select both of these options.



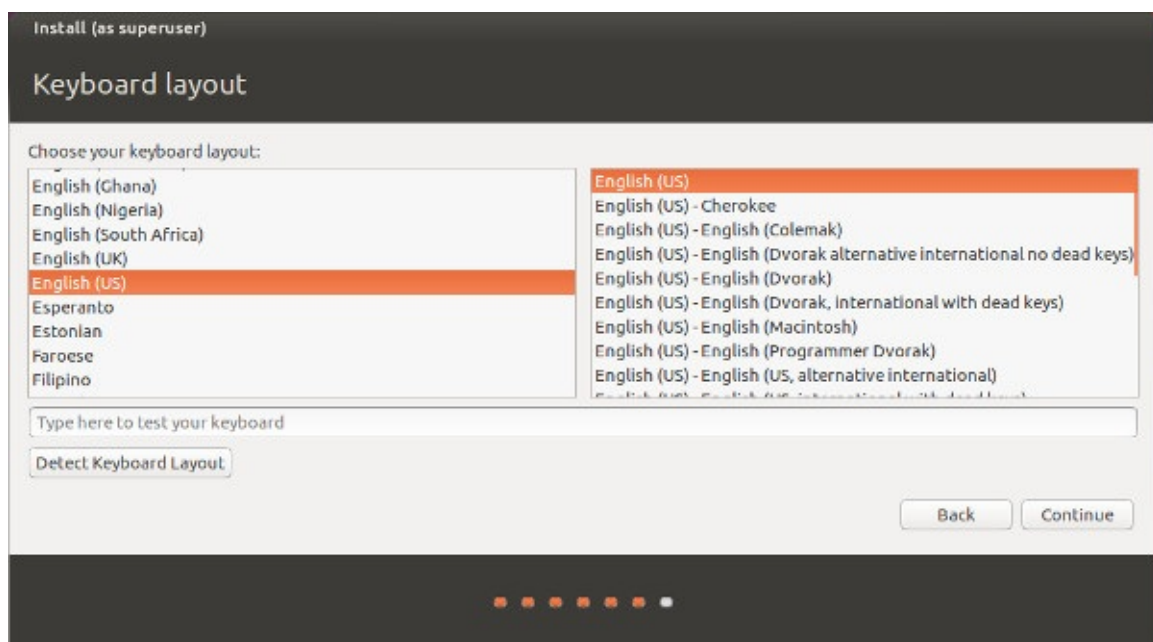
Now it's time to install the system. To keep things simple we will do a simple install. Be sure the *Erase disk and install Ubuntu* option is selected and click the *Install Now* button. You may see a dialog that asks *Write changes to disk?* with some additional details, simply click the *continue* button.



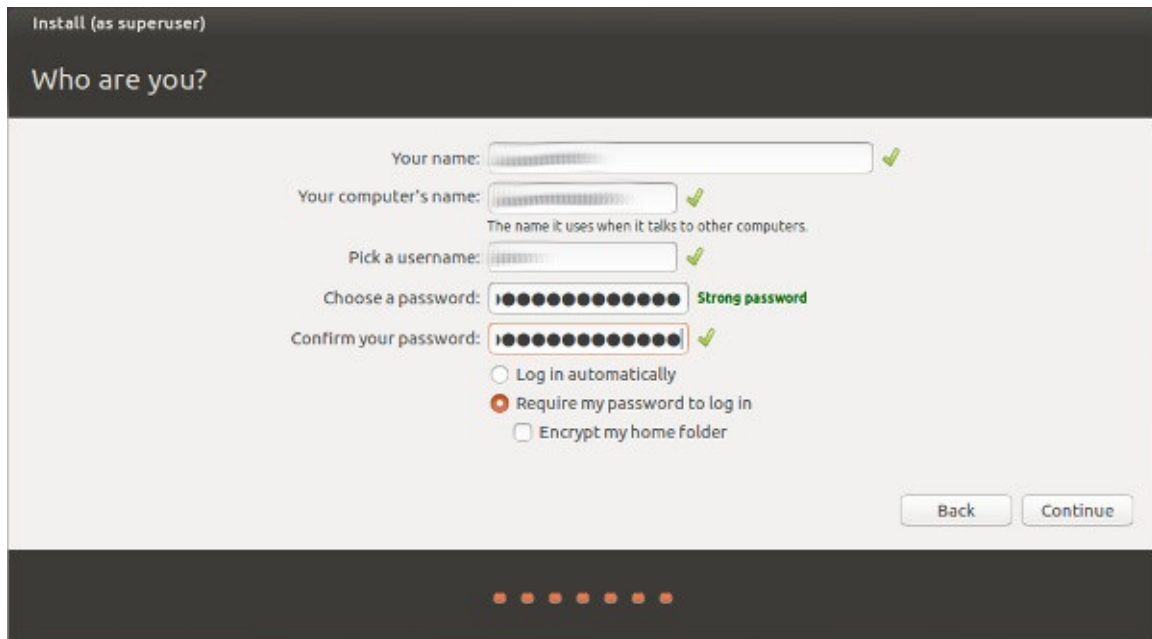
You will now see the *Where are you?* screen. This should default to Chicago (or your default location), if not choose Chicago (or your default location) as your location and click the *Continue*.



This sets your systems local timezone. For *Keyboard layout* choose English (US) and English (US) and click the continue button.



On the *Who are you?* screen enter *Your name*: as your first and last name, accept the default computer and usernames and choose a password. Do not forget your password, we are not able to recover this for you. Be sure the *Require my password to log in* option is selected and that the *Encrypt my home folder* option **IS NOT** selected. Click the continue button and wait for the system to install.

The image shows the 'Who are you?' configuration screen in the Ubuntu installer. It has a dark header with 'Install (as superuser)' and 'Who are you?'. The main area is light gray and contains several input fields: 'Your name:', 'Your computer's name:', 'Pick a username:', 'Choose a password:', and 'Confirm your password:'. Each field has a green checkmark to its right. Below the password fields are three radio buttons: 'Log in automatically' (unselected), 'Require my password to log in' (selected), and 'Encrypt my home folder' (unselected). At the bottom right are 'Back' and 'Continue' buttons. A progress bar at the very bottom shows six dots, with the first one filled.

Install (as superuser)

## Who are you?

Your name:  ✓

Your computer's name:  ✓  
The name it uses when it talks to other computers.

Pick a username:  ✓

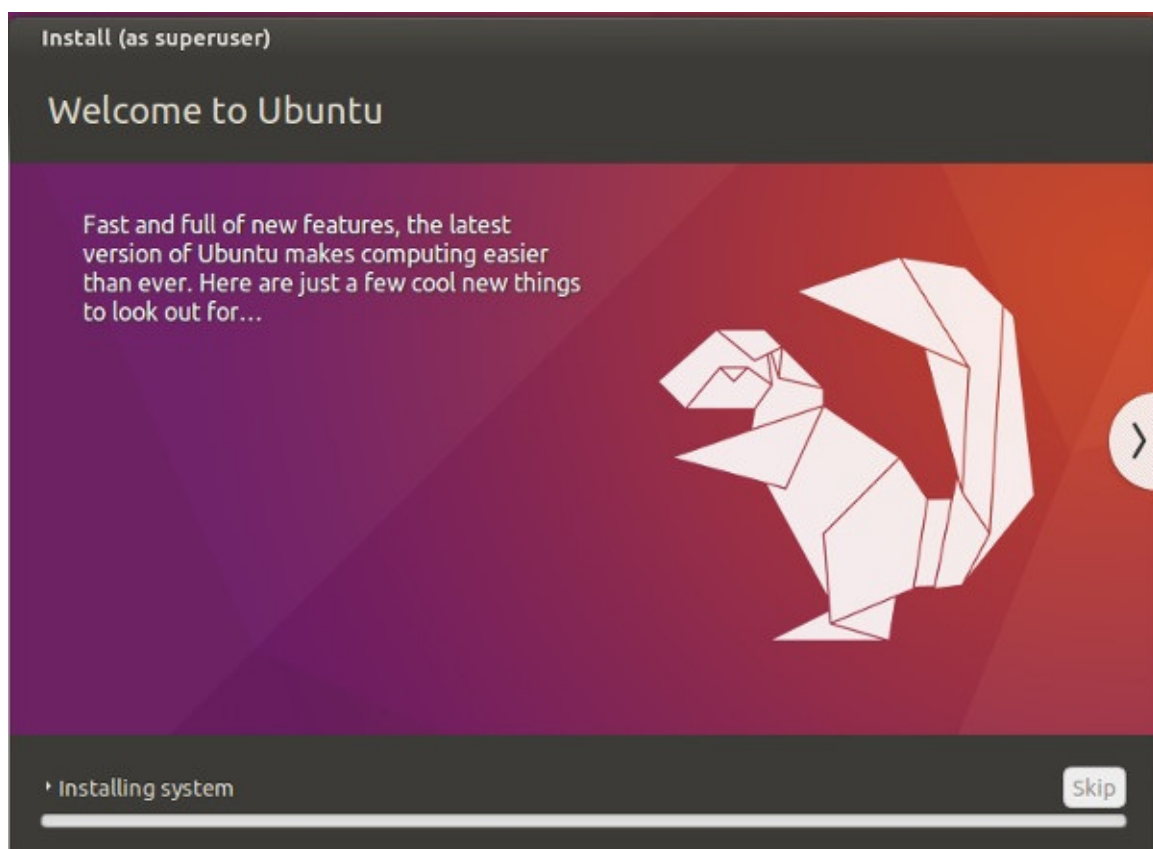
Choose a password:  Strong password

Confirm your password:  ✓

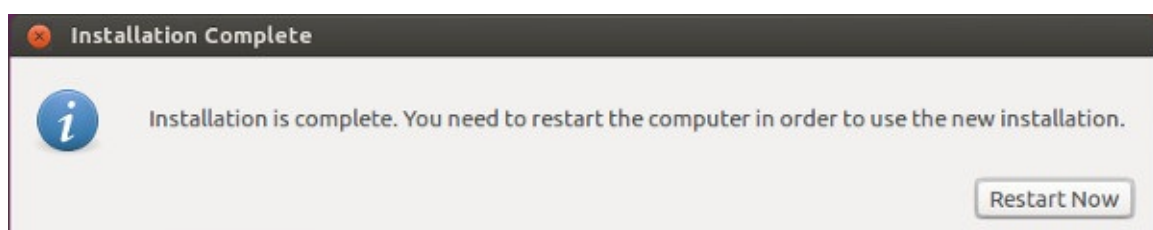
☐ Log in automatically  
☒ Require my password to log in  
☐ Encrypt my home folder

Back Continue

While the system installs, you will see series of flash screens telling you the features of Ubuntu.



One installation has completed you will be prompted to restart your system. Click the *Restart Now* button.





You should see a prompt that says *Please remove the installation medium, then press Enter:.* Pull the dongle from the USB drive and press the enter key.

[Next: Linux Basics](#)

## Summary

In this lesson you learned

- how to install Ubuntu Linux (Ubuntu, Linux)

# Linux Basics

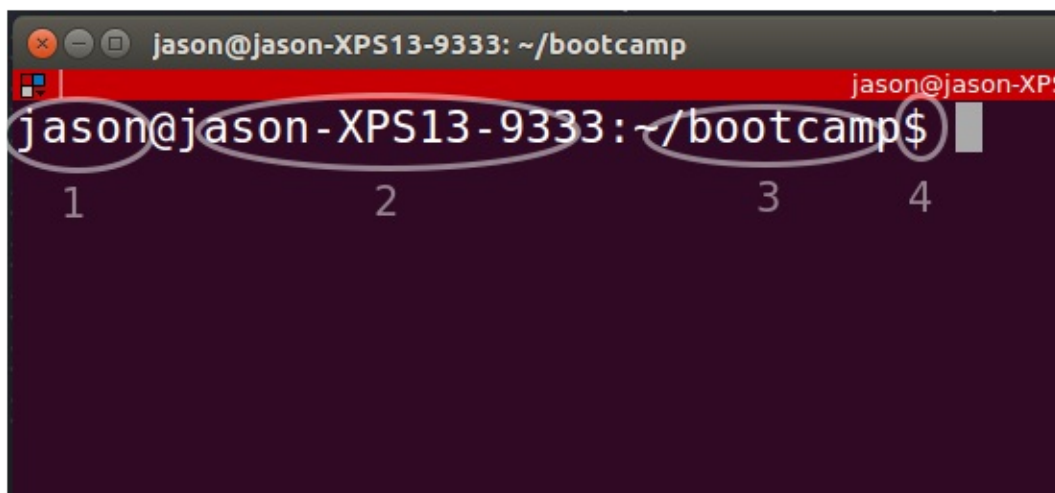
This section is intended to provide a reference for the basic commands needed to navigate a Linux system from the command line. We will take a deeper dive and use these commands in later lessons. While we call these commands, they are really programs. In other words, when you type a command in Linux, you are really invoking a program. Most programs can be invoked with out any additional arguments (aka parameters), but in most cases, you will want to pass additional arguments into the program.

For example `vim filename.txt` would use an editor called vim to open a file that is named filename.txt. Some programs expect arguments to be passed in a specific order while others have predefined arguments. All Linux programs predefine `--help` so you might type `chown --help` to learn how to use the chown command (or program).

## The Linux File System

Linux does not use drive letters as you may be used to if you come from a Windows environment, rather everything is mounted to a root name space. `/`.

- `/` - root
- `/etc` - sytem configuration
- `/var` - installed software
- `/var/log` - log files
- `/proc` - real time system information
- `/tmp` - temp files, cleared on reboot



1. The user name of the logged in user
2. The name of the current machine
3. The current working directory (CWD)
4. User level (\$ for standard user, # for root user)
  - The space immediately to the right of # or \$ is the command line entry point.

Item 3, the current working directory is what we want to focus on at the moment. This tells us how to navigate. The tilde `~` is a short hand for the home directory of the current user which would be `/home/[username]` in my case this would be `/home/jason` which says start at root `/` and find the *home* directory from there find a directory called *jason*. The CWD in the above image `~/bootcamp`. The `cd` (change directory) is how you navigate the file system using a terminal window (aka - console, cli, command line). If i say `cd 01-DevEnvironment` it will look for the *01-DevEnvironment* directory inside on

`/home/jason/bootcamp` as that is my current working directory and the lack of a preceding `/` tells the system to look on a relative file path. If however I add a `/` so that the command reads `cd /01-DevEnvironment` it tells the system to look at the absolute path so the system will look for `01-DevEnvironment` directory to exist directly under root.

## Basic Commands

- `[command] --help` - Returns a help file for any command (or program).
- `sudo` - Super-user do (elevates privs to admin).
- `chown` - CHange OWNership, changes the ownership of a file.
- `chown user1:group1 some-file` - Changes the ownership of some-file to user1 and group1.
- `chmod` - CHange MODe, changes file permissions.
- `chmod +x filename` - Makes a file executable.
- `apt-get` - Retrieves and maintains packages from authenticated sources.
- `apt-get install [package]` - Installs a target package from a repository.
- `apt-get update` - Update your package list.
- `apt-get upgrade` - Upgrade all packages from the updated list
- `apt-get remove` - Remove all packages.
- `apt-get purge` - Remove all packages and their config files.
- `dpkg` - A package manager for Debian-based systems, this is primarily used to deal with files ending with a `.deb` extension.
- `sudo dpkg --install some-pkg.deb` - Installs some-pkg.deb.
- `pwd` - Print working directory (where am I?).
- `ls` - List (a list of files and directories).
- `ls -l` - List long format (file permissions, owner, group, size, last mod, directory name).
- `ls -a` List all (show hidden files).
- `ls -la` List all in long format(`ls -l + ls -a`).
- `ls -R` List recursive (shows all child files).
- `cd` - Change directory.
- `cd ~` - Change directory to home (a shortcut to your home directory).
- `cat` - Concatenate (dumps a file to the console, a handy read only hack).
- `cat [filename]|less` - Pipe the cat command into a paginated CLI (`less --help`).
- `cat [filename]|tail` - Last line of the file, great for looking at log files.
- `cat [filename]|tail -f` - Last line of the file, great for looking at log files.
- `find -name [x]` Find all file for whom the name matches x.
- `find -name [x] Print|less` find all files for which the name matches x and print them to a paginated CLI.

On this system Apache writes log files to `/var/log/apache2`. For this example I only want to retrieve a list of the error logs.

- `cd /var/log/apache2` - Change to the Apache error log directory.
- `find *error.log.*` - Find all error logs in the current working directory (CWD).
- `cd / && locate access.log` - Locate all access logs (recursively) under the root directory.
- `grep` - Globally Search a Regular Expression and Print (Uses Regular Expressions (regex) to search inside of files).
- `*` - wildcard
- `/^d{1,3}\.d{1,3}\.d{1,3}\.d{1,3}z/` - Check an ip address regex in code.

Beyond Linux: Regex in code.

```
//Returns 1 if $string matches a valid IP, returns 0 if it does not.  
$valid = preg_match('/^d{1,3}\.d{1,3}\.d{1,3}\.d{1,3}z/', $string);
```

- `grep ssl error*|less` - Find all ssl errors.
- `grep '\s404\s' access*|less` - Find all 404 errors in the access logs, this is 404 surrounded by whitespace.
- `grep '^127' access*|less` - Find every line that begins with 127 (access logs begin with an IP).
- `sudo grep -ir error /var/log | less` - Find all errors (-i case insensitive) in all logs, we sudo because some log files are only accessible to root.
- `pgrep` - Returns a list of process id(s) for given processes. The process can be requested using regex.
- `pgrep chrome` - Returns a list of all chrome process ids.
- `pgrep chrome | xargs kill -9` - Kills all running chrome processes.
- `cat /etc/passwd|less` - A nice hack to get a list of all users on a system.

tip: Use tab expansions to auto-complete a command or an asterisk as a wild card. The up and down arrows can allow you to browse your command history and replay previous commands. Use `ctrl + r` to search your command history by entering partial commands.

## Summary

In this lesson you learned

- about the Linux file system (Linux)
- basic Linux commands (Linux)

## Additional Resources

- [Ten Steps to Linux Survival](#)
- [Linux Fundamentals](#)

[Next: System Basics](#)

## System Basics

In this section we will learn use some of the basic Linux commands from the previous section to install a few basic software packages.

Login in to your system and open a terminal window using `Ctrl + Alt + T`. Then type the following commands. Pressing enter after each command.

```
sudo apt-get update
sudo apt-get upgrade
```

Lets look at these commands a little closer.

- `sudo` - in short `sudo` will run what ever commands that follow it with root level privileges.
- `apt-get` - `Apt` is a package manager for Linux. This works by maintaining a list of remote repositories from which packages can be installed. Most of the management is done automatically.
  - `apt-get update` - tells the system to update everything it knows about the repositories.
  - `apt-get upgrade` - tells the system to upgrade all packages to their latest versions.

## Terminator

Terminator is a terminal emulator that allows for multiple tabs and split screens. This makes life a lot easier when you are dealing with several command line applications and background processes all at once.

```
sudo apt-get install terminator
```

Now close your terminal window and use `Dash` to find and open Terminator. Open Dash `Super + F` and type `terminator` into the search field. Click the Terminator icon to launch Terminator. You'll notice the Terminator icon is now in the `Launcher` right click the Terminator icon and select `Lock to Launcher` from the context menu.

## VIM

An old school command line text editor. This is really nice to know when you need to edit files on a server or directly on a command line.

```
sudo apt-get install vim
```

## Google Chrome

Download Google Chrome from <https://www.google.com/chrome/browser/desktop/index.html> be sure to save the file. If this returns no errors then your good to go, however, this sometimes fails and if it does you can clean it up with using Apt.

```
cd ~/Downloads
```

- `cd` - Changes your working directory (Change Directory)
- `~` - Tilde Expansion in this case it's a short cut to the home directory (`~` evaluates to `/home/jason`). So `cd ~/Downloads` will change my current working directory to `/home/jason/Downloads`.

You will have downloaded a file called `google-chrome-stable_current_amd64.deb`. `.deb` files are software packages designed for Debian based Linux distros.

```
sudo dpkg --install google-chrome-stable_current_amd64.deb
```

- `dpkg` - A package manager for Debian based systems. Primarily used to work with locally installed `.deb` packages.

```
sudo apt-get install -f
```

Use [Dash](#) to find and open Chrome. Open Dash `Super + F` and type `chrome` into the search field. Click the Chrome icon to launch the Chrome browser. You'll notice the Chrome icon is now in the [Launcher](#) right click the Chrome icon and select *Lock to Launcher* from the context menu. Now right click the FireFox icon in the launcher and click choose *Unlock from Launcher* from the context menu.

## Visual Studio Code

Visual Studio Code (VSC) is the IDE we will be using to write code. Install Atom using the same steps you used to install Chrome. Remember to pin VSC to your launcher after the install. If you cannot find Atom using Dash try launching it from the commandline by typing `atom`.

## Cleanup

Check the contents of your Downloads directory by typing `ls` at the command prompt. You should see the two files we just downloaded and installed.

```
ls ~/Downloads
code*.deb  google-chrome-stable_current_amd64.deb
```

Now type `ls -l` and note the difference between the two result sets.

```
$ ls -l
total 129080
-rw-rw-r-- 1 jason jason 86270030 Feb 16 16:11 code_1.20.0-1518023506_amd64.deb
-rw-rw-r-- 1 jason jason 45892904 Feb 16 16:18 google-chrome-stable_current_amd64.deb
```

Since they have been installed we no longer need the files let's remove them from the system.

```
rm ~/Downloads/*
```

- `rm` - removes a file or a folder
- `-fR` - Force Recursive
  - `-f` - ignore nonexistent files and arguments, never prompt <sup>fn1</sup>
  - `-R` - remove directories and their contents recursively. <sup>fn1</sup>
- `*` - A [wildcard](#) for matching all characters and strings. `rm ~/Downloads/*` will remove everything on the `~/Downloads` path

Now typing `ls ~/Downloads` into the command line will return an empty result set.

## Meld

Meld is a visual diff tool. This is the default tool called by Atom when doing a file comparison.

```
sudo apt-get install meld
```

## Filezilla

Filezilla is my goto [FTP](#) client. While FTP by itself is insecure and not recommended, running FTP over [SSH](#) is secure and Filezilla allows us to do just that. We work with FTP and SSH in later lessons.

## cURL

The best way to think of cURL is as a browser that is used by code.

```
sudo apt-get install curl
```

## Summary

In this lesson you learned

- how to install programs using apt-get (Linux)
- how to install Debian package using dpkg (Linux)
- how to force a broken install using apt (Linux)

## Additional Resources

- [VIM Book](#)
- [Chrome Dev Tools](#)
- [VSC Docs](#)
- [Filezilla Docs](#)
- [SSH Man Page](#)

Next: [LAMP Stack](#)

# Apache Basics

Open a browser and navigate to <http://localhost> and you will land on your machines default apache landing page. That is because the default Apache's configuration points to the path `/var/www/html`, `html` is folder that contains a single file called `index.html`. By default Apache looks for files named `index.*` (where `*` can be any file extension). Let's configure Apache's default path to `/var/www` and see what happens.

First, open a command line and navigate to `sites-available`. On Debian based systems this is where Apache stores per site configurations also known as `vhost` (virtual host) files, on non-Debian systems these may be part of a larger configuration file. Once in the `sites-available` directory run the command to list the directory contents.

```
cd /etc/apache2/sites-available
ls
```

The `ls` command should yield the following results.

```
000-default.conf  default-ssl.conf
```

`000-default.conf` is the default configuration for Apache on Debian based systems. Let's take a look at it's contents. You can read this file without `sudo` but we want to make some changes to it, so lets `sudo`.

```
sudo vim 000-default.conf
```

You will notice a few default settings more commonly called directives in Apache terms. Right now we are only concerned with the `DocumentRoot` directive. Move your cursor down to line 12 and remove `/html` from the end of this line. Remember `i` enters insert mode and `Esc` followed by `:x` saves the file.

The final result will be.

```
DocumentRoot /var/www
```

Now we want to tell apache to reload the configuration. This is a four step process.

- Disable the site configuration `sudo a2dissite 000-default`
- Reload Apache `sudo apache reload`
- Enable the new site configuration `sudo a2ensite 000-default`
- Reload Apache `sudo service apache2 reload`

You can execute all four commands at once with the following.

```
sudo a2dissite 0* && sudo service apache2 reload && sudo a2ensite 0* && sudo service apache2 reload
```

The double ampersand `&&` appends two commands running them one after the other. For example

```
cd /etc/apache2/sites-available && vim 0*
```

Would change your current working directory to `/etc/apache2/sites-available` && `vim 0*` and use vim to open any files that match the pattern `0*` in this case it would only one file `000-default.conf`. Using wild cards can save a few key strokes and cut down on typos. But be careful you don't open the wrong files.



Once you have reloaded the configuration, open your browser and return to <http://localhost>. Now, rather than the default Apache landing page you will see a directory listing with a link to the `html` directory. Clicking this will open the default Apache landing page.

## mod\_ssl

`Mod_ssl` extends the Apache webserver allowing it to work with TLS (Transport Layer Security) connections. While TLS long ago replaced SSL (Secure Sockets Layer), SSL is still the common term used for referring to secure connections be it over SSL or TLS.

In your browser navigate to <https://localhost> notice the `s` in `https` this tells the Apache we want to request a secure version of the website. The page should crash, this is because we haven't told Apache we want to enable a secure version of the site. Now let's open the other configuration in the `sites-available` directory.

```
cd /etc/apache2/sites-available
sudo vim default-ssl.conf
```

As in the previous configuration we are only concerned with the `DocumentRoot` directive. Move your cursor down to line with this directive and remove `/html` from the end of this line. Remember `i` enters insert mode and `Esc` followed by `:x` saves the file.

While the `DocumentRoot` is all we need to change let's take another look at the file. Reopen the file but this time without `sudo`. Around line 25 you will see the `SSLEngine` directive is set to `on`. This tells Apache that this configuration wants to use the SSL module, which we have not enabled yet. Notice lines 32 and 33

```
SSLCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
```

These directives tell Apache where to find information about our SSL certificates. By default Apache creates a snake oil certificate. This is an invalid certificate that offers encryption without verification from a CA (certificate authority) such as DigiCert or RapidSSL. While this is an invalid SSL certificate, it is adequate for testing an SSL connection. Let's enable the SSL module and load the new configuration. To exit vim without saving any changes press `ESC` followed by `:quit!`.

Now that you have returned to the command line load `mod_ssl` (the SSL module).

```
sudo a2enmod ssl
```

Now load the new configuration and restart Apache. This time we will enable the site using *restart* instead of *reload*. *Reload* is a graceful restart that allows minor configuration without killing existing connections. *Restart* is a hard restart of the server that kills all existing connections. *Restart* covers *reload* but *reload* does not cover *restart*.

```
sudo a2dissite d* && sudo service apache2 reload && sudo a2ensite d* && sudo service apache2 restart
```

Now when you navigate to <https://localhost> you will get an insecure warning. Accept the warning and proceed to the site against the browser's advice. You will now see the same file listing you saw when navigating to <http://localhost>

## mod\_rewrite

`Mod_rewrite` provides a rules-based rewriting engine to Apache. This allows the server's admin to rewrite a user's request to do something else. Let's use `mod_rewrite` to force SSL connections when the user makes an http request.

Be sure the `mod_rewrite` is enabled

```
sudo a2enmod rewrite
```

Open the Apache's default configuration.

```
cd /etc/apache2/sites-available && sudo vim 000-default.conf
```

Find the following lines

```
ServerAdmin webmaster@localhost
DocumentRoot /var/www
```

and change them to

```
ServerAdmin webmaster@localhost

RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule (.*?) https://%{SERVER_NAME}/$1 [R,L]

# DocumentRoot /var/www
```

- `RewriteEngine On` - Activates `mod_rewrite`
- `RewriteCond %{HTTPS} off` - Check for a conditions, in this case the https protocol is not active
- `RewriteRule (.*?) https://%{SERVER_NAME}/$1 [R,L]` - If the condition is true http is rewritten to https.
- `# DocumentRoot /var/www` - Comment out the `DocumentRoot` directive. While not required it would guarantee no files get served should the redirect fail (in theory).

In a browser navigate to <http://localhost/> and you will be redirected to <https://localhost/>

## Summary

In this lesson you learned

- how to reload and Apache configuration using `a2dissite` and `a2ensite` (Linux, Apache)
- how to restart the Apache web server using service directives (Linux)
- how to enable Apache modules (Apache)
- how to use Mod Rewrite (Apache)
- how to force a site to use SSL (Apache)

Next: [NPM](#)

## NPM

Non-Parametric Mapping or (as it is most commonly but incorrectly known) Node Package Manager (NPM) is a package management system for managing Node.JS (JavaScript) packages. This is akin to Gems in relation Ruby platform, Composer as it relates to PHP or Python's PIP. Under the hood these may be very different in terms of how they operate but practically speaking they all accomplish the same goal; package management.

Node.JS is an insanely popular platform that allows you to build cross-platform applications using web technology. As a result many web development tools are written in Node. These tools can be installed using NPM. We will need to onstall Node.JS to get started with these tools. Follow the [Debian and Ubuntu Based Linux Distribution](#) instructions from the Node.JS web site. Install the latest greatest version.

After install run the following commands

```
node -v
npm -v
```

These should return versions  $\geq 8.10$  and 5.7.1 respectively.

## Summary

In this lesson you learned

- how to install Node.JS (Linux, Node)

## Additional Resources

- [Debian and Ubuntu Based Linux Distribution](#)

[Next: Git](#)

# GIT

Git is a free and open source distributed [version control system](#) (VCS). Google Trends suggested git is the [most popular](#) VCS in the world. Git is now a part of the Ubuntu standard installation so there is nothing to install.

For a list of git commands type

```
git --help
```

## GitHub

GitHub is a hosted solution popular among both open and closed source developers and will be the solution we use in this course. While we do not need to install Git we will want to bind our GitHub account to our dev machine.

### Setup

The following tutorials will walk you through the process of binding your development machine to your GitHub account using an SSH key. An SSH key uses public key crypto to authenticate your machine against your GitHub account. This allows you to push and pull to and from GitHub without needing to provide a username and password.

### Generating a new SSH key and adding it to the ssh-agent

Accept the defaults for the following. Do not change the file paths, do not enter any passwords.

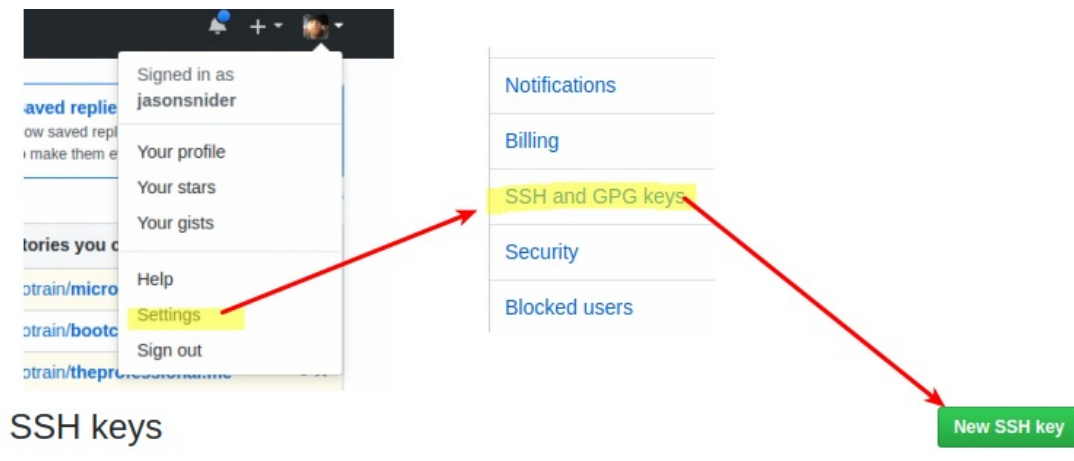
```
ssh-keygen -t rsa -b 4096 -C "YOUR-EMAIL-ADDRESS"
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_rsa
```

### Adding a new SSH key to your GitHub account

Install xclip and copy the contents of the key to your clipboard

```
sudo apt-get install -y xclip
xclip -sel clip < ~/.ssh/id_rsa.pub
```

Log into your GitHub account and find *Settings* in the top right corner under your avatar. Then click on SSH and GPG Keys in the left hand navigation and click the green **New SSH Key** button. Enter a title, this should be something that identifies your machine (I usually use the machine name) and paste the SSH key into the key field.



## Testing your SSH connection

```
ssh -T git@github.com
```

## First-Time Git Setup

Setup your git identity, replace my email and name with your own. For this class we will use VI as our Git editor.

```
git config --global user.email "YOUR-EMAIL-ADDRESS"
git config --global user.name "YOUR-FIRST-LAST-NAME"
git config --global core.editor "vim"
```

In this lesson you learned how to

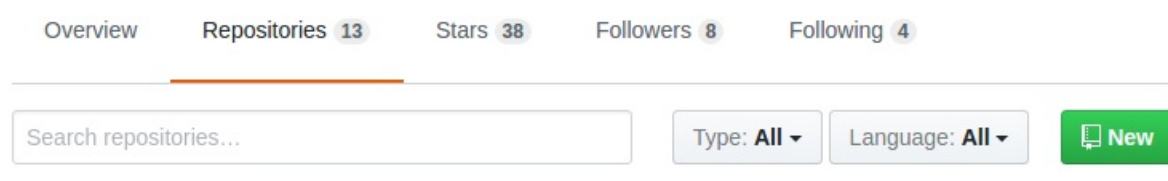
- create an SSH key
- add an SSH key to your GitHub account
- establish a Git identity on your local machine

## Exercise 1 - Create a repository

For this exercise we will create your working directory for this course on GitHub and clone into your local environment. You will do most your work from this repository and push the results to GitHub.

### Create a new repository

Login to your GitHub account and click on the repositories tab then find the *New Repository* button.



Create a project called mtbc (MicroTrain Bootcamp). This is the project you will use for much of this course.

- Create a description, something like *My working directory for MicroTrain's Dev Bootcamp* (you can change this later).
- Be sure *Initialize this repository with a README* is checked.
- Choose a License, for this course I would recommend the MIT License. Since this isn't product it really doesn't matter. Then

click the create button.

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 jasonsnyder ▾

Repository name

/ mtbc ✓

Great repository names are short and memorable. Need inspiration? How about **upgraded-octo-dollop**.

Description (optional)

My working repository for MicroTrain's Dev bootcamp.

☒ **Public**

Anyone can see this repository. You choose who can commit.

☐ **Private**

You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **MIT License** ▾



Create repository

## Clone the repository

Now you will want to pull the repository from GitHub onto your local machine. Then clone your fork onto your local machine. After creating your repository you should be directed to the new repository. If not, you can find it under the repositories tab. Find the *Clone or Download* and copy the URL which will look something like `https://github.com/[your-github-user-name]/mtbc`

jasonsnyder / mtbc

Add RepoUnwatch1Star0Fork0

<> CodeIssues0Pull requests0Projects0WikiSettingsInsights

My working repository for MicroTrain's Dev bootcamp.

Edit

1 commit1 branch0 releases1 contributor

Branch: master ▾New pull requestCreate new fileUpload filesFind fileClone or download ▾

jasonsnyder Initial commit

LICENSEInitial commit

README.mdInitial commit

README.md

Clone with SSH ⓘUse HTTPS

Use an SSH key and passphrase from account.

git@github.com:jasonsnyder/mtbc.git

Download ZIP

Now we will clone this repository into the root directory of our web server. This will allow us to access all of our work through the browser by way of *localhost*.

```
cd /var/www
git clone https://github.com/[your-github-user-name]/mtbc
```

Now use the `ls` command to verify the existence of `mtbc`. If you open your browser and navigate to <http://localhost/mtbc> you will see the following directory structure.

## Index of /mtbc

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">LICENSE</a>	2017-06-13 09:08	1.0K	
 <a href="#">README.md</a>	2017-06-13 09:08	60	

*Apache/2.4.18 (Ubuntu) Server at localhost Port 80*

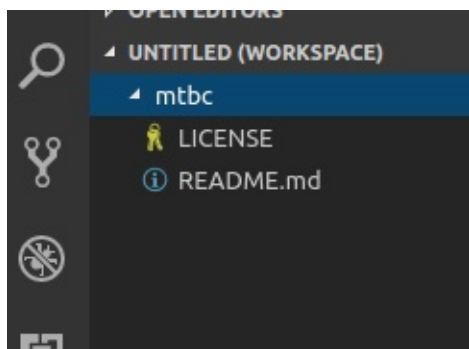
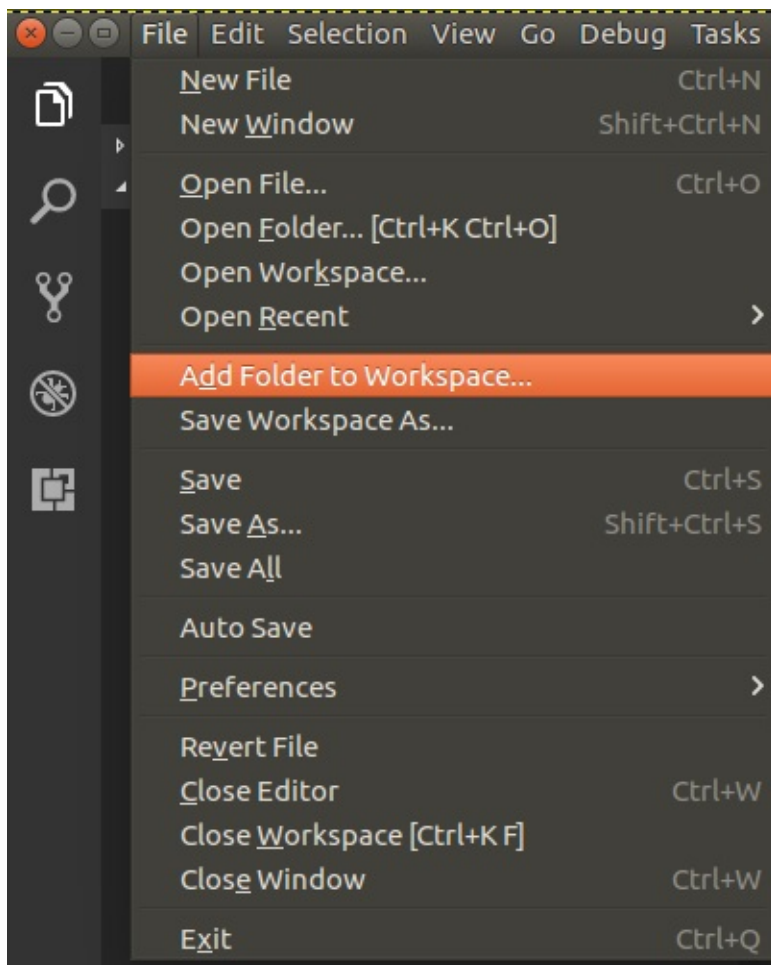
### Summary

In this exercise you learned how to

- create a Git repository on GitHub (Git)
- clone a repository (Git)

## Exercise 2 - Commit a Code Change

Now open VS Code and click into the workspace. Right click choose add a new folder to the workspace. Navigate to `/var/www/mtbc` and press `OK`.



Now click on the file README.md from the *mtbc* project folder. README files are a best practice in software development. README files are human readable files that contain information about other files in a directory or archive. The information in these files range from basic information about the project team to build instructions for source code. A emerging defacto standard is to write in a format called Markdown (.md). A raw Markdown file should be human readable but if you want a formatted version you can use VS Code's *Markdown Preview* by opening the file and pressing `Shift + Ctrl + v`.

Open the file README.md from the *mtbc* project folder in the Atom sidebar and open the *Markdown Preview*. Change the content of the level 1 heading `# mtbc` to `# MicroTrain's Dev Boot Camp`. Save your changes with the keyboard shortcut `[Ctrl + S]`.# MicroTrain's Dev Boot Camp

Open a terminal (command line or CLI) and navigate to the *mtbc* directory.

```
cd /var/www/mtbc
```

Check your repository for changes, this will return a list of dirty files. A dirty file is any file containing a change.



```
git status
```

Commit your code changes

```
git commit README.md
```

VI will open and ask you to enter a commit message.

1. Press the letter [i] to enter insert mode.
2. Then type the message *Proof of concept version*.
3. Press [esc] followed by [:x] and enter to save the commit message.

Push your changes to the master branch.

```
git push origin master
```

You will see a message that indicates the README.md file has been changed.

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

       modified:   README.md
```

```
git commit README.md
```

This will open an editor window that asks you to enter a commit message. Enter *Changed the header* and save the file. You will see a message that indicates the changes to README.md file have been committed.

```
jason@jason-XPS13-9333:/var/www/mtbc$ git commit README.md
[master 67e5568] Changed the header
1 file changed, 3 insertions(+), 2 deletions(-)
```

Finally, push your changes to GitHub.

```
git push origin master
```

```
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 342 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:jasonsnider/mtbc.git
   0da48cc..67e5568  master -> master
```

In the previous example we committed our code changes directly to the master branch. In practice you will never work on a master branch. At the very least you should have a development branch (we call it dev). I like to create branches using the *checkout* command.

```
git checkout -B dev
```

You will see the message *Switched to a new branch 'dev'*.

- `checkout` - This tells git to switch to a different branch.
- `-B` - When following the *checkout* command this tells git to create a new branch. This is a copy of the branch you are

currently on.

- `dev` - The name of the new branch.

In plain English `git checkout -B dev` says make a copy of the branch I am on, name it *dev* and switch me to that branch.

If your are working directly on your dev branch and wanted to push those changes into master it might look like this.

```
git checkout master
git pull origin master
git checkout dev
git rebase master
git checkout master
git merge dev
```

1 `git checkout master` - Switch to the master branch. 1 `git pull origin master` - Pull any new changes into master. 1 `git checkout dev` - Switch back to the dev branch. 1 `git rebase master` - Apply outside changes from master into the dev branch. This will rewind the branch and apply the outside changes. All commits you made after branch deviation will applied on top of the branch deviation. 1 `git checkout master` - Move back to the master branch. 1 `git pull origin master` - Recheck for changes, if any new changes have been applied return to step 3 and repeat. 1 `git merge dev` - Once you have a clean pull, merge your changes into master. 1 `git push origin master` - Push your new changes to the repository.

There is no right way to use git. The only real wrong way to use git is to deviate from that projects branching model. [The Diaspora\\* Project](#) has a very well defined branching model that is typical of what you will see in the real world. I had to come up with one and only one rule it would be to never build on the master branch.

## Summary

In this exercise you learned

- how to create a new branch with the checkout command (Git)
- basic Git commands (Git)

## Additional Resources

- [Documentation](#)
- [ProGit](#).
- [3 Git Commands I use every day](#)

[Next: Programming Basics](#)

## Scripting Basics

In this section you will learn

- Basic control structures
- The basics of Bash scripting

# Bash Scripting and Programming Basics

In this section you will learn how to create a Bash script to automate repetitive tasks.

## Exercise 1 - Scripting Repetitive Tasks

In the previous lesson you learned the four commands for reloading virtual-host configuration. While that may not seem to cumbersome when your not updating your site all that often; it gets a little annoying when your testing updates. We will write a Bash script to reduce the burden of this task. A typical Bash script is little more than scripted arrangement or sequence of Linux commands. In addition to Linux commands shell scripts may accept parameters, may utilize control statements, variables, and functions.

### Requirements

Write a bash script that will reduce the four commands for reloading a virtual host configuration and restarting a server on a Debian based LAMP stack to a single command.

In the previous lesson we used the following four command to reload a vhost configuration and restart the Apache web server.

```
sudo a2dissite * && sudo service apache2 reload && sudo a2ensite * && sudo service apache2 restart
```

That single line equates to the following four lines.

```
sudo a2dissite *  
sudo service apache2 reload  
sudo a2ensite *  
sudo service apache2 restart
```


Anding these statements together makes a copy/paste easier but that is about the only advantage.

[</> code](#) **Create a repository and initial commit**

On GitHub [create a repository](#) called *restart\_apache*.

## Create a new repository

A repository contains all the files for your project, including the revision history.

**Owner**  
 **stack-x** ▾

**Repository name**  
 ✓

Great repository names are short and memorable. Need inspiration? How about **fantastic-sniffle**.

**Description** (optional)

☒ **Public**  
Anyone can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

[?](#)

**Create repository**


Clone the restart\_apache repository onto your local development machine.

> 0 releases 1 contributor

[Create new file](#) [Upload files](#) [Find file](#) [Clone or download ▾](#)

**Clone with SSH** [?](#) [Use HTTPS](#)

Use an SSH key and passphrase from account.



[Download ZIP](#)

```
cd ~  
git clone https://github.com/YOUR-USERNAME/restart_apache
```

</> [code](#) **Proof of Concept**

Often times I like to start with a simple proof of concept, this is working code working code that either gives you a starting point or talking point. In some projects proof of concept code may represent a complete working solution but may not be considered the optimal solution.

Add `~/restart_apache` as a [new folder](#) in your VSC workspace and create a new file `re.sh`.

By default, Ubuntu executes shell scripts using the Dash interpreter. Dash is faster than Bash by virtue of a lack of features and limited syntax, making it ideal for quickly parsing out a large number of simple start up scripts. Bash is better suited for interactive scripts, since these are typically run as one off programs the performance hit is a non-issue. Our scripts will invoke the Bash shell. Add the following to `re.sh`, this will allow you reload all apache configurations with a single command.

A shebang `#!/` followed by a path is used to invoke an interpreter, this must be the first line of the file.

```
#!/bin/bash
```

In Bash any line that begins with a `#` denotes a comment and does not processed by the interpreter. Comments are used to explain the program to other humans.

```
# Move the current execution state to the proper directory
cd /etc/apache2/sites-available

# Disable a vhost configuration
sudo a2dissite *
sudo service apache2 restart

# Enable a vhost configuration
sudo a2ensite *
sudo service apache2 restart
```

Now we want to make sure the file is executable by adding the executable flag.

```
cd ~/restart_apache
chmod +x re.sh
```

Since many of the commands require root access you will want to `sudo` this script when you run it.

```
sudo ./re.sh
```

You should see the following results.

```
Site 000-default disabled.
Site default-ssl disabled.
To activate the new configuration, you need to run:
  service apache2 reload
Enabling site 000-default.
Enabling site default-ssl.
To activate the new configuration, you need to run:
  service apache2 reload
```

## Commit your changes and push them to GitHub

```
git add .
git commit -a
```

VI will open and ask you to enter a commit message.

1. Press the letter [i] to enter insert mode.
2. Then type the message *Proof of concept version*.
3. Press [esc] followed by [:x] and enter to save the commit message.

Push your changes to the master branch.

```
git push origin master
```

## Semantic Versioning

`</>` Semantic versioning is a community standard that helps you communicate the backwards compatibility of a change. We will use it here as an introduction to the concept.

1. Create the file VERSION.txt
2. Add the text *1.0.0*
3. `git add VERSION.txt` `git commit VERSION.txt`

VI will open and ask you to enter a commit message.

1. Press the letter [i] to enter insert mode.
2. Then type the message *Version 1.0.0*.
3. Press [esc] followed by [:x] and enter to save the commit message.

Push your changes to the master branch.

```
git push origin master
```

## Add a tag

In addition to Semantic Versioning a common practice is to tag significant versions.

1. `git tag 1.0.0`
2. `git push origin --tags`

Go to your project on GitHub and find everything that is tagged.

## Summary

In this exercise you learned how

- to create a basic shell script (Bash, Programming)
- commit your code changes (Git)
- apply semantic versioning (Git)
- use tags to create code releases (Git)

## Arguments and conditionals.

### Conditionals

A conditional (aka if-then-else) is a programming construct that uses equality to make decisions. Examples of equality given the variable a is equal to one and the variable b is equal to 2.

If a = 0 and b = 1

- `a == b (a -eq b) //false`

- `a < b (a -lt b ) //true`
- `a > b (a -gt b ) //false`
- `a == 0 (a -eq 0 ) //true`
- `a >= 0 (a -gte 0 ) //true`
- `a <= 0 (a -lte 0 ) //true`

## Arguments

Arguments or parameters are additional data that are supplied when invoking a program, function, method, sub-route, etc. These provide a specific context upon which a called block of logic will execute.

Examples

- `vim hello.txt` - Opens vim and loads the file `hello.txt`
- `add(1, 2)` - Passes two arguments into a function called `add` (the values 1 and 2). One might suspect that this would return the number 3.
- `Check.word('random')` - This passes *random* as an argument into the `word` method of the (fictitious) `Check` class. Perhaps we are checking the spelling of the word *random* or maybe this is an argument that tells the method to return a random word.

## Exercise 2 - Working with Arguments and Conditionals

In this exercise we will work with the file `~/restart_apache/re.sh` on a branch called *feature/arguments*. Create a [feature branch](#) called *feature/arguments*.

```
cd ~/restart_apache
git checkout -B feature/arguments
```

We have reduced four repetitive commands down to a single command, but there is a problem. This only works with a single immutable configuration and an immutable single service directive. It would be far more useful if we could specify which virtual host configuration and which service directive we wanted to use. Let's rewrite `re.sh` to take some arguments.

Our new shell will take two arguments; the target virtual host configuration and the service directive. Bash accepts arguments using a numeric index which starts at zero, zero however, is the name of the script so the argument that sits at index one will access the first parameter. In Bash, the value of stored variables are accessed using a dollar sign. Combining a dollar sign with a number `"$1"` will allow you to access a given argument.

Our first argument will be the virtual host configuration we want to work with and the second argument will be the service command. We will set these to an aptly named variable to make them easier to work with. We will store the first argument in a variable called *CONFIG* and the second in a variable called *COMMAND*. When referencing a variable in bash it is advisable to always [quote the variable](#).

</> [code](#) Add the following lines right after `#!/bin/bash`.

```
CONFIG="$1"
COMMAND="$2"
```

The first thing we want our program to do is to verify we have the correct number of arguments. We will do this with an equality check (if-then-else). In Bash `$#` will return the number of input parameters (starting at index number one). We can check this with an if/then statement *if [ \$# -ne 2 ] then*. In this context *-ne* translates to *not equal* or in plain English *if the number of input parameters is not equal to 2 then do something*. In our case we will want to provide the user feedback about the expected arguments and exit the program.



Add the following lines to the file. Below the `CONFIG` and `COMMAND` variables but above the lines from the previous example. In bash `echo` is a command that writes its arguments to the standard output while `exit` stops the execution of the program and returns control back to the caller. In this case both the standard output and the caller would be the terminal.

```
if [ $# -ne 2 ]
then
    echo "$0 requires two parameters {virtual-host} {restart|reload}"
    exit 1
fi
```

Finally, replace the `*` with a call to the `CONFIG` variable by prefixing `CONFIG` with a dollar sign `$CONFIG` and do the same for `COMMAND`

```
sudo a2dissite "$CONFIG"
sudo service apache2 "$COMMAND"

sudo a2ensite "$CONFIG"
sudo service apache2 "$COMMAND"
```

`./re.sh 000* restart`

Commit your changes to *feature/arguments* with the message *Added the ability to specify virtual hosts and service command*.

Push your new feature branch to GitHub, you can delete this branch once the feature is complete.

```
git push origin feature/arguments
```

`</>` `code` Update `README.txt` so people know how to use it. Add something like the following.

```
## Usage
Clone the repository or download the latest release.

From a command line call re.sh with two arguments.
1. The vhost configuration
1. The service directive {restart|reload}
```sh
./re.sh 000* restart
```
```

Commit the change with the message *README updates*

```
git commit -a
git push origin feature/arguments
```

Then open `VERSION.txt` and move the version to 1.1.0 and commit with a message of *Version 1.1.0*.

```
git commit -a
git push origin feature/arguments
```

Push your changes to the master branch.

```
git push origin master
```

Merge your changes into master

```
git checkout master
```

```
git merge feature/arguments
git push origin master
```

`</>` code Tag a new version

1. `git tag 1.1.0`
2. `git push origin --tags`

Now that all code changes have been applied to master you can remove your working branch.

```
git branch -D feature/arguments
git push origin :feature/arguments
```

## Summary

In this exercise you learned how

- work with arguments and variables (Bash, Programming)
- use an if-then statement (Bash, Programming)
- create a feature branch (Git)
- merge a feature branch into master (Git)
- remove old working branches (Git)

## Exercise 3 - Reject unwanted service commands

For this exercise, create a feature branch called *feature/validate*. When you are finished increment the version to 1.2.0 then merge into and push to master.

### Requirements

The product owner has requested that we only be allowed to pass *reload* or *restart* into the service command. To achieve we will need to run a test against the second argument to verify it matches a valid command.

To accomplish this we can then test against the value of the `$COMMAND` argument to make sure it is in the approved list. If it is then we can allow the program to proceed. Otherwise we will return an error message to the user.

We will start with a simple if-then-else statement. This if statement is differs from the previous exercise in that it adds an OR statement. In Bash OR statements are represented as a double pipe `||`. If either of these conditions are true then the code inside the `then` block will execute. Otherwise we will drop into the `else` block.

```
# only allow reload or restart.
if [ "$COMMAND" == "reload" ] || [ "$COMMAND" == "restart" ]
then
    # ... (previous logic) ...
else
    echo "ERROR: $COMMAND is an invalid service command {restart|reload}"
    exit 1
fi
```

`</>` code Once we have our basic statement in place cut and paste the reload/reset logic into the `then` block.

```
if [ "$COMMAND" == "reload" ] || [ "$COMMAND" == "restart" ]
then
    # Move the current execution state to the proper directory
    cd /etc/apache2/sites-available

    # Disable a vhost configuration
```

```
sudo a2dissite "$CONFIG"
sudo service apache2 "$COMMAND"

# Enable a vhost configuration
sudo a2ensite "$CONFIG"
sudo service apache2 "$COMMAND"
else
    echo "ERROR: $COMMAND is an invalid service command {restart|reload}"
    exit 1
fi
```

```
git add .
git commit -m 'Added the ability to reject unauthorized service directives'
git checkout master
git merge feature/validate
git push origin master
```

</> `code` Update VERSION.txt to 1.2.0

```
git add .
git commit -m 'Version 1.2.0'
git push origin master

git tag 1.2.0
git push origin --tags
```

## Attention to Detail

</> `code` Check out the format of our new error message. It begins with the word *ERROR*: in all caps. This is good UX in that it remove any ambiguity about what the message. To keep user feedback consistant prefix the error message in the first if statement with *ERROR*:

```
git commit -am 'Improved messaging'
git push origin master
```

</> `code` Since this commit does not add or remove it is considered a patch so we will increment version as a patch. Update VERSION.txt to 1.2.1

```
git commit -am 'Version 1.2.1'
git push origin master

git tag 1.2.1
git push origin --tags
```

## Exercise 4 - Loops and Arrays

When I think of a loop I'm usually thinking about iterating over or parsing out some sort of a list. This might be an array of service commands or all of the configuration files in the */etc/apache2/sites-available/* directory. In this exercise we build an array of valid service commands and iterate over those commands.

For each element in the *COMMANDS* array where an element is defined by the variable *COMMAND*, if an element exists (meaning we have not iterated past the end of the list) `do` echo the value of *COMMAND* back to the user otherwise `break` the loop or *do echo the value of COMMAND until the list is done*.

Create the file *~/bash/loop.sh* and make it executable

```
mkdir -p /var/www/mtbc/bash
cd /var/www/mtbc/bash
touch loop.sh
chmod +x loop.sh
vim loop.sh
```

Add the following lines and execute the file.

```
#!/bin/bash

# A list of service commands
COMMANDS=( reload restart )

for COMMAND in "${COMMANDS[@]}"
do
    echo $COMMAND
done
```

Execute the code

```
./loop.sh
```

[</> code](#) Commit your code and push it to the master branch of the mtbc project. *Bash exercise 4 - loop example*

## Summary

In this exercise you learned

- how to create an array
- how to iterate over an array
- how to create a file using the *touch* command

## Exercise 5 - Loop through all files in a Directory

For each file in `VHOSTSPATH` array where a file is defined by `FILENAME`, if an element exists (meaning we have not iterated past the end of the list) `do` echo the value of `FILENAME` back to the user otherwise `break` the loop or `_do` echo the value of `FILENAME` until the list is done.

Add the following to `/var/www/mtbc/bash/loop.sh` with the following.

```
# List all of the configuration files in the _/etc/apache2/sites-available/_ directory
VHOSTS_PATH=/etc/apache2/sites-available/*.conf

for FILENAME in $VHOSTS_PATH
do
    echo $FILENAME
done
```

Running `./loop.sh` will now yield the following.

```
reload
restart
/etc/apache2/sites-available/000-default.conf
/etc/apache2/sites-available/default-ssl.conf
```

[</> code](#) Commit your code and push it to the master branch of the mtbc project. *Bash exercise 5 - loop example*

## Summary

In this exercise you learned

- loop through files in a directory

## Exercise 6 - Strings

String concatenation is the addition of one string to another typically through the use of variables. Create an executable Bash file at `/var/www/mtbc/bash/string.sh` and add the following code.

```
#!/bin/bash

STRING1='Hello'
STRING2='World'
echo "${STRING1} ${STRING2}"
```

`</>` [code](#) Commit your code and push it to the master branch of the mtbc project. *Bash exercise 6 - string example*

## Summary

In this exercise you learned how to

- concatenate two strings.

## Exercise 7 - Not Empty

The [comparison operator](#) `-z` returns true if a string has a length of zero. `!` is the operator for not so `if [ ! -z "$STRING" ]` equates to true if the string contains any characters.

`-z` is an equality check for zero.

Create an executable Bash file at `/var/www/mtbc/notEmpty.sh` and add the following logic.

- While the variable `STRING` is not equal to `"Hello World"` continually check the value of string.
- If `STRING` has a length of zero change the value of `STRING` to `"Hello"`.
- If `STRING` has anything other than a zero length append `" World"` to the current value.

```
#!/bin/bash

STRING=' '
while [ "$STRING" != "Hello World" ]
do
    if [ -z "$STRING" ]
    then
        STRING="Hello"
    else
        STRING="${STRING} World"
    fi

    echo "$STRING"
done
```

`</>` [code](#) Commit your code and push it to the master branch of the mtbc project. *Bash exercise 7 - not empty example*

## Exercise 8 - String Position

String position allows you extract parts of a string by specifying a numeric index starting at 0. Bash allows you to define string positions using colons with a variable.

A starting position of 0 will return the entire string.

```
${VARIABLE:starting-position}
```

A starting position of 0 and end position of 4 will return the first four characters of a string.

```
${VARIABLE:starting-position:length}
```

A starting position of 0 and end position of -4 will return everything between first character and fourth from the end.

```
${VARIABLE:starting-position:end-position}
```

Create the executable Bash file `/var/www/mtbc/stringPosition.sh` and add the following logic.

```
#!/bin/bash

STRING="The quick brown fox jumped over the lazy dog"

echo "${STRING:41}"
echo "${STRING:4:5}"
echo "${STRING:36:-4}"
```

This string will return

```
dog
quick
lazy
```

[</> code](#) Commit your code and push it to the master branch of the mtbc project. Bash exercise 8 - string position example

## Lab

Update the `re.sh` script such that

- The user will receive an error message if they attempt to reload/restart a virtual-host file that does not exist in `/etc/apache2/sites-available`.
- The system will return a list of valid files as a part of the error message.
- The system will exit prior to evaluating the reload command if an invalid virtual-host has been chosen.

## Extra Credit

Google "formatting bash", find and read some results about formatting output text in bash. Look for things like escape characters, font colors, bold and italic text, carriage returns/newlines etc. Using this new knowledge improve the output of the error messages so that they may read in a more intuitive, scan friendly manner.

## Questions

1. In Bash, why should you quote variables when referencing them?
2. In Bash, what is `-z`?
3. In Bash, when do you prefix a variable with a dollar sign?
4. How many ways can you come up with to create an array in Bash.
5. What is an array?
6. What does the `*` in `/etc/apache2/sites-available/*.conf` do?

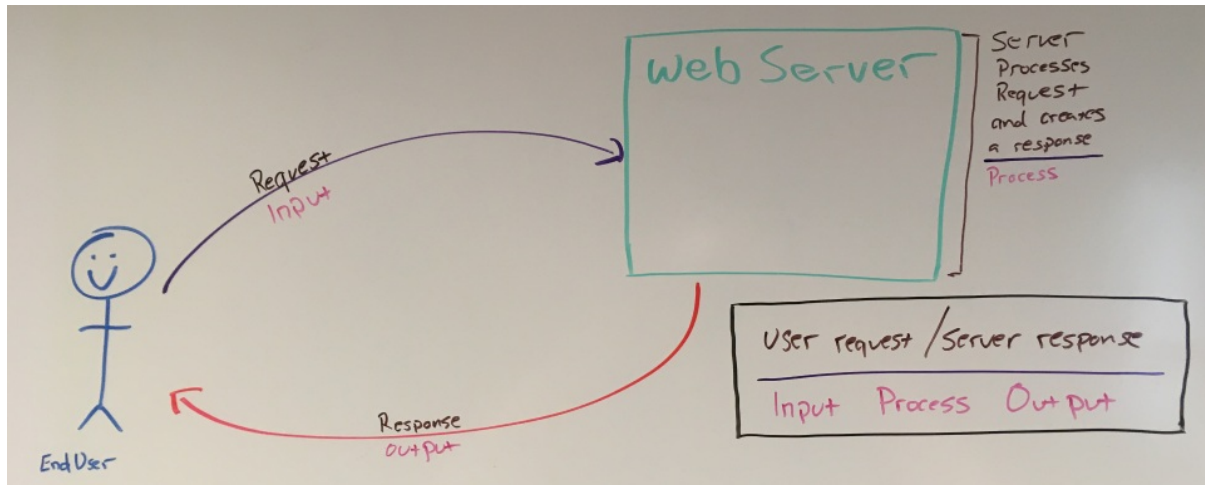
## Additional Resources

- [A Few Words on Shell Scripts](#)
- [Bash Guide for Beginners](#)
- [Advanced Bash-Scripting Guide](#)
- [service](#)
- [Better Bash Scripting in 15 Minutes](#)
- [Better Bash Scripting in 15 Minutes \(Discussion\)](#)
- [Bash String Manipulation](#)
- [FLOZz' MISC » bash:tip\\_colors\\_and\\_formatting](#)

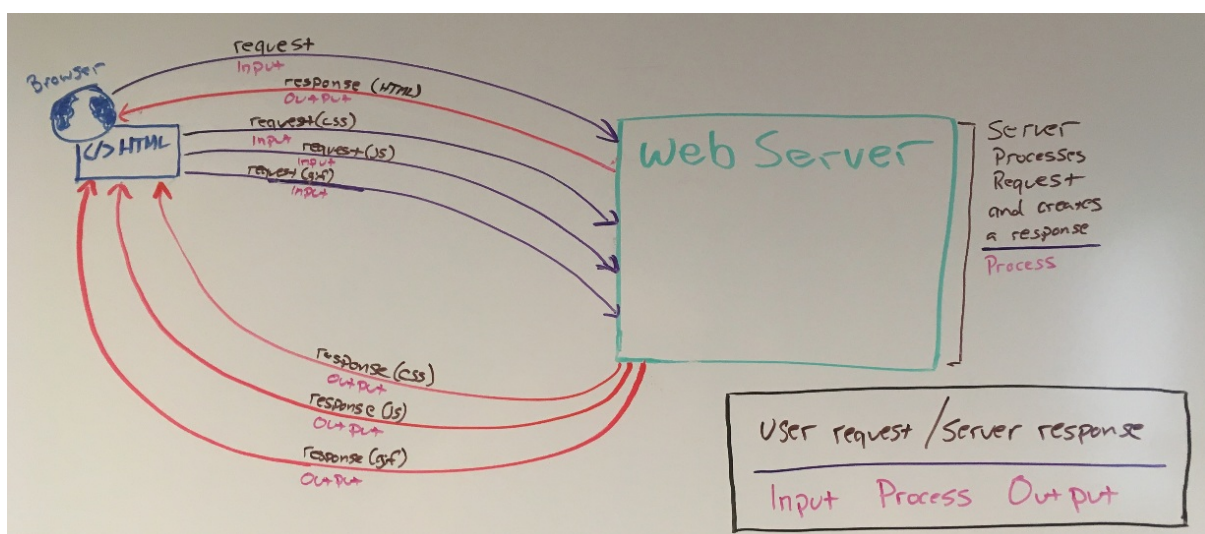
Next: [PHP Basics](#)

## User Request and Server Response

Every web page is created by a server responding to a user request. The user makes a request, the server processes that request and creates a response. That response is most commonly in the form of an HTML document. In reality this response could be sent in any imaginable document format. This is an accurate but simplistic view of an IPO (input, process, output model). In practice, this process may be repeated multiple times in loading/interacting with a single web page.



Let's replace the work user with client. In this description we will use *end user* to represent a human actor and *client* to represent software, in this case the browser. An end user opens a web browser (the client) and enters `www.youtube.com` into the address bar and presses [ENTER]. At this point the client (browser) makes a request with no parameters to the server at `www.youtube.com` the server sees there are no parameters, processes the request and returns the YouTube homepage as an HTML document. This only returns an HTML document that describes how the page should be layout. There are no images, no videos, no styles, no colors, no javascript, no css, no special fonts, and so on<sup>1</sup>. These are all additional requests made by the client. The HTML document contains special references that when interpreted by the client (in this case a web browser) provides instructions for that client to make a request to a server (on behalf of the end user). Each of these additional requests are processed by a server which in turn creates a response and serves that response to the client. These responses are not typically HTML documents, rather these tend to be images, videos, stylesheets, javascript files, etc.



In summary a basic web page is served as follows

- A client makes a request to a server.



- That server processes the user request and returns an HTML document.
- The client processes that HTML document and makes additional requests as instructed by the document.
  - Each of these requests is processed by a server.
  - Each of these servers creates a response and serves it to the client accordingly.
  - The client processes each of these responses and processes the instruction sets accordingly.
    - These may or may not lead to additional server requests.

## Summary

In this section you learned how a user request leads to a server response.

## Footnotes

1. unless these are hardcoded into the page which may be done with description.

# HTML

In this unit you will learn.

- How to mark up a basic web page.
- Create a web form.
- Process user input with PHP
- Send Emails using an API.

# HTML5

## Introduction

This unit will focus on the fundamentals of HTML and building a personal website aimed at personal branding.

## HTML

Hypertext Markup Language (HTML) is a system of elements and attributes that defines the layout of a web page. This system uses markup tags to represent elements ( `<p>This is a paragraph.</p>` ) and attributes ( `<p style="color: blue;">This is a paragraph with blue text.</p>` ) to further describe these elements; to define the context of text and objects on a page.

The official HTML documentation is available from the [W3C](#) this is the standards version of the documentation aimed at browser makers. The [MDN web docs](#) in my opinion is a better practical source of documentation.

## HTML Elements

A typical web page is rendered as an HTML document. HTML documents are made up of HTML elements (aka tags). HTML has two types of elements; inline elements and block-level elements. An HTML element will typically have an opening and a closing tag. `<[element]>`Closing tags have a slash `</[element]>`.

## A Basic Web Page

The World Wide Web started off as a markup standard for sharing academic research over the internet. In those days you got by with just a few tags. In most cases you could think laying out a web page the same way you might write a college paper. A header, sub headers, paragraphs, images, links, bold, italic, lists and tables. Let's start by exploring a basic web page in the form of an academic paper.

Most HTML documents start with a header. Headers are marked up `<h*>` the *can be any value between 1-6 with \*1 being the highest*. `<h1>` opens a header while `</h1>` closes a header. Thus these are called opening and closing tags respectively. `<h1>` is generally accepted as a page title

## Headers

Example headers

```
<h1>This is a level 1 header.</h1>
<h2>This is a level 2 header.</h2>
<h3>This is a level 3 header.</h3>
<h4>This is a level 4 header.</h4>
<h5>This is a level 5 header.</h5>
<h6>This is a level 6 header.</h6>
```

## Paragraphs

An example paragraph.

```
<p>This is a paragraph, paragraphs have an opening and closing tags. Paragraphs are block level elements with margins on the top and bottom.</p>
```

## Images

Images require special attributes. Adding an image tag does not place an image in your page. Rather it creates a reference to an image and defines a space for that image. It is up to the browser to get the image and display as per the author's wishes.

An image has two required attributes **src** and **alt**. *src* is the URI of the image while *alt* holds a non-graphical description.

```

```

## Links

Anchor *a* elements are used to create hyperlinks commonly called links. These are used to link documents together on the web.

An image has one required attribute **href**. This is the URL to any web resource to which you want to link the current document.

```
<a href="https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a">The a tag (HTML Anchor Element)</a>
```

## Bold

The *strong* element is typically rendered in boldfaced type as is the *b* element. The *b* element is considered irrelevant on today's web.

```
<strong>details of strong importance</strong>
```

## Italic

The *em* element is typically rendered in italic type as is the *i* element. The *i* element is considered irrelevant on today's web.

```
<em>emphasis placed here</em>
```

## Lists

## Tables

## Example paper

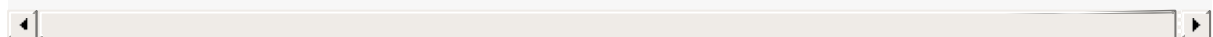
```
<h1>Dogs and Foxes</h1>
<p>The quick brown fox jumps over the lazy dog.</p>

<h2>The truth about dogs</h2>
<p>Some dogs are lazy and just lay around. You can learn more about dogs <a href="https://www.example.com/dogs">here</a></p>

<h3>More about dogs</h3>
<p>There are other dog facts that we are about to get into.</p>

<h2>The truth about foxes</h2>
<p>Many foxes are quick and like to jump.</p>

<h3>More about foxes</h3>
<p>There are other fox facts that we are about to get into. You can learn more about foxes <a href="https://www.example.com/foxes">here</a></p>
```



## HTML Comments

In programming, comments are strings of text that are not processed by a compiler or an interpreter. Comments are for human consumption and are used to help humans follow the flow of source code. Most languages will define their own special syntax for presenting a comment. In HTML, comments are wrapped in comment tags `<!-- this is a comment -->`.

## The Markup

### HTML5

```
<!DOCTYPE html>
<html>
  <head>
    <!-- meta data goes here -->
    <title><!-- Page Title --></title>
  </head>
  <body>
    <!-- layout and content goes here-->
  </body>
</html>
```

## Exercise 1 - Hello World, Getting Started With HTML 5

The exercises in this section will focus on building a website personal web site tailored to developing a personal brand, building a portfolio, etc.

1. [Create a repo on GitHub](#) called *example.com*.
2. Clone the repo into the top level of your Apache server `/var/www`
  - o From a command line

```
cd /var/www/
git clone https://github.com/YOUR-USERNAME/example.com/
cd /var/www/
```

1. From the VCS explorer add the path named *public/index.html* to the project.
2. Open a browser and navigate to <http://localhost/example.com/public/index.html> (At this point you will see a blank page).
3. Now open the file and paste the above mark markup into the file.
4. Change the title element to `<title>Hello World</title>`
5. Add an `h1` element to the body of the page which also reads *Hello World* `<h1>Hello World</h1>`.
6. Now refresh your browser and you will see the text *Hello World*.
7. Add a paragraph tag below the top level header that reads `<p>Welcome to my web site.</p>`
8. Commit your changes and push to master. Stage new files to be committed, in this case *public/public/index.html*.

```
git add .
```

Commit all files, you'll be asked for a commit message say something like *Initial page structure*.

```
git commit -a
```

Add a comment like *Added the initial home page*. then `[esc] :x [enter]` to save. Then push your changes to master.

```
git push origin master
```

Now you can access your page through the following URL.

- <http://localhost/example.com/public/index.html>

For a simple site like this you can usually get away with the above URL, for a more complicated site however it's better to work through a domain name. I like to localize my domain names so that I can still get to the production site. I do this with the *loc* prefix so that *example.com* becomes *loc.example.com*.

```
cd /etc/apache2/sites-available
sudo vim example.com.conf
```

Add the following

```
<VirtualHost 127.0.0.30:80>

    ServerName loc.example.com

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/example.com/public

    # Allow an .htaccess file to serve site directives.
    <Directory /var/www/example.com/public/>
        AllowOverride All
    </Directory>

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Open your local hosts files

```
sudo vim /etc/hosts
```

Add the following line

```
127.0.0.30    loc.example.com
```

Load the configuration and restart the server

```
sudo a2ensite example.com && sudo service apache2 restart
```

Navigate to your new local new domain

- <http://loc.example.com>

## Exercise 2 - HTML Validation

Sometimes our pages do not display as we expect, this is often due to invalid HTML. You can check the validity of your HTML using the W3C Markup Validation Service.

1. Open a browser and go to the [W3C Markup Validation Service](https://validator.w3.org/).
2. Select the third tab *Validate by Direct Input*
3. Copy and paste your HTML5 document code into the window and click *Check*.

## Meta Data

Meta data is data data. For a typical web page, the data is the content that falls between the head and body tags. Meta data helps to describe and classify that data and/or the functionality of your web page. Meta data can be an attribute of a single element or added to the `head` of a document in the form of a meta tag.

Best practices are things that ought to be done given there is not a good reason not to and provided there is not an alternative that better suits a given situation. If I had to pick three pieces of meta data that should always be implemented, they would be as follows.

- `<html lang="en">` - Defines the language of the web page. This would most likely be used by assistive technologies such as screen readers or an automated translator.
- `<meta charset="UTF-8">` - Defines the character set you are using so that there will be no confusion between your source code and the rendering engine. For a data driven web site you will want your websites encoding to match that of your database; UTF-8 is the most common encoding.
- `<meta name="viewport" content="width=device-width, initial-scale=1.0">` - Used by the browser to allow the developer of the site to declare how the site should be viewed across devices.

## Exercise 3 - Meta Data Best Practices

Update ex1.html so that

- The language is declared as *English*.
  - Commit with the message *Set the default language to English*
- The charset is declared as *UTF-8*.
  - Commit with the message *Set the default encoding to utf-8*
- The view port is declared as *content="width=device-width, initial-scale=1.0"*.
  - Commit with the message *Added a responsive viewport*
- Push all changes to master

## HTML Elements

HTML elements more commonly known as tags are bits of markup that provide semantic meaning to text and objects. This markup is interpreted by outside programs (such as browsers, bots, spiders, crawlers, etc) which will often act on your content based on that content's markup. For example `<h1>title</h1><h2>Sub Title</h2>` tells the program reading your page that *Sub Title* belongs to *title* and that *title* should be treated as title of the page (or all content until the next `<h1>` tag is encountered) while **Sub Title** identifies the next block of or all content until the next `<h2>` or `<h1>` tag is encountered. The original goal of HTML was to provide a common format in which we could send academic research papers over the wire. In short, HTML was designed to mimic a word processor. The body of one of those documents may resemble the following. In most cases your HTML elements will have both an opening and a closing tag. Elements open with `<[element]>` and close with `</[element]>` the difference here is `<` vs `</`.

```
<h1>HTML Elements</h1>
<p>HTML elements more commonly know as tags are bits of markup...</p>
<h2>HTML Global Attributes</h2>
<p>Attributes bring your markup to life. Attributes allow for programming...</p>
<h3>Event Handler Attributes</h3>
<p>Event Handler Attributes (UI Events) allow a user to interact...</p>
<p>Here a list of...</p>
<ul>
  <li>Item one</li>
  <li>Item two</li>
  <li>Item three</li>
</ul>
<h2>Summary</h2>
```

```
<p>In summation...</p>
```

Lets review some of these tags.

- h1, h2, h3, h4, h5, h6 - HTML supports 6 header levels, these should always be nested.
- p - Identifies text as a paragraph.
- ul - Identifies an unorganized list. This will create a bulleted list. An unorganized MUST contain one or more list items `<li>`.
- ol - Identifies an organized list. This will create a numbered list. An organized MUST contain one or more list items `<li>`.
- li - A list item, represents an item in either an unorganized or organized list. This MUST be wrapped in a `<ul>` or `<li>` element.
- div - Divs are block level elements that are used to represent divisions in an HTML document. These are typically used to divide a page into sections. These may be used for a logical page division or as anchor to apply style and attributes.
- a - An anchor tag. This is used to create links and is arguably the foundation of the World Wide Web. An anchor tag becomes a link by adding a href ([Hypertext REFERENCE](#)) attribute `<a href="https:\\www.">Example</a>`.

## Exersice 4 - More Elements

Update hello.html as follows. I hope you type this out rather than copy and pasting the entire blob. Read up on character entities by following the links you will be embedding in the page.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>World</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <h1>Hello World</h1>
    <p>Welcome to my web site.</p>
    <h2>HTML Elements</h2>
    <p>
      To paraphrase the lesson text [...] For example &lt;h1&gt;h1&lt;h2&gt;title&lt;/h1&gt;... Notice the use of
      character entities when wanting show the tags in HTML.
    </p>
    <h2>Character Entities</h2>
    <p>Since the keyboard does not have a &copy; key we need a way to reference this so we say &amp;copy;. Addi
      tionally, greater than and less than are interpreted as HTML tags. These are examples of symbols that we may wa
      nt to display but will not be able to with out a work around. This is where character entities come into play.
    </p>
    <ul>
      <li><a href="https://stackoverflow.com/questions/1016080/why-are-html-character-entities-necessary">A S
        tack Overflow thread on the topic.</a></li>
      <li><a href="https://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references">A Wikipedi
        a artcile on the topic.</a></li>
      <li><a href="https://dev.w3.org/html5/html-author/charref">The W3C reference.</a></li>
    </ul>
    <h2>Summary</h2>
    <p>In summation...</p>
  </body>
</html>
```

## HTML Global Attributes

Attributes bring your markup to life. Attributes allow for programming hooks, style and meta-data to be applied to your web page. While HTML has defined a number of standard attributes it allows for custom attributes to be defined.

## Image Tag



When it comes to working with attributes, the image (img) element is a great place to start. Image is a self-closing tag. Self-closing tags do not require a closing tag because they do not contain any content, they only use attributes to mark content to the screen.

Image has two required attributes src and alt ``. Src tells the document where to find the image to be displayed while alt displays alternative text in case the image cannot be displayed. In most cases an attribute will always have a value, this is written as attribute="Some Value". This is often referred to as a key-value pair. The attribute or left side of the equation is the key and the right side of the equation is the value (conceptually speaking it's key="value").

Before you can add an image, you will need an image to link to. Lets use a Gravatar. Gravatar is a free service that allows a user to upload an avatar that will follow them from site to site by hashing the users email address into the name of a file. This is popular in the dev community and used by other services such as GitHub.

[Head over to Gravatar](#) and create a profile and use the provided URL, if you do not want to create a Gravatar account use the following URL.

- <https://www.gravatar.com/avatar/4678a33bf44c38e54a58745033b4d5c6?d=mm>

You would mark this up as:

```

```

## Exercise 5 - Add an Image

Create the path `/var/www/example.com/public/index.html`.

- Markup a valid HTML 5 template.
- The title tag should read *Hello, I am YOUR-NAME*
- Add an H1 element to the body of the document.
  - The contents of this tag SHOULD read *Hello I am YOUR-NAME*
- Add an image element to below the H1 element.
  - The src attribute MUST point to a Gravatar image
  - The value of alt attribute SHOULD be YOUR-NAME

## The Style Attribute

Cascading Style Sheets (CSS) is a language for describing the style of an element. The style attribute in HTML allows you to add a CSS description to a single HTML element. Describing a font's color is a common use of CSS. In CSS we describe style using property's which are key to value pairs separated by a colon [key]:[value]. Using the color property will allow me to describe the font color of an element.

## Exercise 6 - Style an Element

Change the `/var/www/example.com/public/index.html` so that:

- The font color of the top level header is orange.
- The image is presented as a circle.

```
<h1 style="color: #ff9900;">Hello, I am YOUR-NAME</h1>

```

## The Class Attribute

Another way to apply a CSS definition to an HTML element is by defining selectors in a CSS document. A CSS document can be in its own file or you can define a CSS document by adding a style element to the head of an HTML document. We will discuss CSS in detail in later, for now let's apply a quick example.

## Exercise 7 - Style an Element Using Classes

Change the `/var/www/example.com/public/index.html` so that:

- The style tags are converted to CSS definitions in the head of the document.
- The style attributes are replaced with calls to the classes.

Add the following to the head element:

```
<style>
  .header{
    color: #ff9900;
  }

  .img-circle{
    border-radius: 50%;
  }
</style>
```

Update the body of the document as follows:

```
<h1 class="header">Hello, I am YOUR-NAME</h1>

```

## The Title Attribute

In modern browsers, the title attribute provides a tooltip. Hovering your cursor over an element will show the contents of the title attribute.

## Exercise 8 - Add a Tooltip

Change the `/var/www/example.com/public/index.html` so that:

- Hover over the image element MUST show your name in a tooltip.

Add the following attribute to the image element, where YOUR-NAME is *your name*:

```
title="YOUR-NAME"
```

## The ID Attribute

The *id* attribute provides a unique identifier for a given HTML element. No two elements in a single document are permitted to have the same id. We will work with the id attribute in later lessons.

## Event Handler Attributes

Event Handler Attributes (UI Events) allow a user to interact with a web page. These interactions are commonly achieved through the use of an event listener. The W3C specification refers to these as [UI Events](#). A common example is the **click event** which corresponds to the `onclick` attribute. `onclick` serves as an event handler for JavaScript meaning it listens for a click event and executes the attributes value. For example `<a onclick="alert('I was clicked!')">Click Me</a>` would cause

an alert declaring *"I was clicked!"* to be displayed on the screen. When using frameworks such as Angular.js you will see custom attributes such as `ng-click` in which case Angular.js has custom listeners that are designed to listen specifically for the `ng-` prefix. We work with JavaScript in great detail in later lessons. For now, let's apply a quick example.

## Exercise 9 - A Little JavaScript

Change the `/var/www/example.com/public/index.html` so that:

- Clicking a button on the page MUST change the color of the H1 tag to red.

Give the h1 element an id of header.

```
<h1 id="header" class="header">
```

Add a button to the bottom of the page that uses the `onclick` attribute to invoke a line of JavaScript.

```
<button onclick="document.getElementById('header').style='color: #ff0000;'">Click Me</button>
```

## Exercise 10 - Navigation

Change `/var/www/example.com/public/index.html` so that:

- All pages MUST link to each other

Create the following paths with and add the markup as described below.

- `/var/www/example.com/public/resume.html`
- `/var/www/example.com/public/contact.php`

*resume.html*

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Resume</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <h1>My Resume</h1>
  </body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Contact</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <h1>Contact</h1>
  </body>
</html>
```

Commit your changes with the message *Added a basic file structure*.

Create a `nav` element that links all of the pages together. Place this at the top of each of the three web pages.

```
<nav>
  <a href="/">Home</a> |
  <a href="resume.html">Resume</a> |
  <a href="contact.php">Contact</a>
</nav>
```

Commit your changes with the message *Added basic navigation*.

## Lab

Using the *resume.html* file create an HTML version of your resume

## Additional Resources

- [MDN - HTML](#)
- [Introduction to HTML](#)
- [Choosing a Language Tag](#)
- [IANA Language Subtag Registry](#)
- [Why Character Entities](#)
- [XML and HTML Entities](#)
- [The W3C Reference](#)

# CSS

In this section you will learn the basics of

- CSS
- Less
- Sass

# SASS

SASS is a CSS preprocessor is a superset of CSS which means all CSS syntax is considered valid .scss it's a superset because because it extends CSS with programming like capabilities; variables and limited control statements. This is advantageous especially when building large front-end frameworks such [Bootstrap](#) or creating a product with a customizable theme. For example, Bootstrap has a common color for showing danger or errors `#a94442`. If we were to change that color globally we would have to track down every instance of the color and change it manually or we could make a single change to the variable that holds that color.

## Install Sass

since sass is written in ruby we will use gem for the install. Install scss linter, as this is a helpful tool.

```
sudo apt-get install ruby
sudo apt-get install ruby-sass
sudo gem install scss_lint
```

Variables in Sass. Sass denotes variables with a `$` dollar sign. For these lessons we will use the newer SCSS syntax for writing our sass files. These files must have the `.scss` extensions.

## Exercise 1 - Sass Variables

Create the path `~/scss/var.scss` and add the following lines.

```
$font-stack: "Helvetica Neue", Helvetica, Arial, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

```
sass ~/scss/var.scss ~/scss/var.css
```

You will see the following output in the console.

```
body {
  font: 100% "Helvetica Neue", Helvetica, Arial, sans-serif;
  color: #333; }
```

## Exercise 2 - Live Reload / Watch a File

The down side to a preprocessor is the compilation step. This takes time and slows down development. We remedy this by creating a *watcher* this watches a target file for changes and rebuilds it's CSS version in the background. This is one less thing you need to think about which can help keep you in flow. Open a split console window and run the following command in one of the panels.

```
sass --watch ~/scss/var.scss:~/scss/var.css
```

You will see the following output

```
>>> Sass is watching for changes. Press Ctrl-C to stop.
directory ~/scss
write ~/scss/var.css
write ~/scss/var.css.map
```

In the second panel open the scss file in vim, make a change and save it using [esc] then `:x` ; You'll notice a change in the first console window with the following output.

```
>>> Change detected to: var.scss
write ~/scss/var.css
write ~/scss/var.css.map
```

Open the file `~/scss/var.css` and verify your changes.

## Exercise 3 - Implement sass in your project

Move `/var/www/example.com/css/dist/main.css` to `/var/www/example.com/css/src/main.scss`

```
mkdir -p /var/www/example.com/css/src
mv /var/www/example.com/css/dist/main.css /var/www/example.com/css/src/main.scss
```

Then compile the sass file

```
sass /var/www/example.com/css/src/main.scss /var/www/example.com/css/dist/main.css
```

## Mixins

Later we will learn about the Bootstrap framework. Bootstrap is among the most popular frameworks and as such it gets a lot of criticism. One of the those criticisms is the practice of calling multiple class on a single element. The claim is that this can reduce load time. Earlier called multiple classes for the top nav `class="top-nav clearfix"` . The idea here is reusing the clearfix class rather than rewriting it every time we want to use it. Rather than calling two classes we can define a mixin in SASS and reuse it as needed. Now if we need to update our clearfix logic, we can do it in one place and SASS will apply where needed.

## Exercise 4

Create a mix for clearfix by adding the following to the top of `/var/www/about/css/src/main.scss`.

```
/* mixins */
/* clear floats */
@mixin clearfix() {
  &:after {
    content: "";
    display: table;
    clear: both;
  }
}
```

Then change the style declarations for `.clearfix`, `.top-nav` and `#Footer` to the following.

```
.clearfix {
  @include clearfix();
}
```

```
.row() {  
  display: flex;  
  @include clearfix();  
}  
  
nav.top-nav {  
  text-align: center;  
  background: #aaa;  
  @include clearfix();  
}  
  
#Footer {  
  background: #000;  
  color: #fff;  
  padding: 1em;  
  margin: 0;  
  @include clearfix();  
}
```

## Extend/Inheritance

Other examples of calling multiple classes is in the footer navigation as well as the #Content and #Sidebar divs.

```
<ul class="nav-inline pull-right" role="navigation">
```

Another method of reuse in SASS is `@extend` so `.sample{@extend .example;}` would apply the `.example`'s style declaration to `.sample`.

### Exercise 5

Remove the class declaration from the footer navigation element then add the following to the bottom of `/var/www/about/css/src/main.scss`.

```
#Footer ul[role="navigation"] {  
  @extend .nav-inline;  
  @extend .pull-right;  
}
```

Repeat this process for the navigation inside of `.top-nav`.

Update `#Sidebar` and `#Content` to the following.

```
#Sidebar {  
  width: 340px;  
  background: #cdcdcd;  
  @extend .col;  
}  
  
#Content {  
  width: 830px;  
  background: #fff;  
  @extend .col;  
}
```

## Lab - Add Response Classes



Add the following classes to main.scss update the style declarations so that redundant values are called as a variable. Apply these class to error and success messages produced after the form submit in contact.php.

```
.text-success {
  color: #3c763d;
}

.text-error {
  color: #8a6d3b;
}

.text-warning {
  color: #a94442;
}

.message {
  border: 1px solid #ccc;
  border-radius: 4px;
  padding: 10px;
  color: #333;
}

.success {
  @extend .message;
  border-color: #3c763d;
  color: #3c763d;
}

.error {
  @extend .message;
  border-color: #8a6d3b;
  color: #8a6d3b;
}

.warning {
  @extend .message;
  border-color: #a94442;
  color: #a94442;
}
```

[SASS Reference](#)

## Templates Etc.

This section will introduce the student to the idea of a template engine and provide a few final details that should be considered when launching a basic website.

- A basic template engine
- SEO and meta data
- Miscellaneous items for rounding out a website.

# PHP Control Structures

Programming is little more than reading data and piecing together statements that take action on that data. Every language will have its own set of control structures. For most languages a given set of control structures will be almost identical. In the Bash lesson we learned a few control structures most of which exist in PHP. While the syntax may be a little different, the logic remains the same.

## Exercise 3 - If, Else If, Else

Create the following path `/var/www/mtbc/php/if_else.php` and open it Atom. Then add the following lines.

```
<?php

//Initialize your variables
$label = null;
$color = null;

//Check for get parameters
if(!empty($_GET)){
    $color = "#{$_GET['color']}";
}

//Can we name the color by it's hex value
if($color == "#ff0000"){
    $label = "red";
}elseif($color == "#00ff00"){
    $label = "green";
}elseif($color == "#0000ff"){
    $label = "blue";
}else{
    $label = "unknown";
}

//Output the data
echo "<div style=\"color:{$color}\">The color is {$label}</div>";
```

Now open a browser and navigate to [https://localhost/php/if\\_else.php](https://localhost/php/if_else.php) and you will see the message *The color is unknown*. Now add the following string to the end of the URL `?color=ff0000`. Now your message will read *The color is red* and it will be written in red font. That string you added to the end of the URL is known as a [query string](#). A query string allows you to pass arguments into a URL. A query string consists of the query string Identifier (a question mark) `?` and a series of key to value pairs that are separated by an ampersand (`&`). In our example the key is `color` and the value is `ff0000`. If you wanted to submit a query of a first and last name that might look like `?first=bob&last=smith` where first and last are your keys (aka your GET params) bob and smith are your values.

Now let's take a close look at the code. Initializing your variables is a [good practice](#).

```
//Initialize your variables
$label = null;
$color = null;
```

In PHP `$_GET` is a [superglobal](#) so it is always available, this is NOT something you want to try to initialize. `empty()` is used to determine if a variable is [truthy or falsey](#) where falsey values equate to empty or falsey returns true. Prefixing `empty()` with an `!` reverses the return values. In plain English `if(!empty($_GET['color']))` would read *if `$_GET['color']` is not false then do something* or you could say *if `$_GET['color']` has any value then do something*. You will see a lot of curly braces in PHP code due to its use of [C style syntax](#).

If `$_GET['color']` has any value then set the variable `$color` to the value of `$_GET['color']`

```
//Check for get parameters
if(!empty($_GET['color'])){ //This is a control statement
    //This is the body of the statement
    $color = "#{$_GET['color']}";
}
```

The user has submitted a hex value in the form of a get parameter. Do we know what the call that hex value? If the answer is yes set the value of `$label` to that color. Otherwise set the value of `$label` to *Unknown*. Or you could say *if the hex value is red then say it is red; otherwise if it green then say it is green; otherwise if it blue then say it is blue; otherwise say unknown.*

```
//Can we name the color by it's hex value
if($color == "#ff0000"){
    $label = "red";
}elseif($color == "#00ff00"){
    $label = "green";
}elseif($color == "#0000ff"){
    $label = "blue";
}else{
    $label = "unknown";
}
```

Finally we will print some output back to the screen. This time we will wrap the output in some HTML and give it a little style by setting the font color to that of the user input.

```
echo "<div style=\"color:{\$color}\">The color is {\$label}</div>";
```

## Exercise 4 - For Loop

Add the following to the path `/var/www/mtbc/php/for.php`.

```
<?php

$items = array(
    'for',
    'foreach',
    'while',
    'do-while'
);

echo 'PHP Supports ' . count($items) . ' of loops.';

$li = '';
for($i=0; $i<count($items); $i++){
    $li .= "<li>{$items[$i]}</li>";
}

echo "<ul>{$li}</ul>";
```

## Exercise 5 - Foreach Loop

Add the following to the path `/var/www/mtbc/php/foreach.php`.

```
<?php

$items = array(
```

```
'for',
'foreach',
'while',
'do-while'
);

echo 'PHP Supports ' . count($items) . ' of loops.';

$li = '';
foreach($items as $item){
    $li .= "<li>{$item}</li>";
}

echo "<ul>{$li}</ul>";
```

## Exercise 6 - While Loop

```
<?php

$items = [
    'for',
    'foreach',
    'while',
    'do-while'
];

$count = count($items);

echo "PHP Supports {$count} of loops.";

$i = 0;
$li=null;
while ($i < $count) {
    $li .= "<li>{$items[$i]}</li>";
    $i++;
}

echo "<ul>{$li}</ul>";
```

## Exercise 7 - Do While Loop

Add the following to the path `/var/www/mtbc/php/do_while.php`.

```
<?php

$items = [
    'for',
    'foreach',
    'while',
    'do-while'
];

echo 'PHP Supports ' . count($items) . ' of loops.';

$i = 0;
$li=null;
do {
    $li .= "<li>{$items[$i++]}</li>";
} while ($i > 0);
```

## Additional Reading

- [Which Loop](#)

## HTML Forms with PHP Validation

Traditionally, forms have been the most common way to collect data from a user. A form submission is the simplest way to post data to a server. This section will start with a simple POST request and end with complex processing.

Form tags `<form></form>` are used for creating forms in HTML. Every form should have at least two attributes *action* and *method*.

- action - the web address to which the form data will be sent.
- method - the type of request the form should make (probably GET or POST).

### Exercise 1 - Create and Inspect a Contact Form

Create the path `/var/www/example.com/public/contact.php`

`/var/www/example.com/public/contact.php`

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Contact Me - YOUR-NAME</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <h1>Contact YOUR-NAME</h1>
    <form method="post">
      <div>
        <label for="firstName">First Name</label><br>
        <input type="text" name="first_name" id="firstName">
      </div>

      <div>
        <label for="lastName" id="lastName">Last Name</label><br>
        <input type="text" name="last_name">
      </div>

      <div>
        <label for="email" id="email">Email</label><br>
        <input type="text" name="email">
      </div>

      <div>
        <label for="subject" id="subject">Subject</label><br>
        <input type="text" name="subject">
      </div>

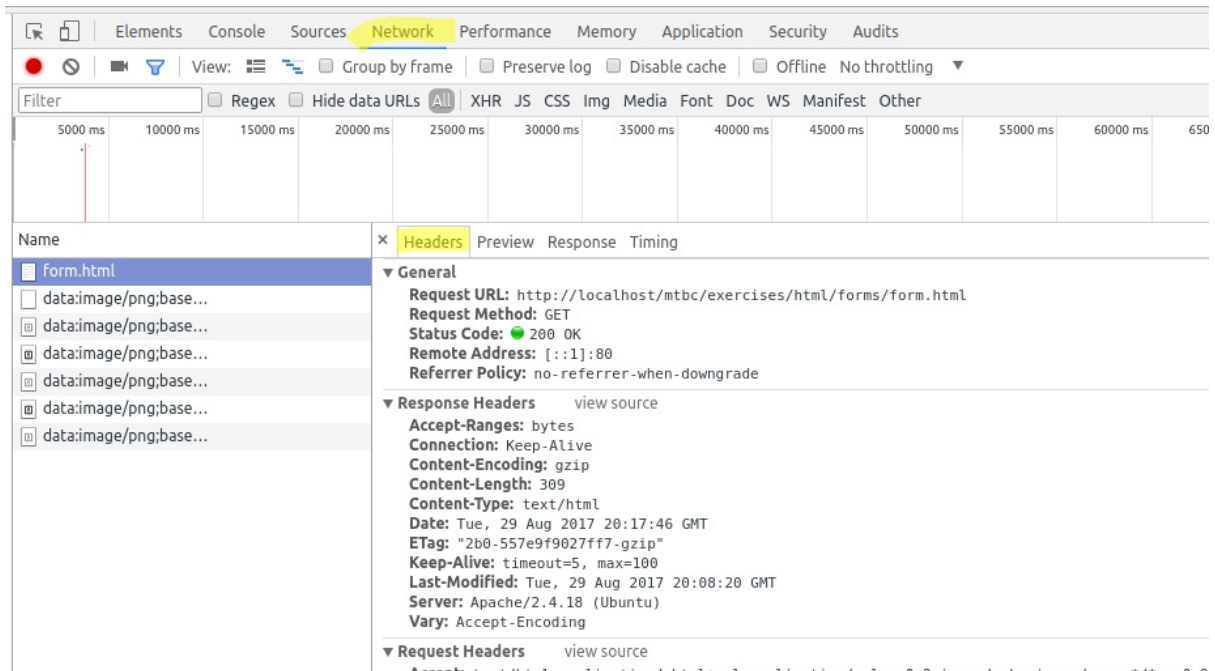
      <div>
        <label for="message" id="message">Message</label><br>
        <textarea name="message"></textarea>
      </div>

      <input type="submit">

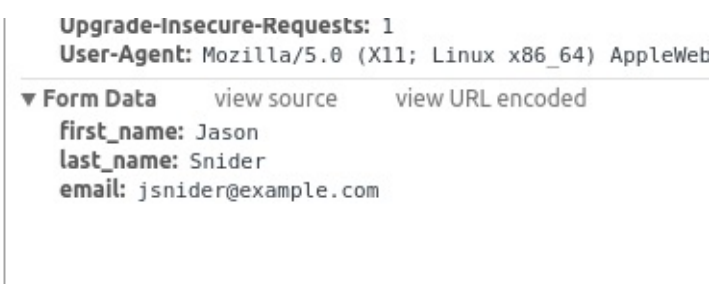
    </form>
  </body>
</html>
```

Now, lets inspect a post request.

- Open the Chrome browser and navigate to <http://loc.example.com>.
- Press the [f12] key to open Chrome's developer tools.
- Click on the network tab.
- Refresh the page.
- Find and click on contact.php
- The headers tab should now be highlighted.



- Fill out the web form and submit the data.
- Once again, find and click on form.html from the network panel.
- Under the headers tab you will see the contents of your web form as key to value pairs. This is how the data will be given to the server.s



Find the opening form tag and set the action attribute as follows. *action="contact.php"*

Add the following to the top of the document, above the DOCTYPE declaration.

```
<?php

$data = $_POST;

foreach($data as $key => $value){
    echo "{$key} = {$value}";
}

?>
```

## Validation and Basic RegEx



Regular Expressions (RegEx) are strings of text that describe a search pattern. You may be familiar with wild cards in which a search for *b\** would return all words that start with the letter b. Now lets say you want your wild card search to still return all words starting with the letter b but only if the word does not contain a number; this is where RegEx comes in `\b(b)+([a-z])*\b` .

- `\b` - a word boundary, the beginning of a word. This would return all words.
- `\b(b)*` - MUST start with the letter b. This would return all words starting with the letter b.
- `\b(b)+([a-z])*` - MAY also contain any lower case letters after the first letter.
- `\b(b)+([a-z])*\b` - Stops each match at the end of a word

[Try It](#)

## Exercise 2 - RegEx

`/var/www/example.com/public/contact.php`

```
<?php
//Create a RegEx pattern to determine the validity of the use submitted email
// - allow up to two strings with dot concatenation any letter, any case any number with _- before the @
// - require @
// - allow up to two strings with dot concatenation any letter, any case any number with - after the at
// - require at least 2 letters and only letters for the domain
$validEmail = "/^[_a-z0-9-]+(\\.[_a-z0-9-]+)*@[a-z0-9-]+(\\.[a-z0-9-]+)*(\\.[a-z]{2,})$/";

//Extract $_POST to a data array
$data = $_POST;

//Create an empty array to hold any error we detect
$errors = [];

foreach($data as $key => $value){
    echo "{$key} = {$value}<br><br>";

    //Use a switch statement to change your behavior based upon the form field
    switch($key){
        case 'email':
            if(preg_match($validEmail, $value)!=1){
                $errors[$key] = "Invalid email";
            }

            break;

        default:
            if(empty($value)){
                $errors[$key] = "Invalid {$key}";
            }
            break;
    }
}

var_dump($errors);
?>
```

RegEx is extremely powerful, flexible and worth learning. Having said that there are a million and one libraries for validating form submissions. I would advise finding a well supported library that meets your projects needs. As of PHP5 [data filters](#) have been natively supported by the language.

**Security Check Point** *Never trust user input. User input is anything come into the server from the client. Even if you have written client side JavaScript to filter out malicious code, the filtered input is still left alone with the client and can be manipulated prior to transit (or even in transit). If it has ever existed outside of the server it CANNOT be trusted.*

## Exercise 3 - Adding a Validation Class

Replace the contents of `/var/www/example.com/public/contact.php` with the following.

Explain the code to the class.

```
<?php

class Validate{

    public $validation = [];

    public $errors = [];

    private $data = [];

    public function notEmpty($value){

        if(!empty($value)){
            return true;
        }

        return false;
    }

    public function email($value){

        if(filter_var($value, FILTER_VALIDATE_EMAIL)){
            return true;
        }

        return false;
    }

    public function check($data){

        $this->data = $data;

        foreach(array_keys($this->validation) as $fieldName){

            $this->rules($fieldName);
        }
    }

    public function rules($field){
        foreach($this->validation[$field] as $rule){
            if($this->{$rule['rule']}($this->data[$field]) === false){
                $this->errors[$field] = $rule;
            }
        }
    }

    public function error($field){
        if(!empty($this->errors[$field])){
            return $this->errors[$field]['message'];
        }

        return false;
    }

    public function userInput($key){
        return (!empty($this->data[$key])?$this->data[$key]:null);
    }
}
```

```

$valid = new Validate();
?>

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Single Page App with a Validation Class</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <nav><a href="/">Home</a> | <a href="contact.php">Contact</a></nav>
    <?php if(empty($valid->errors) && !empty($input)): ?>
      <div>Success!</div>
    <?php else: ?>
      <div>You page has errors.</div>
    <?php endif; ?>

    <form method="post" action="contact.php">

      <div>
        <label for="firstName">First Name</label><br>
        <input type="text" name="first_name" id="firstName">
        <div style="color: #ff0000;"><?php echo $valid->error('first_name'); ?></div>
      </div>

      <div>
        <label for="lastName" id="lastName">Last Name</label><br>
        <input type="text" name="last_name">
        <div style="color: #ff0000;"><?php echo $valid->error('last_name'); ?></div>
      </div>

      <div>
        <label for="email" id="email">Email</label><br>
        <input type="text" name="email">
        <div style="color: #ff0000;"><?php echo $valid->error('email'); ?></div>
      </div>

      <div>
        <label for="subject" id="subject">Subject</label><br>
        <input type="text" name="subject">
        <div style="color: #ff0000;"><?php echo $valid->error('subject'); ?></div>
      </div>

      <div>
        <label for="message" id="message">Message</label><br>
        <textarea name="message"></textarea>
        <div style="color: #ff0000;"><?php echo $valid->error('message'); ?></div>
      </div>

      <input type="submit">

    </form>
  </body>
</html>

```

## Include Files and Namespaces

So far we have a lot of code in the what would typically be considered a *view* (the presentation layer) which should be separated from data and logic as much as possible. In this lesson we will create include and class files that will help us keep our views clean.

**PHP Fig** is a working group aimed at developing interoperability standards for PHP libraries. This will help us build a standardized directory structure.

- [PSR 0](#)
- [PSR 4](#)

## Include Files

An include file is a file that contains a snippet of code that is referenced by another file. In PHP include files can be accessed using [include](#), [include\\_once](#), [require](#) or [require\\_once](#) functions along with a relative or absolute file path.

```
// relative include
include 'util.php';

// absolute include
include '/var/www/example.com/util.php';
```

## Namespace

At the end of the day a name space is simply a way to disambiguate name collisions. Earlier we created a class called `Validate()`. Validation classes are fairly common and let's say you liked specific methods from two different vendors both of who named the class `Validate`, suddenly you have a collision.

Lets say we have two vendors Sally and Bob and I like Sally's email method and Bob's phone method. I want to load this class from both vendors but without a name space the autoloader would not know which class to load into a given object. I might try to include then instantiate but there is no guarantee this will work as classes tend to get cached.

```
include 'vendor/Sally/src/Validation/Validate.php';
$v1 = new Validate();
$v1->Validate->email($email); //This will probably work

include 'vendor/Bob/src/Validate/Validate.php';
$v2 = new Validate();
$v2->Validate->phone($phone); //Sally's version of the class may or may not be cached so the method we want may
or may not be there.
```

With name spaces.

```
// You can probably use an autoloader so you will not have to worry about this.
include 'vendor/Sally/src/Validation/Validate.php';
include 'vendor/Bob/src/Validate/Validate.php';

// The namespace disambiguates class names so $v2's object will have the target class.
$v1 = new \Sally\Validation\Validate();
$v2 = new \Bob\Validate\Validate();

$v1->Validate->email($email);
$v2->Validate->phone($phone);
```

## Exercise 4

Create the path `/var/www/example.com/core/About/src/Validation/Validate.php` and copy the `Validates` class into the file. Add a name space declaration as the first line of the file.

`/var/www/example.com/core/About/src/Validation/Validate.php`

```
<?php

namespace About\Validation;
```

```
class Validate{

    public $validation = [];

    public $errors = [];

    private $data = [];

    public function notEmpty($value){

        if(!empty($value)){
            return true;
        }

        return false;
    }

    public function email($value){

        if(filter_var($value, FILTER_VALIDATE_EMAIL)){
            return true;
        }

        return false;
    }

    public function check($data){

        $this->data = $data;

        foreach(array_keys($this->validation) as $fieldName){

            $this->rules($fieldName);
        }
    }

    public function rules($field){
        foreach($this->validation[$field] as $rule){
            if($this->{$rule['rule']}($this->data[$field]) === false){
                $this->errors[$field] = $rule;
            }
        }

        //Make sure the array is empty if no errors are detected.
        if(count($this->errors) == 0){
            $this->errors = [];
        }
    }

    /**
     * Detects and returns an error message for a given field
     * @param string $field
     * @return mixed
     */
    public function error($field){
        if(!empty($this->errors[$field])){
            return $this->errors[$field]['message'];
        }

        return false;
    }

    /**
     * Returns the user submitted value for a give key
     * @param string $key
     * @return string
    */
}
```

```

    */
    public function userInput($key){
        return (!empty($this->data[$key])?$this->data[$key]:null);
    }
}

```

Since we have pulled the validation logic into a library all we need to do in the contact form is call the class and process it.

`/var/www/example.com/public/contact.php`

```

<?php
//Include non-vendor files
require '../core/About/src/Validation/Validate.php';

//Declare Namespaces
use About\Validation;

//Validate Declarations
$valid = new About\Validation\Validate();

$args = [
    'first_name' => FILTER_SANITIZE_STRING,
    'last_name' => FILTER_SANITIZE_STRING,
    'email' => FILTER_SANITIZE_EMAIL,
    'subject' => FILTER_SANITIZE_EMAIL,
    'message' => FILTER_SANITIZE_EMAIL,
];
$input = filter_input_array(INPUT_POST, $args);

if(!empty($input)){

    $valid->validation = [
        'first_name'=>[[
            'rule'=>'notEmpty',
            'message'=>'Please enter your first name'
        ]],
        'last_name'=>[[
            'rule'=>'notEmpty',
            'message'=>'Please enter your last name'
        ]],
        'email'=>[[
            'rule'=>'email',
            'message'=>'Please enter a valid email'
        ]],
        [
            'rule'=>'notEmpty',
            'message'=>'Please enter an email'
        ],
        'subject'=>[[
            'rule'=>'notEmpty',
            'message'=>'Please enter a subject'
        ]],
        'message'=>[[
            'rule'=>'notEmpty',
            'message'=>'Please add a message'
        ]],
    ];

    $valid->check($input);
}

if(empty($valid->errors) && !empty($input)){
    $message = "<div>Success!</div>";
}else{
    $message = "<div>Error!</div>";
}

?>

```

```
<!DOCTYPE html>
...
```

## Additional Resources

- [PSR-0: Autoloading Standard](#)
- [PHP filter\\_input\\_array\(\)](#)
- [PHP filter\\_var\(\)](#)
- [Email RegEx Examples](#)
- [RegEx 101](#)
- [HTML Purifier](#)
- [Yahoo's HTML Purify](#)
- [Google Caja](#)
- [Sanitize HTML](#)

# JavaScript

As a web developer, I need a solid foundation in JavaScript so I can implement a web sites interactive features.

As a web developer, I need a solid foundation in jQuery because it is the most popular library for DOM manipulation and is often a prerequisite for entry-level web development jobs.

As a web developer, Angular will give me a basic understanding of reactive programming and the observable pattern.

As a web developer, I need a basic understanding of reactive programming and the observable pattern to diversify my skill set.

As a developer, a foundation in node.js will allow me to build across multiple platforms.



## JavaScript Control Structures

Programming is little more than reading data and piecing together statements that take action on that data. Every language will have its own set of control structures. For most languages a given set of control structures will be almost identical. In the Bash and PHP lessons we learned a few control structures most of which exist in JavaScript. While the syntax may be a little different, the logic remains the same.

### Exercise - If, Else If, Else

Create the following path `/var/www/mtbc/js/if_else.html` and open it Atom. Then add the following lines.

```
<script>

//Initialize your variables
var label = null;
var color = null;
var GET = {};

//parse the GET params out of the URL
if(document.location.toString().indexOf('?') !== -1) {
    var query = document.location
        .toString()
        // get the query string
        .replace(/^.*\?\/, '')
        // and remove any existing hash string (thanks, @vrijdenker)
        .replace(/#.*$/, '')
        .split('&');

    for(var i=0, l=query.length; i<l; i++) {
        var aux = decodeURIComponent(query[i]).split('=');
        GET[aux[0]] = aux[1];
    }
}

//Check for get parameters
if(GET['color'] !== 'undefined'){
    color = `#${GET['color']}`;
}

//Can we name the color by it's hex value
if(color == "#ff0000") {
    label = "red";
} else if(color == "#00ff00") {
    label = "green";
} else if(color == "#0000ff") {
    label = "blue";
} else {
    label = "unknown";
}

//Output the dataa
document.write(`<div style="color:${color}">The color is ${label}</div>`);

</script>
```

Now open a browser and navigate to [https://localhost/js/if\\_else.html](https://localhost/js/if_else.html) and you will see the message *The color is unknown*. Now add the following string to the end of the URL `?color=ff0000`. Now your message will read *The color is red* and it will be written in red font. That string you added to the end of the URL is known as a [query string](#). A query string allows you to pass arguments into a URL. A query string consists of the query string Identifier (a question mark) `?` and a series of key to value pairs that are separated

by an ampersand (&). In our example the the *key* is *color* and the *\_value* is *ff0000*. If you wanted to submit a query of a first and last name that might look like *?first=bob&last=smith* where first and last are your keys (aka your GET params) bob and smith are your values.

Now let's take a close look at the code. Initializing your variables is a [good practice](#).

```
//Initialize your variables
var label = null;
var color = null;
```

JavaScript does not have a `$_GET` super global like PHP so we will build one by parsing out the URL

```
//parse the GET params out of the URL
if(document.location.toString().indexOf('?') !== -1) {
    var query = document.location
        .toString()
        // get the query string
        .replace(/^\.*?\?/, '')
        // and remove any existing hash string (thanks, @vrijdenker)
        .replace(/#.*$/, '')
        .split('&');

    for(var i=0, l=query.length; i<l; i++) {
        var aux = decodeURIComponent(query[i]).split('=');
        GET[aux[0]] = aux[1];
    }
}
```

*If GET['color'] is defined then the set the variable color to the value of GET['color']*

```
//Check for get parameters
if(!empty($_GET['color'])){ //This is a control statement
    //This is the body of the statement
    color = `#${GET['color']}`; //ES^ String literal
}
```

The user has submitted a hex value in the form of a get parameter. Do we know what to the call that hex value? If the answer is yes set the value of *label* to that color. Otherwise set the value of *label* to *Unknown*. Or you could say *if the hex value is red then say it is red; otherwise if it green then say it is green; otherwise if it blue then say it is blue; otherwise say unknown*.

```
//Can we name the color by it's hex value
if(color == "#ff0000") {
    label = "red";
} else if(color == "#00ff00") {
    label = "green";
} else if(color == "#0000ff") {
    label = "blue";
} else {
    label = "unknown";
}
```

Finally we will print some output back to the screen. This time we will wrap the output in some HTML and give it a little style by setting the font color to that of the user input.

```
document.write(`<div style="color:${color}">The color is ${label}</div>`);
```

## Exercise - For Loop

Add the following to the path `/var/www/mtbc/js/for.html`.

```
<script>
var items = [
    'for',
    'do...while',
    'for...in',
    'for...of'
];

var msg = 'JavaScript supports ' + items.length + ' types of loop.';

var li = '';
for(i=0; i<items.length; i++){
    li += '<li>${items[i]}</li>';
}

msg += '<ul>${li}</ul>';

document.write(msg);
</script>
```

## Exercise - For...in Loop

Add the following to the path `/var/www/mtbc/js/forin.html`.

```
<script>
var items = {
    0:'for',
    1:'do...while',
    2:'for...in',
    3:'for...of'
};

var msg = 'JavaScript supports ' + items.length + ' types of loop.';

var li = '';
for(var item in items){
    li += '<li>${items[item]}</li>';
}

msg += '<ul>${li}</ul>';

document.write(msg);
</script>
```

## Exercise - For...of Loop

Add the following to the path `/var/www/mtbc/js/forof.html`.

```
<script>
var items = [
    'for',
    'do...while',
    'for...in',
    'for...of'
];

var msg = 'JavaScript supports ' + items.length + ' types of loop.';

var li = '';
```

```
for(var item of items){
    li += `<li>${item}</li>`;
}

msg += `<ul>${li}</ul>`;

document.write(msg);
</script>
```

## Exercise - While Loop

```
<script>
var items = [
    'for',
    'do...while',
    'for...in',
    'for...of'
];

var msg = 'JavaScript supports ' + items.length + ' types of loop.';
var i=0;
var li = '';
while(i < items.length){
    li += `<li>${items[i]}</li>`;
    i++;
}

msg += `<ul>${li}</ul>`;

document.write(msg);
</script>
```

## Exercise - Do...While

Add the following to the path `/var/www/mtbc/js/do_while.php`.

```
<script>
var items = [
    'for',
    'do...while statement',
    'labeled statement',
    'break statement',
    'continue statement',
    'for...in statement',
    'for...of statement'
];

var msg = 'JavaScript supports ' + items.length + ' types of loop.';
var i=0;
var li = '';
do {
    li += `<li>${items[i]}</li>`;
    i++;
} while (i < items.length)

msg += `<ul>${li}</ul>`;

document.write(msg);
</script>
```

## Exercise - Switch Statement

```
<script>

var color = null;
var GET = {};

//parse the GET params out of the URL
if(document.location.toString().indexOf('?') !== -1) {
    var query = document.location
        .toString()
        // get the query string
        .replace(/^\.*?\?/, '')
        // and remove any existing hash string (thanks, @vrijdenker)
        .replace(/#.*$/, '')
        .split('&');

    for(var i=0, l=query.length; i<l; i++) {
        var aux = decodeURIComponent(query[i]).split('=');
        GET[aux[0]] = aux[1];
    }
}

//Check for get parameters
if(GET['color'] !== 'undefined'){
    color = `#${GET['color']}`;
}

switch (color) {
    case 'ff9900':
        console.log('The color is red');
        break;
    case '00ff00':
        console.log('The color is green');
        break;
    case '0000ff':
        console.log('The color is blue');
        break;
    default:
        console.log('Sorry, I cannot determine the color');
}

</script>
```

## Additional Resources

- [Control Flow and Error Handling](#)
- [Loops and iteration](#)
- [Switch](#)

# Walking the DOM

The Document Object Model (DOM) is an API that treats markup languages (xml, xhtml, html, ect) as a tree structures. A easier way to think of this might be as an interface that allows a programmer to access tags and the attributes of tags. Later we will learn about jQuery; a library for querying the DOM (among other things). First we will learn basic manipulation using straight JavaScript.

In the previous lesson we used `document.getElementById()`; this method queries the DOM for an element with a matching id. There are many similar methods.

## Collection Live vs Static (not live)

A collection is an object that represents a lists of DOM nodes. A collection can be either live or static. Unless otherwise stated, a collection must be live.<sup>1</sup>

If a collection is live, then the attributes and methods on that object must operate on the actual underlying data, not a snapshot of the data.<sup>1</sup>

When a collection is created, a filter and a root are associated with it.<sup>1</sup>

The collection then represents a view of the subtree rooted at the collection's root, containing only nodes that match the given filter. The view is linear. In the absence of specific requirements to the contrary, the nodes within the collection must be sorted in tree order.<sup>1</sup>

</blockquote>

## By ID

`document.getElementById(id_string)`

Return a element object.

Create the path `mtbc/js/dom/by_id.html`, then open the source file in your editor. Add the following lines to the script tags then refresh the page and note the changes.

```
var elem = document.getElementById("a"); // Get the elements
elem.style = 'color: #FF0000;'; // change color to red
```

## By Tag

`document.getElementsByTagName(tag_name)`

Return a live HTMLCollection (an array of matching elements).

The tag\_name is "div", "span", "p", etc. Navigate to `mtbc/js/dom/by_tag_name.html`, then open the source file in your editor. Add the following lines to the script tags then refresh the page and note the changes.

```
var list = document.getElementsByTagName('blockquote'); // get all p elements
list[0].style = 'color: #FF0000;';
```

## By Class

*document.getElementsByClassName("class\_values")*

Return a live HTMLCollection.

The *class\_values* can be multiple classes separated by space. For example: "a b" and it'll get elements, where each element is in both class "a" and "b". Navigate to *mtbc/js/dom/by\_class\_name.html*, then open the source file in your editor. Add the following lines to the script tags then refresh the page and note the changes.

```
// get all elements of class a
var list = document.getElementsByClassName('b');

// Make it red
list[0].style = 'color: #FF0000;';

// get all elements of class a b
var list2 = document.getElementsByClassName('a b');

//Make them bold and apply the color from list[0]
list2[0].style = 'font-weight: bold; color:' + list[0].style.color;
```

## By Name

*document.getElementsByName("name\_value")* Return a live HTMLCollection, of all elements that have the name="name\_value" attribute and value pair.

Navigate to *mtbc/js/dom/dom/by\_name.html*, then open the source file in your editor. Add the following lines to the script tags then refresh the page and note the changes.

```
//Get all elements with a name of a
var list = document.getElementsByName('a');

//Loop through all of the matching elements
for (var i=0; i<list.length; i++) {
    //Make them red
    list[i].style = 'color: #FF0000;';
}
```

## By CSS Selector

*document.querySelector(css\_selector)*

Return a non-live HTMLCollection, of the first element that match the CSS selector *css\_selector*. The *css\_selector* is a string of CSS syntax, and can be several selectors separated by comma.

Navigate to *mtbc/js/dom/by\_css\_selector.html*, then open the source file in your editor. Add the following lines to the script tags then refresh the page and note the changes.

```
//Find the element with an id of bold
var bold = document.querySelector('#bold');

//Make it bold
bold.style = 'font-weight: bold;';

//Find all elements with a class of blue
var blues = document.querySelectorAll('.blue');
```

```
//Loop through all of the matching elements
for (var i=0; i<blues.length; i++) {
  //Make them blue
  blues[i].style = 'color: #0000FF;';
}
```

## Additional Resources

- <sup>1</sup>[Collections](#)

### Udemy

[Javascript Intermediate level 1 - Mastering the DOM](#)





## Toolkits

Utilizing a toolkit can reduce the time it takes to get your product to market. Toolkits often take care of the repetitive tasks that while necessary change very little from project to project. These often employ "best" or "common" practices. While we will discuss a few toolkits our focus will be on only one; [Bootstrap](#).

### HTML Boilerplate

I would describe HTML5 Boilerplate as a collection of good practices, especially for those using Apache and it comes with a nice .htaccess file that explains each setting in great detail.

### Material Design Light

Material Design is Google's take on UI.

### Bootstrap

A responsive, mobile first front end library.

# Yarn and Gulp

Like [NPM](#), [Yarn](#) is a package manager for NodeJS. It is similar [Composer](#) for PHP, [PIP](#) for Python or [Gradle](#) for Java.

```
sudo npm install yarn --global
cd /var/www/bootstrap
yarn init
```

[Gulp](#) is a toolkit for automating workflows and tasks.

```
sudo npm install --global gulp-cli
npm install --save-dev gulp@next
```

Create the file *gulpfile.js* and add the following.

```
var gulp = require('gulp');

gulp.task('default', defaultTask);

function defaultTask(done) {
  // place code for your default task here
  done();
}
```

## Additional Resources

- [Yarn](#)
- [Gulp](#)



# SQL

Structure Query Language (SQL) is language designed specifically for working with databases. SQL is an open standard maintained by the American National Standards Institute (ANSI). Despite being an actively maintained open standard you'll most likely a lot of time working with vendor specific variants. These differences are typically subtle and quick Google search will typically get you around them. For instance if you're used to working with Oracle's `TO_DATE()` and now you're working with MySQL; `TO_DATE()` will not work. Google something like *TO\_DATE in mysql* and one of the first results will likely point to MySQL's `STR_TO_DATE()` method. Point being, learning any vendor variant of SQL will be enough to allow you to work with just about any vendor variant. The same Google tricks can even be used for NoSQL databases which we will learn about later. Knowledge of SQL is desired in many industries outside of tech as it is considered to be the *English* of the data world. Understanding SQL helps you understand other data paradigms such as *noSQL*.

## Database

Not to be confused with a Database Management System (DBMS) is a container. This may be a file or a set files, it really doesn't matter as you will likely never see the data. You will be working exclusively with the DBMS. The DBMS is the software that works with the database. For example products such as Oracle, MySQL, SQL Server and even MongoDB are all database management systems. The first three are also known as a Relation Database management System (RDBMS) while the latter is a NoSQL DBMS.

## Schema

Schema is defined as *a representation of a plan or theory in the form of an outline or model*. Unfortunately, the terms database and schema are often used interchangeably. For our purposes *schema* will be used to describe a database. This may be a visualization or the actual SQL file that contains the build commands for the database.

## Tables

A table is a structured list of data typically designed to represent a collection like items. These collections may be users, customers, products, etc. Naming your tables, especially in larger project can prove difficult; defining a [style guide](#) and sticking to it can save you some headaches in the long run. One common rule is for all tables to end in a plural form of the entity it represents.

## Columns

A column is often referred to as a field. A column has several attributes a name, data type, default value and indices are typically what you'll be concerned with. The data type itself may have additional parameters such as length.

Column names should be short and to the point. While style guides vary typical rules state a column name should differ from that of the table and must not overlap with any [reserved words](#).

[Data types](#) cast restraints on a column. For example a column of type integer (INT) would not allow any non-numeric characters. A `VARCAHR(10)` would allow any combination of characters but will truncate the string after 10 characters.

Default values define what is to be entered if no value is present. This may be null or not null (which will force a value) or any other value compatible with the data type.

[Indices](#) are used to improve performance. These take specific columns from a table or tables and stores them in a way that allows them to be looked up quickly with out needing to reanalyze all of the content of all tables involved in the query. I have seen good good indexing literally cut minutes of page load times. While not required all tables should have a primary key.

A primary key is unique index for a row of data. The most common primary key is a simple id. This can be an [auto-incrementing](#) number, a universally unique identifier ([UUID](#)) or any other bit of unique data such an email address.

### Security Checkpoint

*I have seen a number of systems that ask for sensitive data such as social security or employer identification numbers simply because that is the only way they could think of not to risk duplicate data. Think twice before doing this, if your business does not absolutely require this sort of information do not even think about storing it. If it is required, please consult a security professional.*

## Additional Resources

- [NoSQL Not Only SQL](#)
- [Naming Conventions: Stack Overflow Discussion](#)
- [Integer Types](#)

#

<https://microtrain.udemy.com/sql-for-beginners-course/>

Next: MySQL

# MySQL

[MySQL](#) is a free and open source [relational database management system](#) (RDBMS) currently under the control of Oracle. An RDBMS will store data in a container called a table. This data is organized by rows and columns similar to a spreadsheet (aka tabular data). Relationships are joined by creating foreign key relationships between rows of data across multiple tables. MySQL uses Structured Query Language (SQL) to retrieve data from tables.

Much like SQL Server, Oracle or PostgreSQL, MySQL is client-server software. This means the database and database management software is running on a server. Even if you're running a local instance of MySQL, you're running a server. The client is the software you interact with. When you interact with the software, the software makes a request to the server on your behalf. Some common clients are the [MySQL CLI](#), [phpMyAdmin](#) and [MySQL Workbench](#). Even programming languages act as clients; for example PHP uses the [PDO library](#) to interact with a DBMS. [ORMs](#) are another type of client package; an example of this would be [Doctrine](#) for PHP.

## Core Concepts of an RDBMS

### Table Structure

A database is made up of tables which are defined by columns, rows, data types and values. Conceptually you can think of a table in the same way you think of a spreadsheet.

My customers table might look like the following.

COLUMNS				
id	firstname	lastname	dob	
1	sally	smith	1977-01-22	R
2	frank	brown	1951-04-01	W
3	bob	gray	2004-08-17	S

### Keys and Indices

#### Index

An index provides a faster means of lookup for an SQL query. A good rule of thumb is to create an index for every column that appears in a WHERE clause. You should not create an index for a column that is not used for looking up data.

#### Primary Key (Unique Index)

A primary key in MySQL is both an index and a unique identifier for a given column. Typically an auto-incrementing integer or a [UUID](#). In the above example the id column would be my primary key.

#### Foreign Key (Index)

This is a column that links one table to another. For example if I have an address table that I want to link to my customers table I would add a column called `customer_id` and the value of this column would match an id in the customers table. In this case `customer_id` is said to be a `foreign_key`.

## Unique Keys (Unique Index)

This is an indexed column that requires a unique value for every row in the table.

## Composite Keys (Unique Index)

This is an indexed key pair that can uniquely identify a row in the table.

## FullText Index (Unique Index)

This is a special type of index that allows various configurations string searches against rows containing large amounts of text.

## Relationships

### One-to-one

Table A may (or must) have one and only one corresponding rows in table B.

### One-to-many

Table A may (or must) have one or more corresponding rows in table B.

### Many-to-many

Table A may (or must) have one or more corresponding rows in table B and table B may (or must) have one or more corresponding rows in table A. A many-to-many relationship must be resolved by an associative entity. An associative entity is table that links two or more tables together using foreign keys.

## Additional Resources

- [PDO Library](#)
- [MySQL for NodeJS](#)
- [MySQL Client Programs](#)
- [ORM Is an Offensive Anti-Pattern](#)
- [To ORM or Not to ORM](#)
- [ORM Hate](#)

[Next: Working with MySQL](#)



## Data Models

[https://en.wikipedia.org/wiki/First\\_normal\\_form](https://en.wikipedia.org/wiki/First_normal_form) Customer data with one or many phone numbers stored in a single row.

customers			
id	first_name	last_name	phone
123	John	Smith	(555) 861-2025, (555) 122-1111
456	Jane	Doe	(555) 403-1659, (555) 929-2929
789	Cindy	Who	(555) 808-9633

```
SELECT phone FROM customers WHERE id = 123
```

1 result (555) 861-2025, (555) 122-1111



# CakePHP

CakePHP is an MVC (Model, View, Controller) based rapid application development (RAD) framework built using PHP. CakePHP has a solid eco-system and is designed around test driven development (TDD).

## MVC

MVC is a software design pattern that splits your application into three distinct layers: data (Model), Business Logic (Controller), and presentation (View).

### Model

A model can be anything that provides data such as a table in a database, a reference to an API, a spreadsheet, etc. For our user a model will reference a table in a database.

### View

Views are the presentation layer. Views will typically be HTML but can be anything a client can read such as .html, .json, .xml, .pdf or even a header that only a machine can access.

### Controller

## CRUD

## Migrations

-->

## Installation

First make sure you have installed internationalization functions for PHP.

```
sudo apt-get install php-intl
```

Create a CakePHP project via composer. Sticking with the *example.com* nomenclature we will call this one *cake.example.com*.

```
cd /var/www
composer create-project --prefer-dist cakephp/app cake.example.com
```

Answer yes to the following

```
Set Folder Permissions ? (Default to Y) [Y,n]?
```

## Your First App

---

Move into the new project folder

```
cd cake.example.com
```

Spin up a development web server.

```
bin/cake server
```

and in a browser go to <http://localhost:8765/>. You will be presented a default home page that shows that gives you plenty of resources to help you learn CakePHP and it will return a system status that makes sure your system is set up correctly. Everything should be green except for the database.



Please be aware that this page will not be shown if you turn off debug mode unless you replace `src/Template/Pages/home.ctp` with your own version.

## Environment

- Your version of PHP is 5.6.0 or higher (detected 7.0.22-0ubuntu0.16.04.1).
- Your version of PHP has the mbstring extension loaded.
- Your version of PHP has the openssl extension loaded.
- Your version of PHP has the intl extension loaded.

## Filesystem

- Your tmp directory is writable.
- Your logs directory is writable.
- The *FileEngine* is being used for core caching. To change the config edit `config/app.php`

## Database

- CakePHP is NOT able to connect to the database.  
Connection to database could not be established: SQLSTATE[HY000] [1045] Access denied for user 'my\_app'@'localhost' (using password: YES)

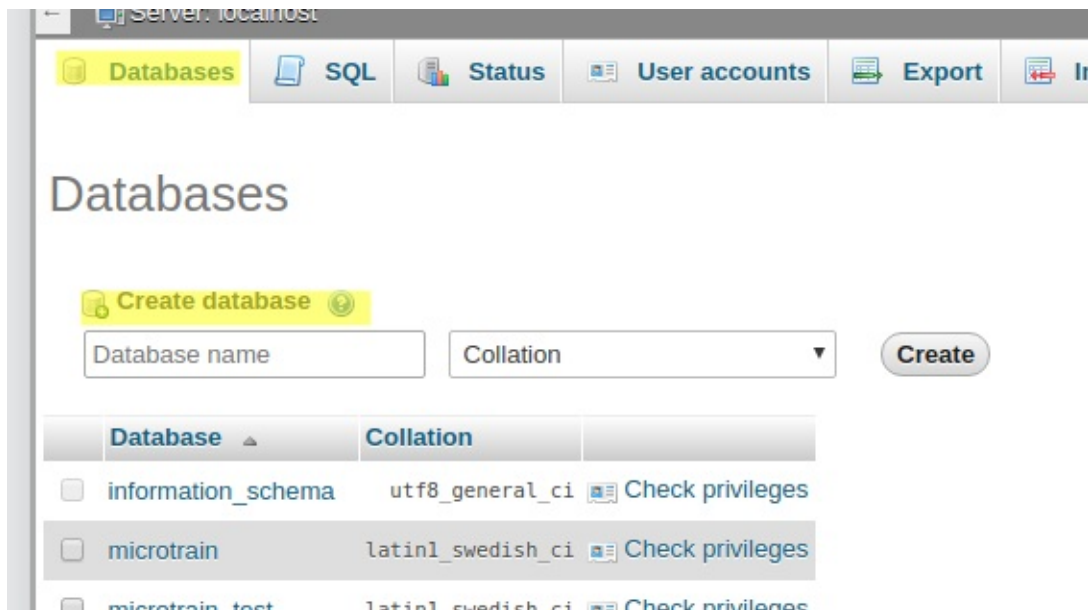
## DebugKit

- DebugKit is loaded.

Add `cake.example.com` as a project in Atom. Navigate to `config/app.php` this is the default configuration file for your application. CakePHP stores its configuration as an array, find the *Datasources* attribute somewhere around the line 220 (you can use the shortcut [ctrl] + [g] and enter 220). You will notice two child attributes *default* and *test*. *default* holds the configuration for your application's database while *test* holds the configuration for running unit tests.

## Setup Your Database

Go to <http://localhost/phpmyadmin> and login as with `root:password`. Find the Databases tab and under the *Create database* header enter `cake_app` as your first database. This will now ask you to create a table, skip this step and find your way back to the Databases tab and create another database called `cake_test` you can now close out of phpMyAdmin and return to the `app.php` file in Atom.



Update the database configurations as follows.

*default*

```
'username' => 'root',
'password' => 'password',
'database' => 'cake_app',
```

*test*

```
'username' => 'root',
'password' => 'password',
'database' => 'cake_test',
```

Return to <http://localhost:8765/>(<http://localhost:8765/>) and refresh the page, all settings should now be green.

Let's set up an Apache configuration with a local hosts entry for development purposes.

```
sudo vim /etc/apache2/sites-available/cake.example.com.conf
```

```
<VirtualHost 127.0.0.32:80>

    ServerName loc.cake.example.com

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/cake.example.com/webroot

    # Allow an .htaccess file to ser site directives.
    <Directory /var/www/cake.example.com/webroot/>
        AllowOverride All
    </Directory>

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
```

```
</VirtualHost>
```

Add the following to `/etc/hosts`

```
127.0.0.32      loc.cake.example.com
```

and finally load the new site.

```
sudo a2ensite cake.example.com && sudo service apache2 restart
```

Should you encounter any write permission issues

```
sudo chown www-data:jason logs
sudo chown www-data:jason logs/*
sudo chown www-data:jason tmp
sudo chown www-data:jason tmp/*
```

@todo navigate to src, explain the directory structure Model, Views and Controller @todo callback methods and lifecycles as it pertains a CakePHP and Programming in general.

## Blog Tutorial

We will start by building an Articles CRUD based on CakePHP's [CMS tutorial](#). Then we will use Composer to install a user [Authentication plugin](#). Then we will then tie Users to Articles (a blog post).

- Login to phpMyAdmin
- Click into cake > cake\_app from the side bar
- Click on the SQL tab
- Copy and Paste the following the text area and hit submit
- Repeat this process for cake > cake\_test

```
/* First, create our articles table: */
CREATE TABLE articles (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  title VARCHAR(255) NOT NULL,
  slug VARCHAR(191) NOT NULL,
  body TEXT,
  published BOOLEAN DEFAULT FALSE,
  created DATETIME,
  modified DATETIME,
  created DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT 'When the post was created',
  modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT 'When the post was last edited',
  UNIQUE KEY (slug)
) ENGINE=INNODB;

/* Then insert some articles for testing: */
INSERT INTO articles (title,body,created)
VALUES ('The title', 'This is the article body.', NOW());
INSERT INTO articles (title,body,created)
VALUES ('A title once again', 'And the article body follows.', NOW());
INSERT INTO articles (title,body,created)
VALUES ('Title strikes back', 'This is really exciting! Not.', NOW());
```

Since we have the table in our database we can automate the build by *baking* the model. Run the following command and take note of what files get created.

```
bin/cake bake model Articles
```

You'll notice fixtures and tests are created, this provide a placeholder for building unit tests.

Navigate to the `src/Model` and check out the files in *Entity* and *Article* folders.

Complete the [articles tutorial](#). Do not move on to the Tags and Users tutorial.

## Users

1. We will install the users plugin developed by CakeDC. The documentation is available [here](#).
2. Install the core by running.

```
cd /var/www/cake.example.com
composer require cakedc/users
```

3. Use [the migrations plugin](#) to install the required tables.

```
bin/cake migrations migrate -p CakeDC/Users
```

1. Add the following line to the end of `config/bootstrap.php`, this bootstraps the plugin to application start up.

```
Plugin::load('CakeDC/Users', ['routes' => true, 'bootstrap' => true]);
```

## Tie Users to Articles

Add a column called `user_id` to the articles table and create a foreign key relationship to the users table.

```
ALTER TABLE articles ADD user_id INT UNSIGNED NOT NULL DEFAULT 0;
ALTER TABLE articles ADD CONSTRAINT user_id FOREIGN KEY (user_id) REFERENCES users(id);
```

## Configuration

Navigate to <http://loc.cake.example.com/users/users> and create a user account.

## Labs

### Lab 1 - Composer

Using the documentation for the users plugin add the ability to login using a social media platform of your choice.

### Lab 2 - Comment System

1. Create a table
  - `id` - the primary key of the comment system
  - `article_id` - the id of the article for which the comment is being made
  - `first_name` - the first name of the reader making a comment

- last\_name - the last name of the reader making a comment
  - email - the email of the reader making a comment
  - comment - the readers comment
  - created - current time stamp at the time of submission
2. At the bottom of each article provide a form that will collect the above data and on submit
    - Save the data to the comments table
    - Use the MailGun API to send your self an email telling you someone has commented on your article.

## Lab 3 - Contact Form

1. Create a contact form.
2. When a user submits the form, save the contents to a database.
3. When the user submits the form, use the MailGun API to send your self an email every time someone submits the contact form.

## Additional Resources

- [CakePHP](#)



## Introduction

MongoDB is a distributed noSQL noSchema document database designed for scalability, high availability, and high performance.

# MongoDB

MongoDB is a distributed noSQL noSchema document database designed for scalability, high availability, and high performance.

## Install MongoDB

### Install MongoDB on Ubuntu 16.04

By default the Ubuntu package manager does not know about the MongoDB repository so you'll need to add it to your system. First add MongoDB's private key to the package manager. Then, update the repository list. Finally, reload the package database.

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5

echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.6 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.6.list

sudo apt-get update
```

Now you can install the latest stable release of MongoDB.

```
sudo apt-get install -y mongodb-org
```

## Working with MongoDB

Start the database.

```
sudo service mongod start
```

Access the database by typing `mongo` at the command line.

```
mongo
```

Create a database called `cms`.

```
use cms
```

Show a list of all databases.

```
show Databases
```

You will not see your new database listed until you insert at least one document into it. This will create a collection called `users`.

```
db.users.insert({"email":"youremail@example.com","firstname":"YourFirstName","lastname":"YourLastName"})
```

Let's break that last command down.

`db` this calls the database object.

`insert()` is a method of the collection object, this inserts a record into a target collection. The JSON string contains the key/value pairs that will be inserted into the document. Documents in MongoDB are JSON strings.

`users` this is the collection upon which collection methods are called. If a collection does not exist, calling `insert` against the collection will create the collection.

Now if you run `sh show databases` you will see `cms` in the list. Next, you will want to look up your list of collections.

```
show collections
```

Now we will create some new documents in the `users` collection.

```
db.users.insert({"email":"sally@example.com", "firstname":"Sally", "lastname":"Smith"})
db.users.insert({"email":"bob@example.com", "firstname":"Bob", "lastname":"Smith"})
```

Now we will return all of the documents in the database by calling the `find` method (will no arguments) against the `users` collection of the database.

```
db.users.find()
```

We can look up specific documents by building a JSON string.

```
db.users.find({"email":"sally@example.com"})
```

RegEx allows you to search partial strings like every user who's last name ends in *th*.

```
db.users.find({"lastname": /. *th/})
```

Update a document by calling `update()` with the update modifier `$set` against a target collection and passing in the `_id` object. Passing the same JSON string into the `save` method of a collection would completely replace the document.

Update only, using the update modifier.

```
db.users.update({ "_id" : ObjectId(PASTE TARGET ID HERE)}, {$set:{"email":"new@email.com"}})
```

Full document replacement by omitting the update modifier

```
db.users.update({ "_id" : ObjectId(PASTE TARGET ID HERE)}, {"email":"new@email.com"})
```

The same JSON string you build for looking up documents can be used for deletion by passing that string into the `remove` method of collection.

```
db.users.find()
db.users.remove({"email":"bob@example.com"})
db.users.find()
```

Use `drop()` to remove an entire collection;

```
show collections
use cms
db.users.drop()
show collections
```

You can drop the entire database by running *dropDatabase()* directly against the *db* object.

```
use cms
db.dropDatabase()
```

## Atlas

1. Create an Atlas Account
2. Explain the IP Addresses

## Compass

1. Install Compass

## Additional Resources

- [Why You Should Never Use MongoDB](#) - Read the article then dig into the comments.

Express is a web application framework for NodeJS.

## Introduction

Angular is a toolkit that makes it easy to build web, mobile, or the desktop applications with using web technology. Angular combines TypeScript, design patterns and auto bundling (think automated gulp) to compile, transpile and package front end assets in realtime. Angular is a CLI based application that runs a dev server on port 4200. Once your satisfied with your project it can be packaged into a distribution package consisting of HTML, CSS and JavaScript.

In this unit we will

- Learn the basics of TypeScript.
- Complete Angular's Tour of Heroes (TOH) tutorial.
- Add a TOH API to an existing stack.
- Modify the TOH app to work with our API.
- Apply Angular to our existing ExpressJS app by adding
  - An authentication app
  - An app for managing blog posts (aka a CMS)
  - A app for managing users

### Install Cordova

- Convert the Nasa App to mobile.
- Install Ionic
- Build the users App

# Apache Cordova

[Apache Cordova](#) is a free and open source environment that allows us to use web technologies to target multiple platforms with a single code base. This is used primarily for mobile development. While we are able to write the code using web technologies, we still need access to that platforms build environment. Cordova provides a mechanism to compile technology into native application code but it still needs the native environment to build and test that package. If you want to compile your application into an iOS app you will still need access to a MAC running the latest version of xCode. Android on the other hand can build on Windows, MAC or Linux as a result we will focus our build on Android. For the most part, anything you build in Cordova for Android should run on iOS save for the occasional tweaks.

[Android Studio](#) is the official IDE (Integrated Dev Environment) for building Android applications, this provides everything you will need to build Android applications. There are however a few dependencies.

In this lesson we will install

- Apache Cordova
- Java
- Gradle
- Android studio
- A few 32 bit binaries.

## Install Apache Cordova

Cordova is built on top of Node; we will use npm to do a global install.

```
sudo npm install -g cordova
```

## Install the Java SDK

Android runs on top of Java (and Java compatible APIs) we will need to install Java so we can compile our web based build into Java. We will use Oracle's JDK for this (there are rumors that Google will switch future build to Open-JDK).

Start by adding Oracle's PPA.

```
sudo add-apt-repository ppa:webupd8team/java
```

Update your apt repos list

```
sudo apt-get update
```

Install the JDK

```
sudo apt-get install oracle-java8-installer
```

Choose the desired installation Run the command `sudo update-alternatives --config java` and chose from the resulting menu, which should be similar to the following. In this case, I selected option 0 *auto mode*

Selection	Path	Priority	Status
-----			



0	/usr/lib/jvm/java-8-oracle/jre/bin/java	1081	auto mode
* 1	/usr/lib/jvm/java-8-oracle/jre/bin/java	1081	manual mode

You will need to set your `JAVA_HOME` Environment Variable (so that running programs can find Java). To do this you will need to find your Java path; do this with the following command (the result of which will contain your Java path).

```
sudo update-alternatives --config java
```

Now set the path using your favorite editor. In my case the path is at `/usr/lib/jvm/java-8-oracle/jre/bin/java` so I will add this line `JAVA_HOME="/usr/lib/jvm/java-8-oracle/jre/bin/java"` to the environment file.

```
sudo vim /etc/environment
```

Once you have added the that line, you will want to reload the file.

```
source /etc/environment
```

## Install Gradle

In short Gradle is a the build system used by Android. Stack Overflow has a [more detailed answer](#). You can install this using Apt, but the Ubuntu repos are a little behind on this one, so it's better to install it manually.

## Download and Unpack Gradle

```
cd ~/Downloads
wget https://services.gradle.org/distributions/gradle-4.5-bin.zip
sudo mkdir /opt/gradle
sudo unzip -d /opt/gradle gradle-4.5-bin.zip
```

## Add an Environmental Variable on Startup

Open the *environment* file

```
sudo vim /etc/environment
```

add the following lines, the first is for Gradle, the others you will need later so add them now. Replace *YOUR\_USER\_NAME* with the user name you use to login to your system.

```
export PATH=$PATH:/opt/gradle/gradle-4.5/bin
export ANDROID_HOME=/home/YOUR_USER_NAME/Android/Sdk
export PATH=${PATH}:${ANDROID_HOME}/tools:${ANDROID_HOME}/platform-tools
```

## Restart environment

```
source /etc/environment
```

Install additional 32 bit libraries

```
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1 libbz2-1.0:i386
```

## Install the Android SDK

If you were to build an Android application from scratch, this is what you would use. Cordova needs access to the SDK for it's builds and we need access to the emulators. You will use Cordova to write the code, Cordova will use the Android SDK to build your applications and you will use Android Studio to build your emulators.

### Download Android Studio

```
cd ~/Downloads
sudo unzip android-studio-ide-*-linux.zip -d /usr/local
cd /usr/local/android-studio/bin
./studio.sh
```

Follow the prompts and keep choosing next.

\_ At some point you will be asked to create or import a new project. This is required but we will not use it. When prompted to do so, go ahead and create a project. Create a new project called Hello World

- Choose create with no activity \_

You should now have a running instance of Android Studio. Add a desktop entry *Tools > Create desktop entry...* and lock Android Studio to your launcher.

Now it's time to create an (AVD (Android Virtual Device))(<https://developer.android.com/studio/run/managing-avds.html>)

Tools > Android > AVD Manager Click the *Create Virtual Device* button Choose Nexus 5x Click on the *x86 images* tab and choose *Nougat 25 x86\_64* (Download if required) Choose the default options from the AVD screen and click *Finish* From the *Your Virtual Devices* dialog click the green arrow beside our new device.

## Hello World

Now let's get started with Cordova. We will start with the classic Hello World example. We will create our Hello World application is a package called hello. This will create a starter package with a few lines of source code to get your started.

```
cd ~
cordova create hello com.example.hello HelloWorld
cd hello
```

Check your list of platforms

```
cordova platform ls
```

Add the Android platform to your project.

```
cordova platform add android
```

Build and Android package from source code.

```
cordova build android
```

Start the emulator

```
cordova emulate android
```

Close the emulator.

## Debugging with Logcat.

[Logcat](#) is the default android debugger, we can use this for tracking down issues in our web code. To do this we will want to add the [console plugin](#) to our Cordova app.

```
cordova plugin add cordova-plugin-console
```

Start your emulator and in another terminal start logcat

```
cordova emulate android  
  
adb logcat
```

Everything that happens in the emulator is logged. Logcat `adb` is installed with android studio and lets us read the logs in real time. In a new console, run the logcat command.

```
adb logcat
```

You'll notice straight away that this is far too verbose to be useful, so you will want to run it with filters. We will use the regex filter to only return messages that have contain the string *INFO:CONSOLE*. This will limit Logcat's output primarily to `console.log()` calls. Making it far less verbose and allowing us focus on messages that matter.

```
adb logcat ActivityManager:I Cam:V -e INFO:CONSOLE*
```

Camera Plugin <https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-camera/>

## Additional Resources

[Use SQLite In Ionic 2 Instead Of Local Storage](#)

## Syllabus (4/2/2018 - 6/13/2018)

- Class Hours: Monday-Friday, 9am-5pm
- First Day: Monday, April 2, 2018
- Last Day: Wednesday, June 13, 2018
- Class will **NOT** meet on
  - Monday April, 23, 2018
  - Friday May, 25, 2018
  - Monday May, 28, 2018

## Description

The class is 40 hours a week for 10 weeks. It is comprised of a series of units. A unit is comprised of working lectures, exercises, labs and additional resources. The goal of this class is to provide a working knowledge of web and mobile application development. This course is not a deep dive on any one topic, rather it is intended to provide a working foundation that will allow you to hit the ground running and give you the knowledge base upon which to build a career. In most cases you will get a light introduction to a topic; typically within a specific context. We will build on these topics as we progress through the course. The syllabus is tentative and may be adjusted to the pace of the class as a whole. The last two to three weeks of class are open and are intended for to allow each student to complete a single or multiple projects to add to their portfolio.

## Unit Composition

### A Lecture

An overview of the topic(s) at hand.

### Exercises

Guided application of the topic(s) at hand.

### Labs

The work you are expected to do on your own. Lectures and exercises should take roughly 40% - 50% of the class time with the rest of the time being spent in labs. If you get ahead it is expected that

### Additional Resources

This section provides links to additional reading, conversation, ebooks, videos online courses ect. While not required it is encouraged and expected that you will spend some time reviewing this material.

### Udemy

We use Udemy as supplemental material. While it is not required it is encouraged to at least review the courses provided in the Additional Resources section. You will have access to Udemy for one year from today so even after this course is finished you can continue to learn in a structured environment.

### Khan Academy

This provides free online learning for various topics. This is mostly included as a review of Algebra, Geometry, etc. We do not get too heavy with these topics but any one wanting a review can do so through the links provided.

## TENTATIVE SCHEDULE

### Week 1 (Dev, Bash, HTML, CSS)

#### Build and Understand a Development Environment ~1 Day

- Install Ubuntu Linux
- Linux basics
- Install dev tools
- Install the LAMP stack
- Configure the Apache Web Server
- NPM
- Git

#### Programming Basics ~1 day

- Shell Scripting with Bash

#### Introduction to Web ~3-5 days

- The Anatomy of a URL
- HTML
  - Introduction to HTML
  - GitHub Pages (Basic Website)
  - Web Forms
  - HTML Resume
- CSS/Preprocessors
  - CSS and Basic Styles
  - SASS/SCSS and Gulp Processes
  - CSS Layouts (floating grids, CSS grids, flexbox)
  - Responsive CSS

### Weeks 2-3 (Server Scripting, PHP, JavaScript)

#### Server Scripting with PHP ~1-3 days

- Introduction to Templates
  - Build a template in PHP
  - PHP Web Site
  - Form Processing in PHP

#### JavaScript, jQuery, AJAX and Bootstrap ~5-7 days

- Programming Basics with JavaScript
- Walking the DOM
- Catch Events
- Programming the Canvas

- jQuery Basics
- Ajax
- Introduction to Bootstrap

## **Weeks 3-4 (SQL, NoSQL, Web Frameworks)**

### **MySQL and CakePHP ~3 days**

- MySQL
- CakePHP
- Unit Testing
- Automated Code Review with Code Climate.

### **MongoDB, Express ~5 days**

- MongoDB
- Express
- REST API
- Web Sockets

### **Introduction to the Cloud 1-2 days**

- Launch a production site to Digital Ocean on your preferred tech stack (MEAN or LAMP).

**Output** Your personal production site on Digital Ocean on your preferred tech stack (MEAN or LAMP).

## **Week 5 (Angular)**

### **Angular ~5 days**

- Tour of Heroes Tutorial
- Interact with your REST API

## **Week 6-7 (Hybrid Mobile)**

### **Apache Cordova and Ionic ~10 days**

- Learn how to build mobile device emulators.
- Learn how to emulate mobile devices using the Chrome browser and Dev Tools.
- Run existing web code as a mobile app
- Ionic Applications

## **Weeks 8-10**

If applicable week 8 may serve as catchup, review or overflow time for any topics of which there are issues.

## **Projects**

Whatever you want to build. Start thinking of a project sooner than later

