Table of Contents

MicroTrain's Dev Bootcamp

Resources for MicroTrain's Developer Bootcamp.

Structure

This course is designed to provide the student a hands on approach to learning web and hybrid mobile development. The goal is to provide someone who is new to development just enough to get themselves started building building real world applications. Expect lectures to makeup roughly 25%-50% of the course time with 50-75% of the time independently completing labs. Many subjects will receive little to no coverage the first time they are presented many of these subjects will be expanded upon in latter lessons and some will be covered under additional reading.

Lecture

Lecture will cover the material presented in this repository. The root level of this repo is broken down into numbered sections i.e. *01-DevEnvironment* with each section consisting of multiple lessons or topics. The lectures are designed to be working lectures in which a topic will be explained and the students will work out examples during lecture time.

Exercises

Think of these as a guided lab in which the instructor walks the student solving common development problems relating to the topic at hand.

Labs

The best way to learn is by solving real problems. Labs are designed to build upon the current topic and previous lessons. Students may split up into pairs or solve the problem independently. If a pairs approach is chosen each student is still expected to submit their own source code.

Additional Resources

Each topic will provide an additional reading section. These sections are comprised of links to free ebooks and articles. These will not be covered in class, it is expected that the student will review these links outside of lecture. Like additional reading these are links to third party tutorials. These may be completed during normal lab time if time permits or independently. Either way the student is expected to review and complete these tutorials.

Suggested Reading

- How To Become A Hacker
- PROGRAMMING CONCEPTS: A BRIEF TUTORIAL FOR NEW PROGRAMMERS

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 your 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. Lets 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.
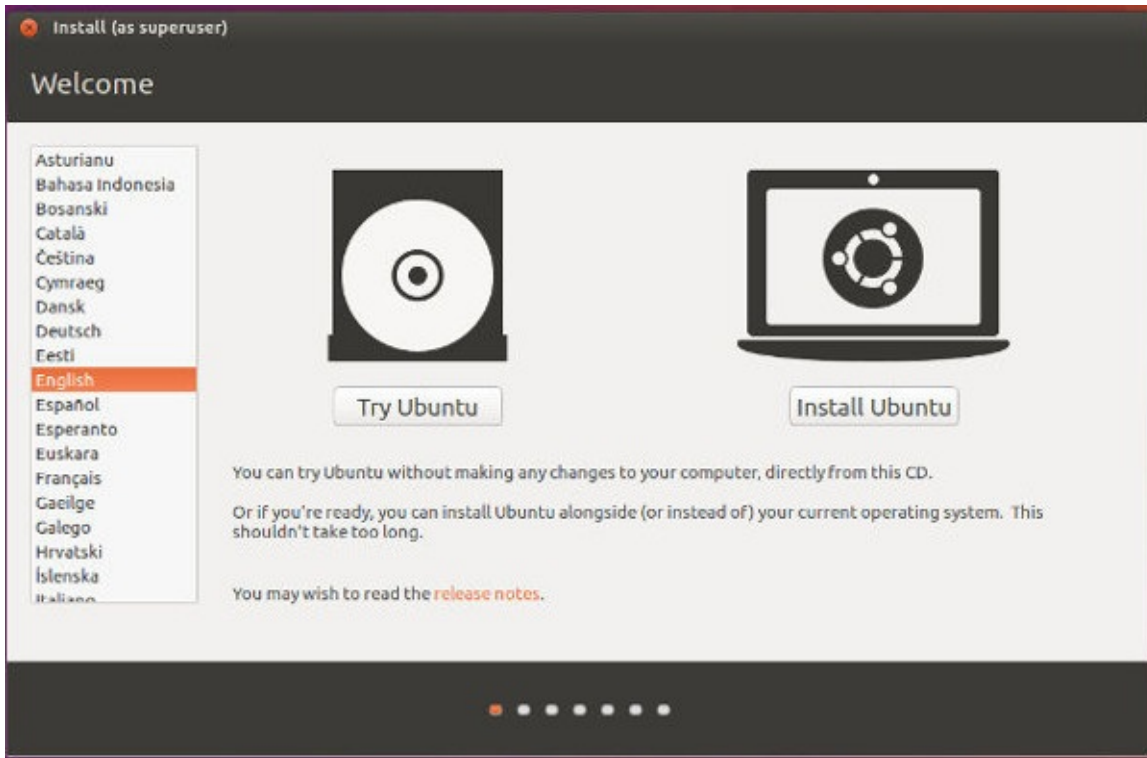
You will now be given two options *Download updates while installing Ubuntu* and *Install third-party software...* select both of these options.
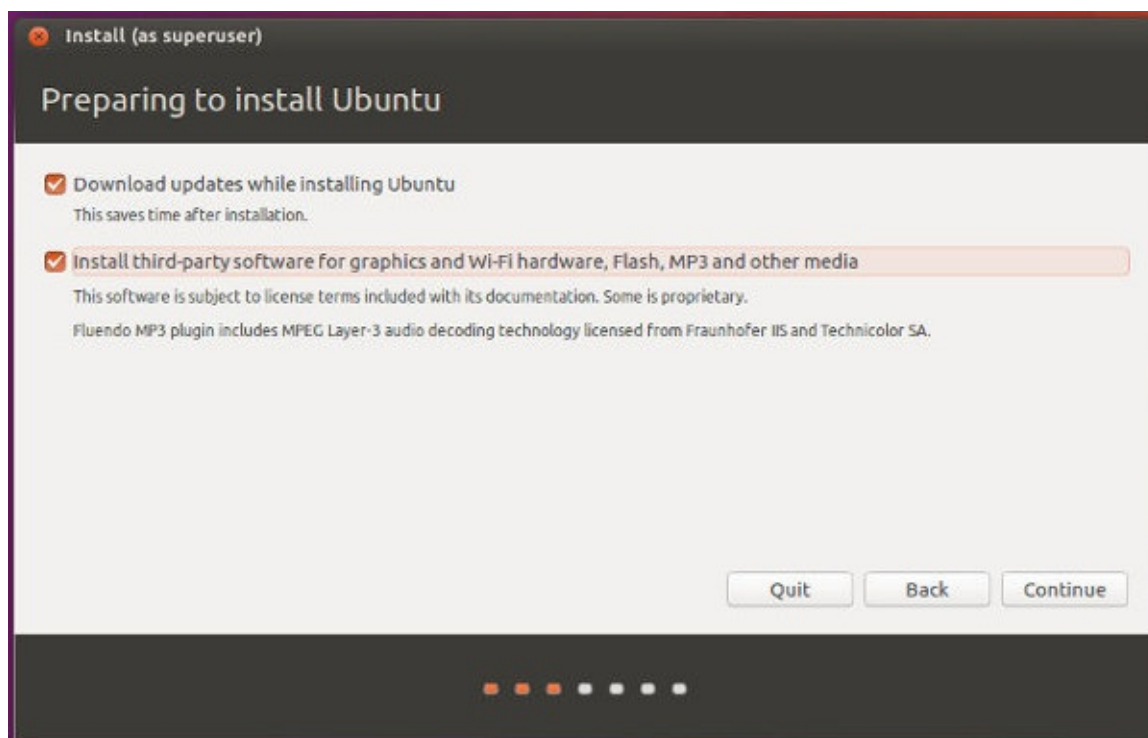
Now it's time to install the system. To keep things simple we will do a simple install. Be sure the *Erase disk and install Ubuntu* option is selected and click the *Install Now* button. You may see a dialog that asks *Write changes to disk?* with some additional details, simply click the *continue* button.
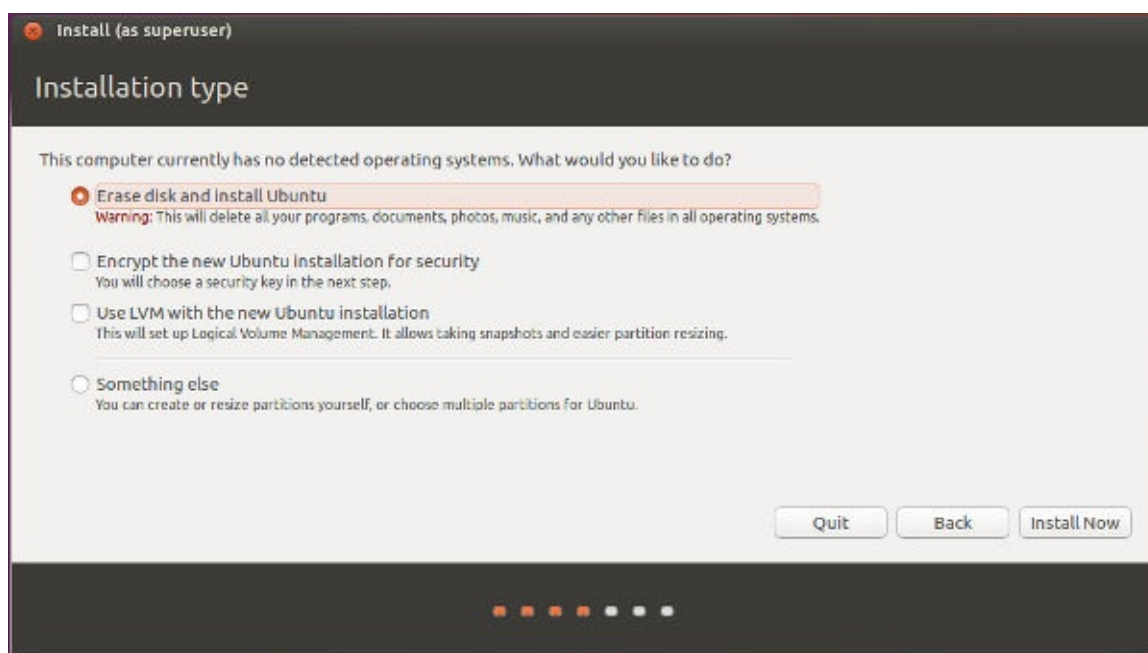
You will now see the *Where are you?* screen. This should default to Chicago (or your default location), if not choose Chicago (or your default location) as your location and click the *Continue*.
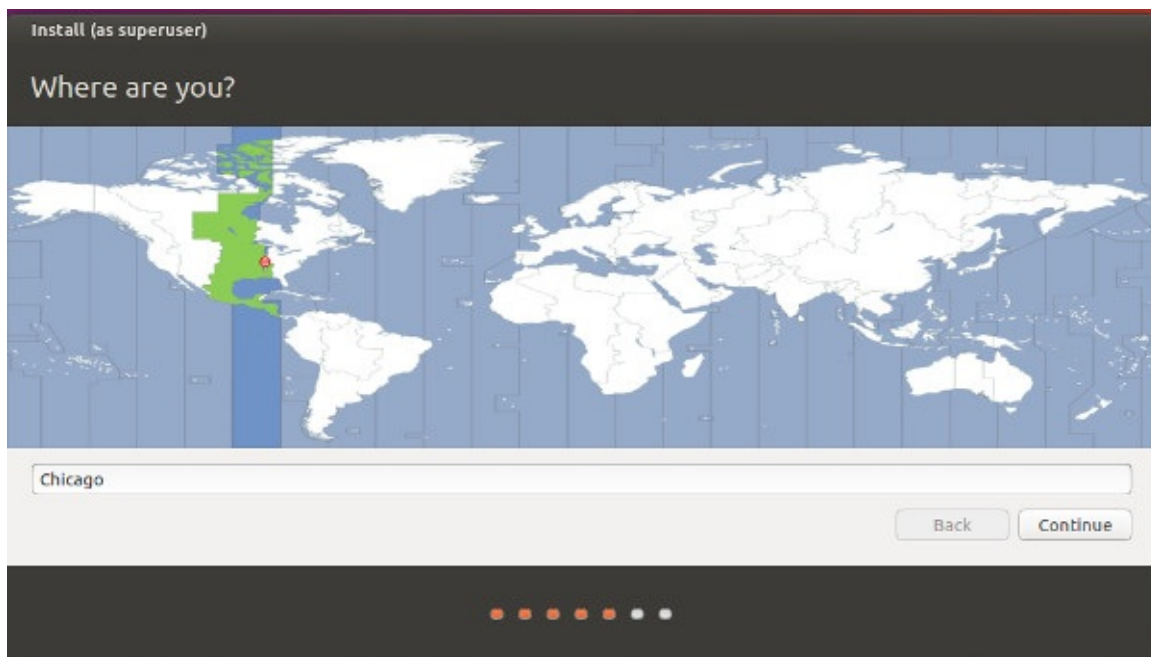
This sets your systems local timezone. For *Keyboard layout* choose English (US) and English (US) and click the continue button.
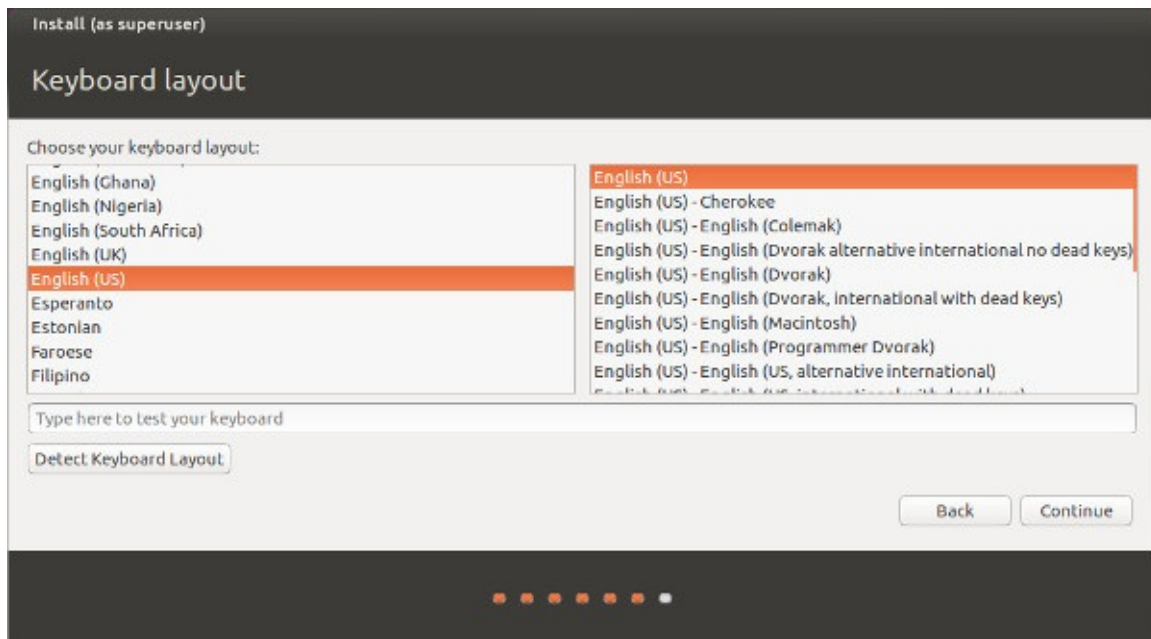
On the *Who are you?* screen enter *Your name:* as your first an last name, accept the default computer and usernames and choose a password. Do not forget your password, we are not able to recover this for you. Be sure the *Require my password to log in* option is select and that the *Encrypt my home folder* option **IS NOT** selected. Click the continue button and wait for the system to install.

While the system installs, you will see series of flash screens telling you the features of Ubuntu.
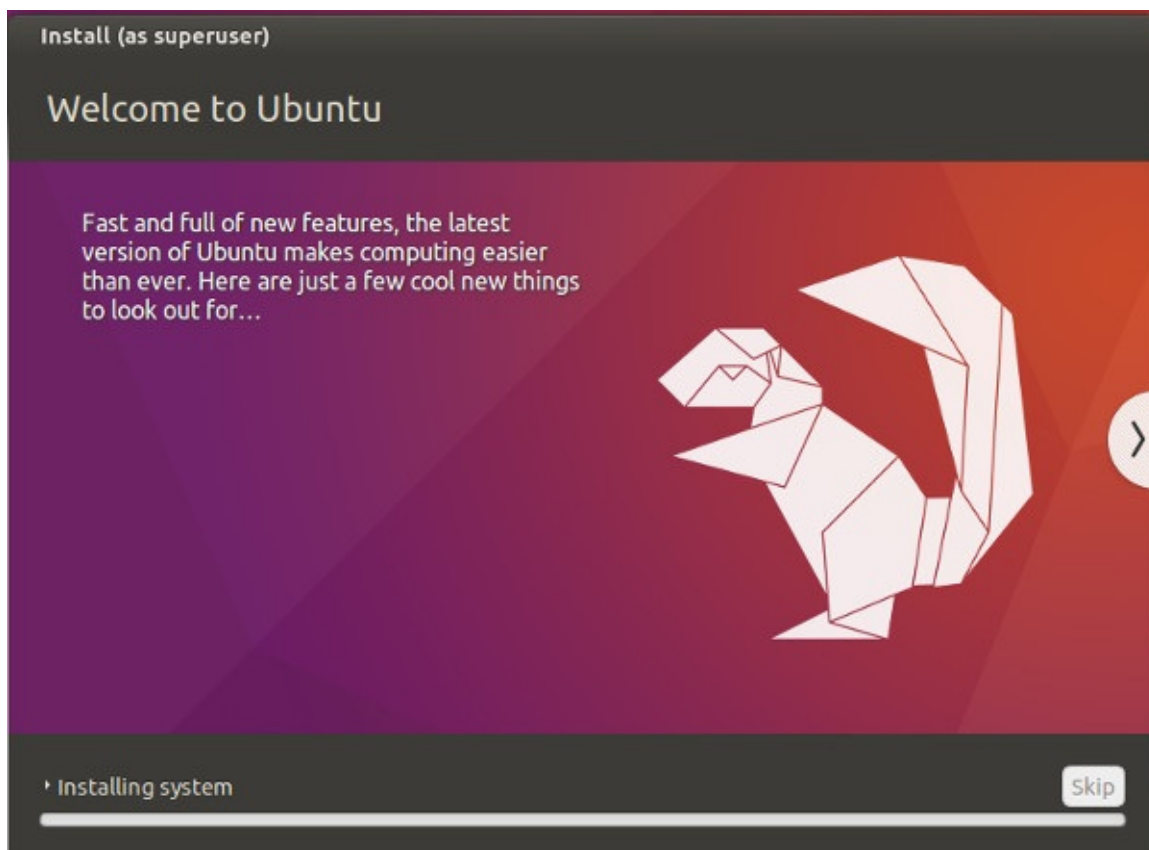
One installation has completed you will be prompted to restart your system. Click the *Restart Now* button.
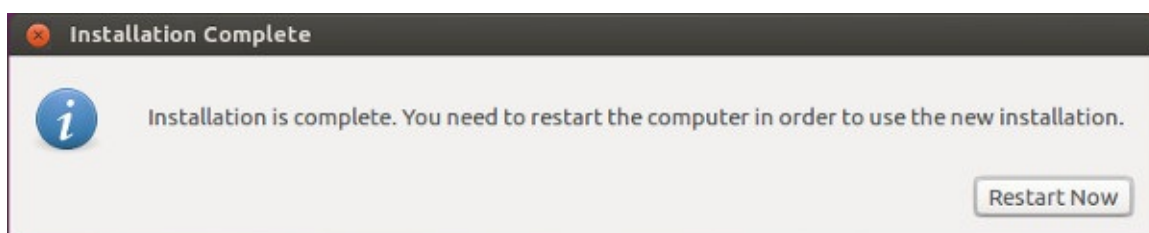
You should see a prompt that says *Please remove the installation medium, then press Enter:*. Pull the dongle from the USB drive and press the enter key.

Next: Linux Basics

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)
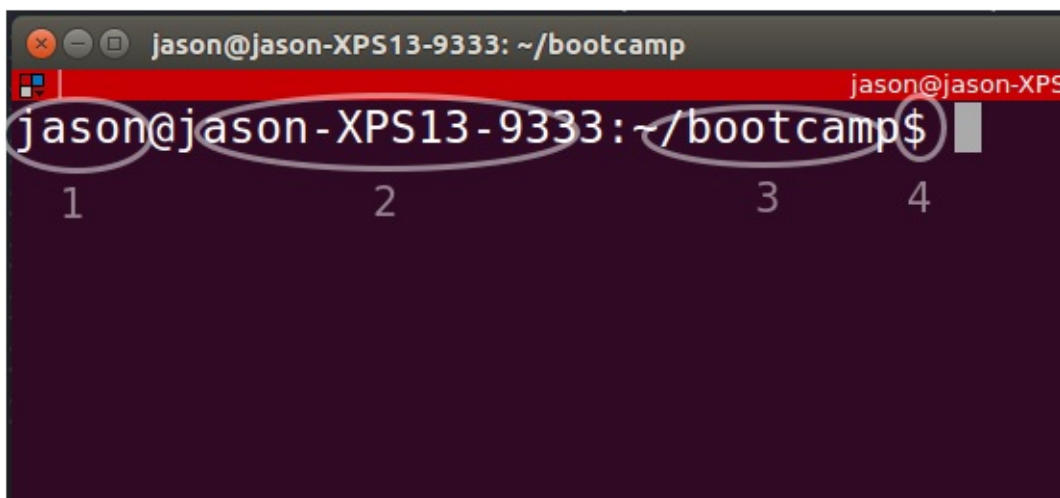   - The space immediately to the right of # or $ is the command line entry point.

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 surounded 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.

Additional Resources

- Ten Steps to Linux Survival
- Linux Fundamentals

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 privleges.
- `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 Linus 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.

Atom

Atom is the text editor or pseudo IDE we will be using to write code. Install Atom using the same steps you used to install Chrome. Remember to pin Atom 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
atom-amd64.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 atom-amd64.deb
-rw-rw-r-- 1 jason jason 45892904 Feb 16 16:18 google-chrome-stable_current_amd64.deb
```

Since the have been installed we no longer the files lets 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 [fn1]
  - `-R` - remove directories and their contents recursively. [fn1]
- * - 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
```

Additional Resources

- VIM Book
- Chrome Dev Tools
- Atom Flight Manual
- Filezilla Docs
- SSH Man Page

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 >= 7.1 and 4.2 respectively.

Additional Resources

- Debian and Ubuntu Based Linux Distribution

GIT

Git is a free and open source distributed version control system (VSC). 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 binding process.

1. Add an SSH Key
2. Setting up Git
3. Testing your SSH connection

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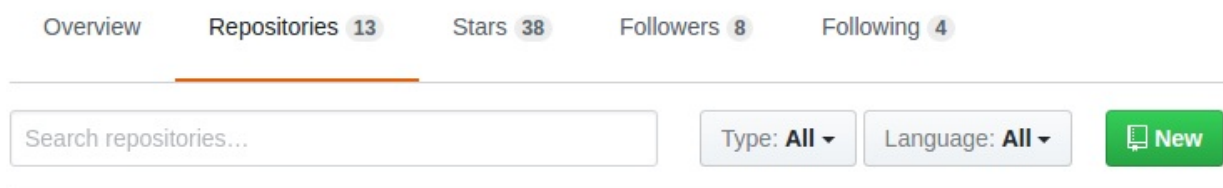
Create a project called mtbc (MicroTrain Bootcamp). This is the project you will use for much of this course.

- 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   **Repository name**

jasonsnider ▾   /   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.
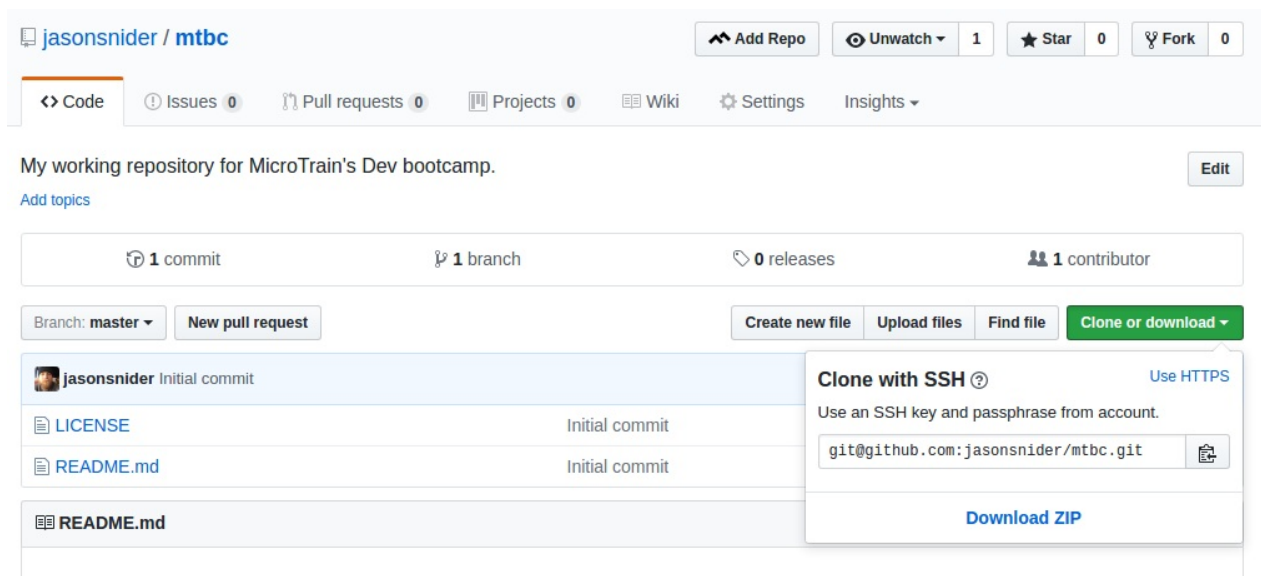
Add .gitignore: **None** ▾   Add a license: **MIT License** ▾   ⓘ

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 been directed to the new repository. If not, you can find it under the 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`

📖 jasonsnider / **mtbc**   Add Repo   ⊙ Unwatch ▾ 1   ★ Star 0   ⑂ Fork 0

<> Code   ⓘ Issues **0**   ⑂ Pull requests **0**   ▥ Projects **0**   📖 Wiki   ⚙ Settings   Insights ▾

My working repository for MicroTrain's Dev bootcamp.   Edit
Add topics

⊙ **1** commit   ⑂ **1** branch   ◇ **0** releases   ⚑ **1** contributor

Branch: master ▾   New pull request   Create new file   Upload files   Find file   Clone or download ▾

jasonsnider Initial commit

📄 LICENSE   Initial commit

📄 README.md   Initial commit

📖 **README.md**

Clone with SSH ⑦   Use HTTPS
Use an SSH key and passphrase from account.

git@github.com:jasonsnider/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

Name	Last modified	Size	Description
Parent Directory		-	
LICENSE	2017-06-13 09:08	1.0K	
README.md	2017-06-13 09:08	60	

Apache/2.4.18 (Ubuntu) Server at localhost Port 80

Exercise 2 - Commit a Code Change

Now open Atom and click on the file menu. A use the key combo `Shift + Ctrl + A`, this will raise an *Open Folder* dialog. Navigate to */var/www/mtbc* and press `OK`.

```
07-Git.md — ~/bootcamp — Atom

  ∨ 📖 mtbc
     > ■ .git
       📄 LICENSE
       📑 README.md
```
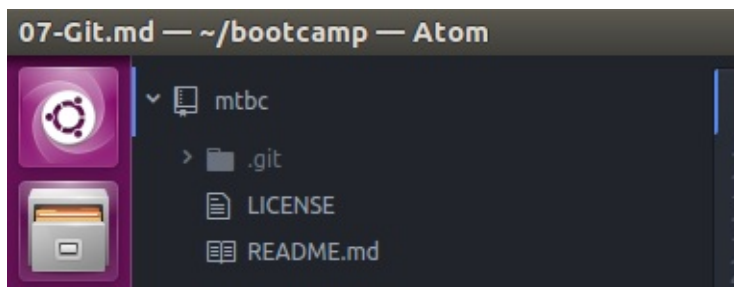
Now click on the file README.md from the *mtbc* project folder. README files are a best practice in software development. REMDME files are human readable files that contain information about other files in a directory or archive. The information in these files range from basic infomation about the project 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 Atom's *Markdown Preview* by opening the file and pressing `Shift + Ctrl + M`.

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 `#` 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.

```
git status
```

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 ask 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 develop branch (we call it dev). I like to create branches as follows.

```
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.

Additional Resources

- Documentation
- ProGit.
- 3 Git Commands I use every day

Next: Programming Basics

Programming Basics

In this section you will learn

- Basic control structures
- The anatomy of a URL/DNS Basics
- The basics of Bash scripting
- The basics of PHP

The Anatomy of a URL

Uniform Resource Locators (URL) is an interface through which a user can communicate with a web page. In modern browsers this is accessed through the address bar. If you were to open a CLI based browser such as wget this would be passed as an argument into the command line..

Let's break this down by taking a look at https://www.google.com/search?q=anatomy+of+a+URI#jump.

PROTOCOL	DOMAIN	PORT	PATH	QUERY STRING	FRAGMENT
https://	www.google.com	:443	/search	?q=anatomy+of+a+URL	#jump

Simply stated a website is a program accessed through using a series of web protocols. For example `http://wwww.example.com` would tell the broswer to go to the root of www.example.com (typically this is a file called index). Everything you see after that is a series of commands. Originally the web served static files so what I am calling commands was really just a directory structure. For example `http://wwww.example.com/blog/post/27` is most likely requesting the 27th post from the websites blog. At one point in time this would have been a file named 27 in a directory called which would have been in a directory called blog. Today, this is most likely a stored in a data base and post is probably a script that has just been instructed to return a given row (in this case 27) from the database.

GET and POST parameters

When dealing with web applications the most common types of requests are GET and POST. These types of requests the user to pass data to the server either through the URL (a GET request) or through a form submission (a POST request). Modern web platforms and programming languages will provide an interface for dealing with the data passed through either of these request types.

PHP provides access through the use of superglobals `$_GET['email']` and `$_POST['email']` . CakePHP and MVC framework written in PHP uses a request class `$this->request->params->named['email']` and `$this->request->data['email']` . If your running Express on top of Node.JS you'll use something like the following `req.query.email` (GET) `req.body.email` (POST).

It would be common to refer to these practices as retrieving GET parameters (params) and retrieving POST data.

Additional Resources

- The Difference Between URLs and URIs
- Uniform Resource Identifier (URI)
- StackOverflow: Node.JS GET Params
- StackOverflow: Node.JS POST Data

Khan Academy

IP addresses and DNS

Next: PHP Basics

PHP Control Structures

Programming is little more that reading data and piecing together statements that take action on that data. Every language will have it's 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
<?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 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 know 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* 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.

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

In PHP `$_GET` is a superglobal so it is always available, this is NOT something you want to try to initialize. `empty()` is used to determine if a variable is truthy or falsey where falsey values equate to empty or falsey returns true. Prefixing `empty()` with an *!* `!empty()` reverses the return values. In plain English `if(!empty($_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 it's use of C style syntax.

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

```php
//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 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.*

```php
//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.

```php
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
<?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
<?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
<?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
<?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

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.

CSS

In this section you will learn the basics of

- CSS
- Less
- Sass

LESS and SASS

LESS and SASS (SCSS) are the two leading preprocessors. A CSS preprocessor is a superset of CSS which means anything that is written in raw CSS will run under both LESS and SASS. LESS and SASS extends CSS with programming like capabilities; basically variables and limited control statements. This is adventurousness 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.

Less

Install less, since less is written in Node.JS we will use npm.

```
sudo npm install -g less
```

Variables in less. Less denotes variable with an @ at symbol. Less files must have the *.less* extension.

Exercise 2 - Less Variables

Create the path *~/less/var.less* and add the following lines.

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

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

Then run the less compiler against that file.

```
lessc ~/var/less/var.less
```

You will see the following output in the console.

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

Sass

Install sass, since less is written in ruby we will use gem for the install. Start by installing ruby.

```
sudo apt-get install ruby
sudo su -c "gem install sass"
```

OR

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

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 2 - Sass Variables

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

```scss
$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.

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

Exercise 3 - 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* amd verify your changes.

Exercise 3 - Implement sass in your project

Move */var/www/about/css/dist/main.css* to */var/www/about/css/src/main.scss*

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

Then compile the sass file

```
sass /var/www/about/css/src/main.scss /var/www/about/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 mulitple 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 that 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*.

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

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

```scss
.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.

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

```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.

```scss
#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.

```scss
.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 Less 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.

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

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

JavaScript Control Structures

Programming is little more that reading data and piecing together statements that take action on that data. Every language will have it's 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.

```html
<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 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 know 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* 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*.

```html
<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

```html
<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.[1]

> 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.[1]

> When a collection is created, a filter and a root are associated with it.[1]

> 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. [1]

</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.

```javascript
// 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.

```javascript
//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.

```javascript
//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

- [1]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 ofter 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 .htacces 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 alot 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 somthing like *TO_DATE in mysql* and one of the first results will liekly 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. Once common rule is for all tables to end in a plural form of the entity it represents.

Columns

A column is often refereed 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 a length.

Column names should be short and two the point. While style guides vary typical rules state a column name should differ from that of the table and must not over lap 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.

Indicies 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 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 your running a local instance of MySQL, your 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 data base 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
--- ----------- ---------- -------------      O
2  | frank      | brown     | 1951-04-01      W
--- ----------- ---------- -------------      S
3  | bob        | gray      | 2004-08-17
--- ----------- ---------- -------------
```

Keys and Indicies

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. Typcially an aut-incrementing integer or a UUID. In the above example the id column would be my primary key.

Forgien 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 forgien_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 togeather 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 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 and 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 patter 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 a table in a database, a reference to an API, a spreed sheet, etc. For our user a model will reference a table in a database.

View

View's 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 you 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 it configuration as an array, find the *Datasources* attribute somewhere arounf the 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.
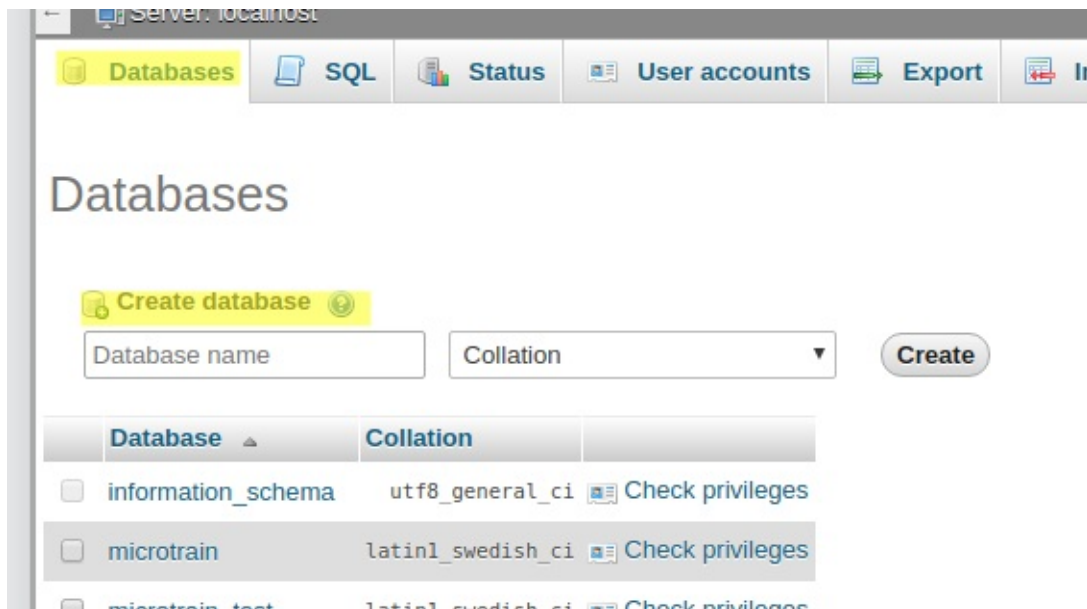
Update the database configurations as follows.

default

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

test

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

Return to http://localhost:8765/ 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 lifcycles 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

```sql
/* 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
```

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 on 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 exists, calling insert against the collection will create the collection.

Now if run you `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()* 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.

```
db.users.find({ "_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.