

## Appendix

Here's a few extra materials and resources on helpful topics that would have been covered in the full course but can't be fit into the intensive as there isn't enough time. Take the time to read these extra materials in your own time, if you wish.

### DNS

IP addresses are a sequence of numbers and '.'s such as **212.58.244.67**. These aren't very easy to remember. Instead the internet works on a domain name system, that matches domain names such as **bbc.co.uk** to IP addresses.

#### **Task:**






Type **74.125.224.72** into your browser's address bar. What happens?

One of the first things that happens when you type a URL into your browser is a *DNS lookup*: your computer contacts a DNS server (*Domain Name System server*), which is basically a massive address book of IP addresses. The DNS server converts the domain name from the URL (e.g. **bbc.co.uk**) into an address for a server (such as **212.57.244.67**).

Once the address of the web server has been found your request is forwarded on to it, the web server will interpret your request and send back one or more files.

### Server-side technologies

There are many options for building a dynamic server-side site. Common choices are:

 <b>Ruby on Rails</b>	 <b>PHP</b>	 <b>Django</b>	 <b>node.js</b>	 <b>WORDPRESS</b>
Web development framework written in Ruby (programming language)	Programming language popular in the early 2000s.	Web development framework written in Python (programming language)	Web development framework written in Javascript.	A blogging platform (now capable of much more) written in PHP.

## The Command Line

The command-line interface, sometimes referred to as the CLI, is a **tool** into which you can type text commands to perform specific tasks—in contrast to the mouse's pointing and clicking on menus and buttons.

Since you can directly control the computer by typing, many tasks can be performed more quickly, and some tasks can be automated with special commands that loop through and perform the same action on many files—saving you, potentially, loads of time in the process.

The command line is simply a place where you **type commands directly to the computer**.

The computer will take these commands at face value, and will attempt to carry out any command that it understands. Unfortunately, the computer does not speak in any language spoken by humans (although it has recognizable elements). The people who created the operating systems you work on have thus created a standard set of commands that are built into the computer without having to be processed or compiled.

**NOTE:** The command line, as with all power, has its [risks](#). You have the capability to instruct the computer to do anything it has the capability of doing. If you instruct the computer to erase all of your data, it will cheerfully proceed to do so. **Do not run a command just to see what it does.** Make sure you understand what the command is supposed to do first, especially if the command involves changing or removing files.

The application or user interface that *accepts your typed responses and displays the data on the screen* is called a **shell**, and there are many different varieties that you can choose from, but the most common these days is the **Bash** shell, which is the default on **Linux** and **Mac** systems in the **Terminal** application. By default, **Windows** systems only include the anemic Command Prompt application, which has nowhere near the power of Bash, so for the purposes of this article we're going to suggest you use the open source [Cygwin](#) tool as your Windows command line, since it's quite a bit more powerful.

## Git

Git is a very powerful and popular version control system, and the only place you can run **all** Git commands is from the command line. **Please note git is not the same as Github!** You'll see the difference in practice very soon.

Most GUIs only use a subset of all functionalities for simplicity, so if you know how to run the command line version, you can probably figure out any GUI versions. Command-line tools are also available across operating systems, whilst GUIs are subjective for different companies or teams. So it's best to understand it for the future.

**Task:**

Try Git [here](#).

We recommend that you follow [this guide](#) to try it yourself in your own command line, the GitHub guide for this is [here](#).

## Publishing 'Hello World' on GitHub Pages

### Getting to grips with git

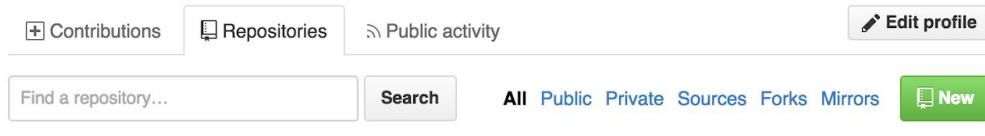
What makes a folder a 'git repository' is a hidden folder with details of changes. Today we're going to use Terminal (Mac) or the command line (Windows). The basic commands we're going to need are as follows:

Command	What it does	Example
<code>cd + { folder }</code>	Change directory/location	<code>cd Documents/my_site</code>
<code>touch + { filename }</code>	Create a file without opening it	<code>touch index.html</code>
<code>mkdir + { folder }</code>	Create a folder	<code>mkdir images</code>
<code>cd ..</code>	Changes directory to the containing folder	<code>cd ..</code>

What we're going to do is open up the terminal. Your default location will either be '-' on a Mac which means you are in your home directory, which is the folder of the profile name. On a Windows machine your default location will be your home directory.

## Creating a Git Repository

We are going to head over to Github and login. On your profile page select the ‘Repositories’ tab and select the ‘New’ button.

**EChesters**

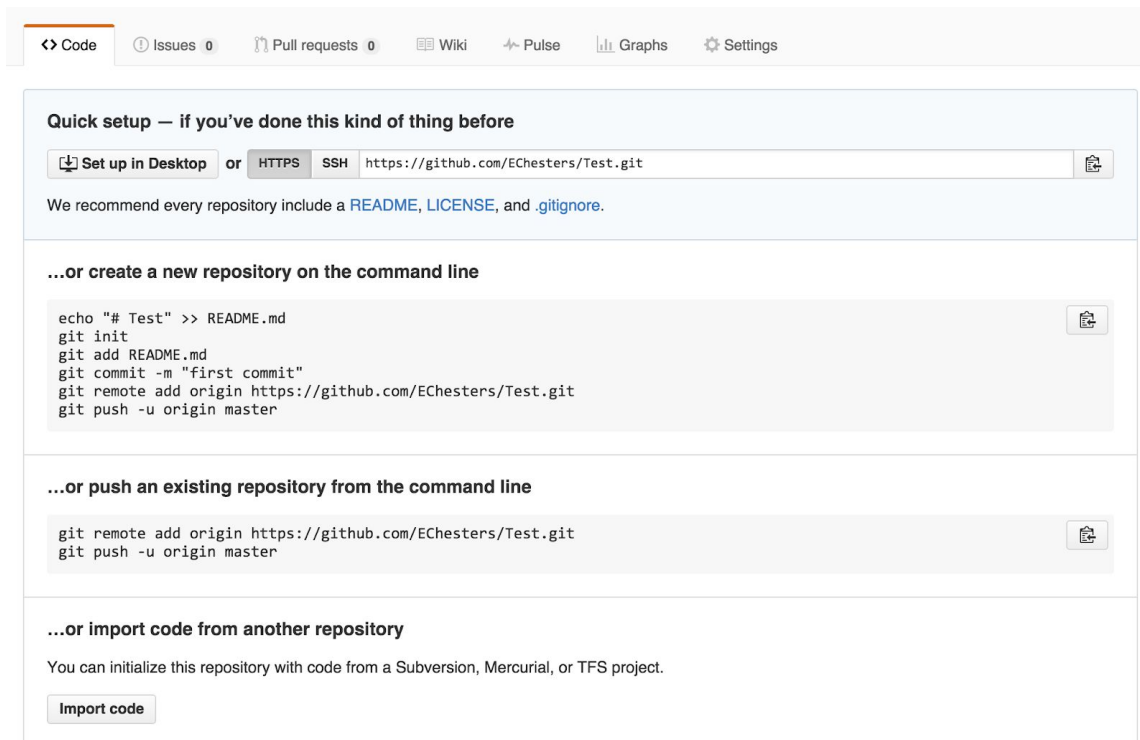
CSS ★ 0 0

Personal Blog with Ruby Sinatra

Updated 3 minutes ago

Github then presents you with a form, asking for the name of the repository, which in this case is the name of your website. You can enter a description, but this is optional and for now we’re going to keep the rest of the fields as default.

Then select “Create Repository”. This will take you to the repository page, where no code has been added yet. Github then offers you different options of uploading code. Since we have no code yet we’re going to clone the repository on our machines.



Head to the terminal on your machine and change the directory to the desktop. Type in the following command:

```
$ cd Desktop
```

**Please note: you do not enter the \$ sign.** This is the standard which shows what comes after the dollar needs to be entered into the terminal.

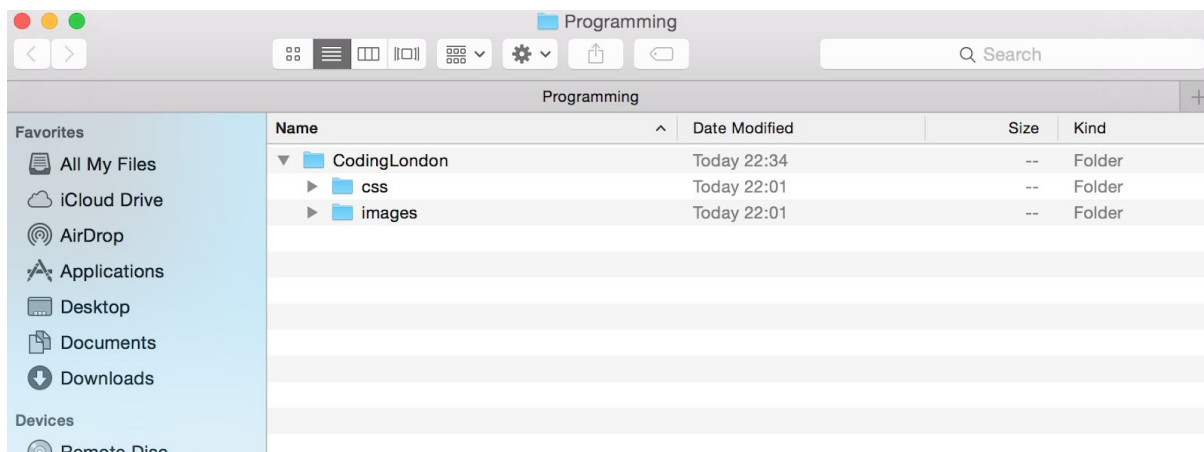
Then the command we're going to run is the following:

```
$ git clone https://github.com/YOUR_REPOSITORY
```

What this does is copies the folder from Github and puts it on your machine. You now have your first git repository!

## Creating the website structure

Now let's create a folder structure for our website. To keep things separate we will need the following folders:



The "CodingLondon" folder here is the folder that you just copied from Github. Once you have the folders we can now start adding files.

For all web sites you need an index.html If you do not have this file, Github pages will throw you an error. You can either make a new file in the terminal using:



## Summer Intensive 2018 – Appendix

```
$ touch index.html
```

or by opening a text editor and saving a blank file called index.html

The next file we're going to make is a basic styles.css inside the css folder. This can be done the same way as above.

**Git will only push folders which has a file in them.**

For images, grab any image you want on your website and save it inside the images folder. We now have a file in each folder, which means we are ready to push our first commit to Github!

### Our first commit

When we make changes we have two choices, we can either commit to them or we can revert them. When we commit a change, first of all we need to add files to what's called a staging area. This is where we prepare which files we want to commit. If for example we have changed too much, you can add half of the files, commit that half and then revert the other half.

To add files to the staging area we need to run in the Terminal (located at the folder):

```
$ git add -A
```

This adds **everything**: new, deleted and modified files. In order to see what has changed and what will be committed you can run a status command like so:

```
$ git status
```

After adding the files we want to commit, we then need to commit to that code. At this stage, we could still change our minds. So we need to enter the commit command:

```
$ git commit -m "Your change message goes here"
```

At this stage we have committed only to our local machines, which means Github still has no idea that this code even exists. Now what we need to do is **push** the code to a server.

```
$ git push -u origin master
```



## Summer Intensive 2018 – Appendix

By default your git “tree” only has one branch, which is master. **Do not delete this branch.** After this, go check out your Github account and repo and admire your wonderful code!

### How Github Pages Works

Github Pages is a free hosting tool offered by Github. They only serve basic static web changes where no logic such as logging in and storing data is required.

Going back to the branches that Git has, Github pages looks for a branch called gh-pages and builds a web page from that.

What we’re going to do is create a second branch and “switch” to it. This is where any code you want to push up to your site needs to be. If it’s on master, Github Pages will ignore it.

As you’ve seen above, we’re going to use [GitHub Pages](#) to host our site – it will be the server where the files for our site are stored. Once you publish your site on github pages, your website will show up at the url [your-github-username].github.io.

GitHub provides a developer pack for students as well, [here](#), including discounts on DNS management, a GitHub account upgrade (for free), Microsoft Visual Studio, and one year free on a domain name from namecheap.

Create a branch and checkout that version:

```
$ git checkout -b gh-pages
```

Now you have officially created a second ‘version’ of your code!

### ‘Hello World’ Page

Now what we’re going to do is create a basic HTML page to push up to Github Pages. In your index.html file create a basic HTML page. If you’ve forgotten how, you can quickly Google this. If you’re using Sublime Text, enter ‘html’ and then press tab. This should give you the structure.

In between the <body> tags, enter in your first header:

```
<body>
```



## Summer Intensive 2018 – Appendix

```
<h1>Hello World!</h1>
</body>
```

Open the file in a browser locally to make sure it works. Your screen should just show a header with “Hello World!”. If this works we need to then head back to the terminal, add the files, commit them and push.

```
$ git add -A
$ git commit -m “Added hello world page”
$ git push --set-upstream origin gh-pages
```

The last command is different because we’re pushing a new branch and creating this on Github as well. This just sets the location to be that specific branch.

After this is done, we can now check the code is on Github. Then head to settings on the tabs on Github. Under its own box is a Github Pages section which will give you the link to your new amazing website!

**GitHub Pages**

✓ Your site is published at <http://echesters.github.io/WHFNP-Website>.

**Update your site**

To update your site, push your HTML or [Jekyll](#) updates to your gh-pages branch. Read the [Pages help article](#) for more information.

Click on that link and you should now have your website up and running!

Any changes you want to take affect on your website **must** be on this branch. Now you can make changes, push them live and build your own free website.

### Conflict scenario!

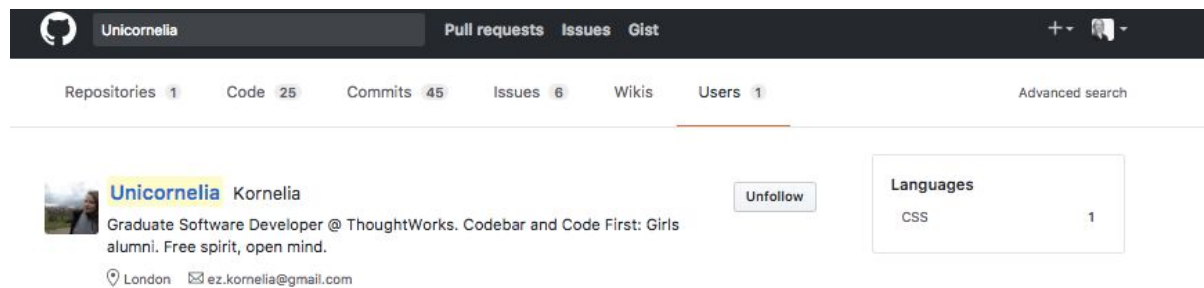
So you now have your very first repository– congratulations!

However, as you’ll be working with other students to carry out your end of course projects, we want to prepare you for some potential conflicts!



**Task:** Turn to your neighbour and decide who will be making the conflicts ! We'll refer to this person as Person 1, and Person 2 as the person who created the repository. Now follow these steps!

Person 1 ask Person 2 for their username, then search for them on Github. Click on their user profile, where you'll also find their repositories. Person 2, on your repository click 'settings', then 'collaborators' then add Person 1 as a collaborator.



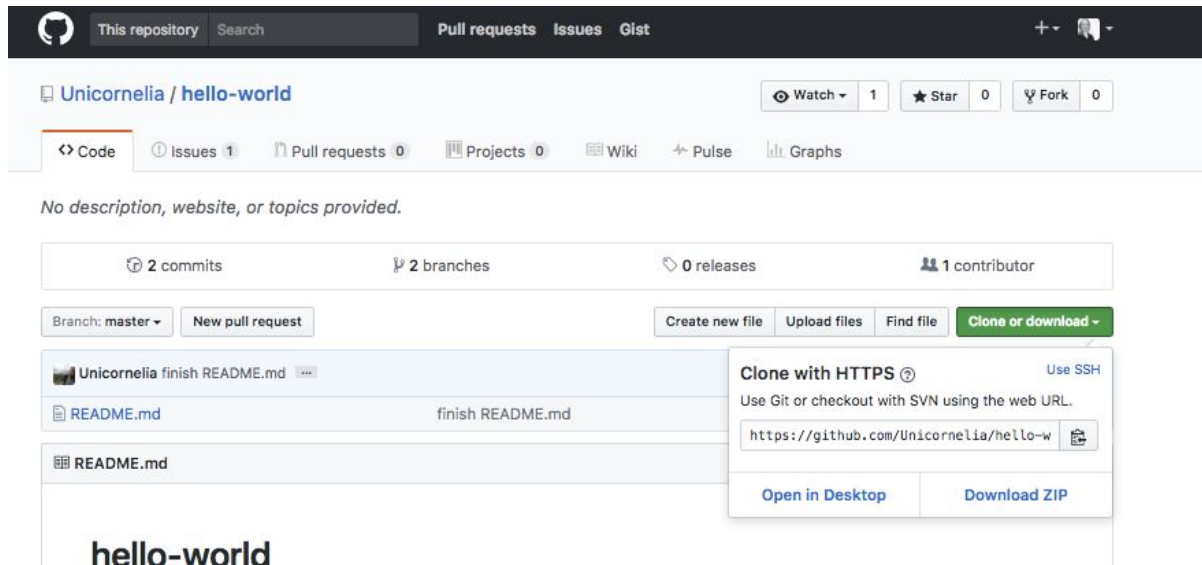
Person 1, firstly make sure you're in coding folder. Then make a new directory in that folder called "hello-world" :

```
$ mkdir hello-world
```

Now, Person 1, click on the green 'clone or download' button on Person 2's repository.



## Summer Intensive 2018 – Appendix



Now copy the 'Clone with HTTPS' link using git clone command:

```
$ git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
```

You now have a copy of Person 2's repository in your folder. Time to make some conflicts!

Open their repository with your text editor. Person 1, please change "world" to "mum". So the text should now read: "Hello mum", rather than "Hello world".

Click save and then add your changes: Then, as before git add:

```
$ git add -A
```

Git status, to double check your changes have been added:

```
$ git status
```

Then git commit:

```
$ git commit -m "Your change message goes here"
```

Remember, at this stage Person 1 has committed only to their local machines, which means Github still has no idea that this code even exists. Now push the code to the server:

```
$ git push -u origin master
```

Now Person 2, please go back into your text editor and amend your original “Hello world” read “Hello dad”. Follow the same steps as above to add, check status, commit changes and attempts to push the commit. However, they won’t be able to push that commit because their repository is not up to date.

Person 2 now has to deal with the merge conflict, because its local copy is out of date. Person 2 should have carried out git pull before making any changes, to get the most up to date version:

```
$ git pull origin master
```

However, they did not! Sadly for Person 2, whoever syncs last must deal with a ‘merge conflict’. Person 2 please follow these instructions to fix this:

**Fix merge conflicts:**

1. Open the files in a text editor (e.g. Sublime or Atom)
2. Merge conflicts will have lines around them that look like this <<<<<<<<<<
3. Choose the correct lines to keep
4. Edit the lines if necessary
5. Remove the lines with arrows <<<<<<<<<<
6. Add and then commit the changes
7. Push the changes

## Avoiding merge conflicts

Anytime Person 1 or person 2 want to make changes they need to do the following: This will come in very handy for the course competition!

**Make changes and share them**

1. **Git pull origin master** – sync & download any changes from github (from other collaborators) to their local repository
2. **Edit** – make changes in their favourite text editor (e.g. Sublime or Atom)
3. **Commit** – save their set of changes on their own computer
4. **Git push** – sync upload the changes back to github so others can see them

**Google Analytics**

(From Session 4)

Google Analytics is an analytics service provided for free by Google. It allows you get an overview of how many people are visiting your site, where they come from, what they do on your site, and much more.

**How it works**

To use Google Analytics you need to place JS Plug-In, a snippet of JavaScript, (that they provide) on each of the HTML pages on your site. When a user visits the page, the javascript sends a message to the Google Analytics site logging the visit.

**Task:**

1. Set up a Google Analytics account – You want to choose the default 'Universal Analytics' option.
2. Go to the **Admin** section, create a new account for your personal site.
3. Click on the **Tracking Info** under the **Property** section, click on “**Tracking Code**” and install the analytics code on all the pages of your site.

## Introduction to Web Frameworks

A web application framework is a **type of software framework** designed to **support development** of dynamic websites, web applications, web services and of web resources.

A **software framework**, in computer programming, is an abstraction in which **common code** providing **generic functionality** can be selectively *overridden* or *specialized* by user code providing specific functionality.

**Frameworks** are a **special case** of software libraries in that they are **reusable** abstractions of code wrapped in a well-defined Application Program Interface (API), yet they contain some key distinguishing features that separate them from normal libraries.

Software frameworks have these distinguishing features that separate them from libraries or normal user applications:

1. **inversion of control** – In a framework, unlike in libraries or normal user applications, the overall program's flow of control is not dictated by the caller, but by the framework.[1]
2. **default behavior** – A framework has a default behavior. This default behavior must actually be some useful behavior and not a series of no-ops.
3. **extensibility** – A framework can be extended by the user usually by selective overriding or specialized by user code providing specific functionality.
4. **non-modifiable framework code** – The framework code, in general, is not allowed to be modified. Users can extend the framework, but not modify its code.

By using frameworks, we benefit from **peer-reviewed**, **tested** and **pre-written functionalities** that save us time and allow us to share and contribute to the wider developer community. More [here](#).