Comp 388/424 - Client-Side Web Design

Fall Semester 2015 - Week 1

Dr Nick Hayward

Course Details

Lecturer

- Name: Dr Nick Hayward
- Office: 316 Loyola Hall (LSC) & 531 Lewis Towers (WTC)
- Office hours
  - *Thursday afternoon by appointment (WTC)*
  - *Friday afternoon by appointment (LSC)*
- Faculty Page

TA

- Name: Nema Nemati
- Email: nenemati@gmail.com

Course Schedule

Important dates for this semester

- Thursday @ 4.15pm to 6.45pm (6.30pm with no break)
  - *Corboy Law Center, Room 105, WTC*
- DEV week: 5th to 9th October 2015
  - *No class: 8th October 2015*
  - *Demo due 15th October 2015 @ 4.15pm*
- Thanksgiving break: 25th to 28th November 2015
  - *No class: 26th November 2015*
- Final class: 3rd December 2015
  - *Demonstration of final assessment @ 4.15pm*
- Exam week: 7th December to 12th December 2015
  - *Final assessment report due 10th December 2015 by 6.45pm*

Initial Course Plan - Part I

(up to week 7 - 8th October 2015)

- Build and publish a web app from scratch
- general setup and getting started
- maintenance and publication
- basic development and manipulation (HTML, JS...)
- add some fun with Ajax, JSON, server-side...
- useful data storage techniques and options
- testing...

Initial Course Plan - Part 2

(Up to Week 16 - 10th December 2015)

- Augment and develop initial app
- Explore other options
- publication frameworks
- further libraries and options
- tools and workflows
- visualisations, graphics...
- publish (again...)

Assignments and Coursework

Course will include

- weekly bibliography and reading (where applicable)
- weekly notes, examples, extras...

Coursework will include

- quizzes or group exercises at the end of each section (Total = 30%)
    - *based on course notes, reading, and examples*
- development and project assessment (Total = 70%)
    - *mid-semester assessment (Total = 30%)*
        - end of DEV week
        - demo due 15th October @ 4.15pm
    - *final assessment (Total = 40%)*
        - demo due 3rd December 2015 @ 4.15pm
        - report due 10th December 2015 @ 6.45pm

Quizzes, group exercises...

Course total = 30%

- at least one week notice before quiz
  - *average time ~30 minutes (can be extended...)*
  - *taken towards the end of class*
- group exercises
  - *help develop course project*
  - *test course knowledge at each stage*
  - *get feedback on project work*

Development and Project Assessment

Course total = 70% (Parts 1 and 2 combined total)

Initial overview

- combination project work
    - *part 1 = mid-semester **DEV Week** work (30%)*
    - *part 2 = final demo and report (40%)*
- group project (max 5 persons per group)
- design and develop a web app
    - *purpose, scope etc is group's choice*
    - ***no** blogs, to-do lists, note-taking...*
    - *chosen topic requires approval*
    - *must implement data from either self-hosted data, public API, or combination of both*

DEV Week Assessment

- web app developed from scratch
  - *examples, technology etc outlined during weeks 1 to 6*
- demo and project report
  - *week 8 - 15th October 2015*
- anonymous peer review
  - *similar to user comments and feedback*
  - *chance to respond to feedback before final project*

Final Assessment

- working final app
- presentation can be a live demo, slides, video...
    - *week 15 - 3rd December 2015*
    - *show and explain implemented differences from DEV week project*
    - *where and why did you update the app?*
    - *benefits of updates?*
- how did you respond to peer review?
- final report
    - *due week 16 - 10th December 2015 @ 6.45pm*

Goals of the course

A guide to developing and publishing interactive client-side web applications and publications.

Course will provide

- guide to developing client-side web applications from scratch
- guide to publishing web apps for public interaction and usage
- best practices and guidelines for development
- fundamentals of web application development
- intro to advanced options for client-side development
- ...

Course Resources

Website

Course website is available at
https://csteach424.github.io

- timetable
- course overview
- course blog
- weekly assignments & coursework
- bibliography
- links & resources
- notes & material

GitHub

Course repositories available at
https://github.com/csteach424

- weekly notes
- examples
- source code (where applicable)

Intro to Client-side web design

- allows us to design and develop online resources and publications for users
  - *both static and interactive*
- restrict publication to content
  - *text, images, video, audio...*
- develop and publish interactive resources and applications
- *client-side scripting* allows us to offer
  - *interactive content within our webpages and web apps*
- interaction is enabled via code that is downloaded and compiled, in effect, by the browser
- such interaction might include
  - *a simple mouse rollover or similar touch event*
  - *user moving mouse over a menu*
  - *simple but effective way of interacting*

Client-side and server-side - Part 1

Client-side

- scripts and processes are run on the user's machine, normally via a browser
    - *source code and app is transferred to the user's machine for processing*
- code is run directly in the browser
- languages include HTML, CSS, and JavaScript (JS)
    - *HTML = HyperText Markup Language*
    - *CSS = Cascading Style Sheets*
    - *many compilers and transpilers now available to ease this development*
    - *eg: Go to JavaScript...*
- reacts to user input
- code is often visible to the user (source can be read in developer mode etc...)
- in general, cannot store data beyond a page refresh
- in general, cannot read files directly from a server (HTTP requests required)
- single page apps create rendered page for the user

Client-side and server-side - Part 2

Server-side

- code is run on a server
    - *languages such as PHP, Ruby, Python, Java, C#...*
    - *in effect, any code that can run and respond to HTTP requests can also run a server*
- enables storage of persistent data
    - *data such as user accounts, preferences...*
- code is not visible to the user
- responds to HTTP requests for a given URL
    - *not direct user input of any kind*
- can render the view for the user on the server side

and so on...

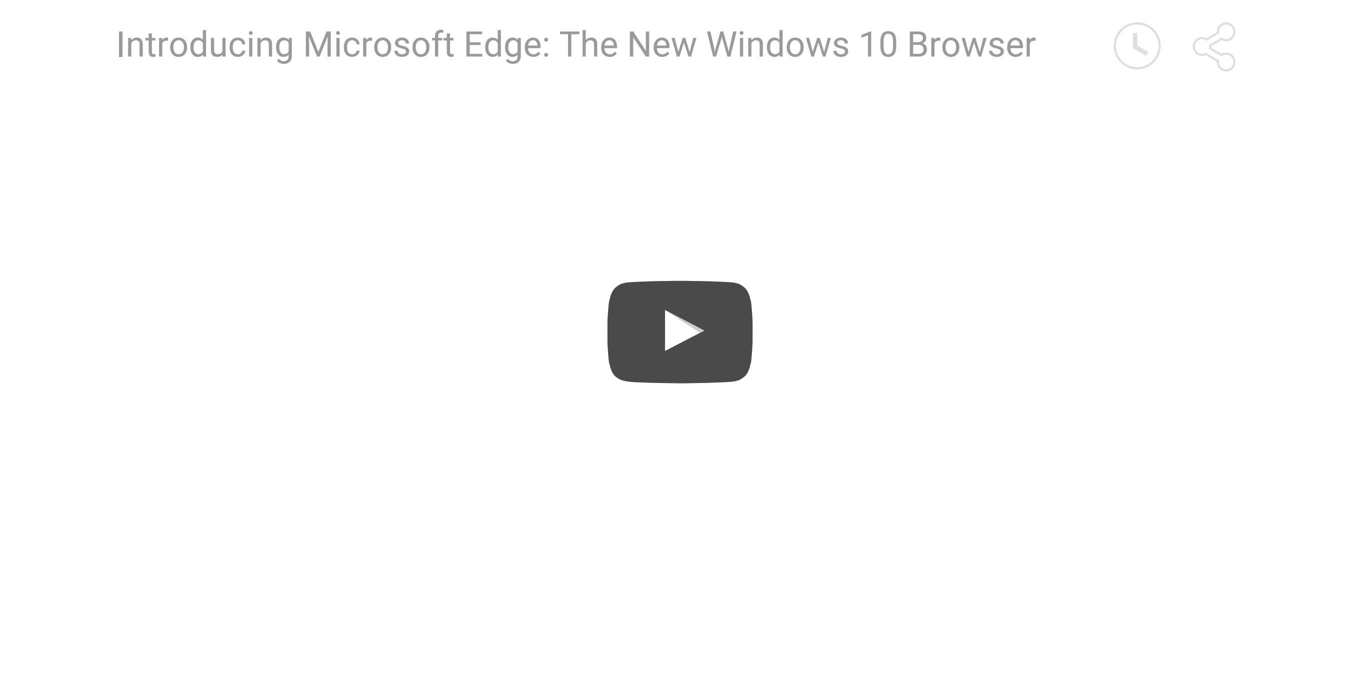
Getting started

- basic building blocks include HTML, CSS, and JS
- many tools available to work with these technologies
- three primary tools help with this type of development
- web browser
    - *such as Chrome, Edge (IE?), Firefox, Opera, Safari...*
- editor
    - *such as Atom, Sublime, Visual Studio Code...*
- version control
    - *Git, Mercurial, Subversion*

Getting started - Web Browsers

- choose your favourite
  - *Chrome, Firefox, Safari, Edge...*
  - *not IE*
- developer specific tools
  - *Chrome etc view source, developer tools, JS console*
  - *Firefox also includes excellent developer tools*
  - *Firebug*
- cross-browser extension for web developers
  - *Web Developer*

Video - Microsoft Edge



Source - YouTube - Introducing Microsoft Edge

Getting started - Editors

Many different choices including

Linux, OS X, and Windows

- Atom
- Sublime
- Visual Studio Code
  - ***NB:*** *in preview, but interesting to test*

OS X specific

- BBEdit
  - *TextWrangler*

and so on.

Video - Atom 1.0

Source - YouTube - Introducing Atom 1.0

Browser technologies

- browser rendering engines
- web standards
  - *HTML*
  - *CSS*
  - *XML*
  - *XHTML*
- application foundations
- open web platform

Browser rendering engines

- Until 2013, *WebKit* was the default rendering engine for both Safari and Chrome
- Google switched to the open source alternative, *Blink*, whilst Safari continues to use *WebKit*
- Firefox continues to use the *Gecko* rendering engine
- Microsoft's new Edge browser uses a new proprietary engine called *EdgeHTML*
    - *fork of the Trident rendering engine*
    - *Microsoft notes that EdgeHTML will largely behave like Chrome and Safari*

Web standards

- many disparate web standards
    - *include the broader internet beyond WWW...*
    - *subset of particular interest to web developers*
- primary web standards
    - ***Recommendations** published by the W3C (World Wide Web Consortium)*
    - ***Unicode** standards published by the Unicode Consortium*
    - ***ECMA** standards now published by ECMA International*
    - *examine with React etc*

W3C Recommendations

Recommendations of the W3C of particular interest includes

- HTML (HyperText Markup Language)
  - *key building block of the web*
  - *stored as plain text*
  - *includes selection of tags*
  - *eg: headings, images, links, lists, paragraphs, tables...*
- CSS (Cascading Style Sheets)
  - *commonly used with HTML*
  - *controls rendering and stylistic characteristics of a web page*
  - *CSS concerned with presentation of the structure and data*
- XML (Extensible Markup Language)
  - *often considered a meta-language*
  - *follow-on from SGML*
  - *used to describe data & not presentation, rendering of data*
  - *element tags not inherently pre-defined*
  - *foundation for many XML languages such as RSS, MathML, MusicML...*
- XHTML (Extensible HyperText Markup Language)
  - *attempt to update and rewrite HTML based on experience from XML*
  - *very similar to HTML with stricter rules*
  - *eg: HTML lapse in enforcing case sensitivity, closing tags...*
  - *strict rules structure inherited from XML style languages*

Video - W3C Web standards for the future

Web standards for the future

from W3C PLUS

Source - Vimeo - W3C

Application foundations - Part 1

W3C, on the occasion of HTML5 achieving the status of W3C Recommendation, proposed

> *a set of technologies for developing distributed applications with the greatest interoperability in history. Application Foundations for the Open Web Platform*

- known as the OWP (Open Web Platform)
- driven by a blog post by Jeff Jaffe in October 2014
    - *suggested W3C's next priority should be Open Web Platform*
    - *OWP should be easier to use for developers*

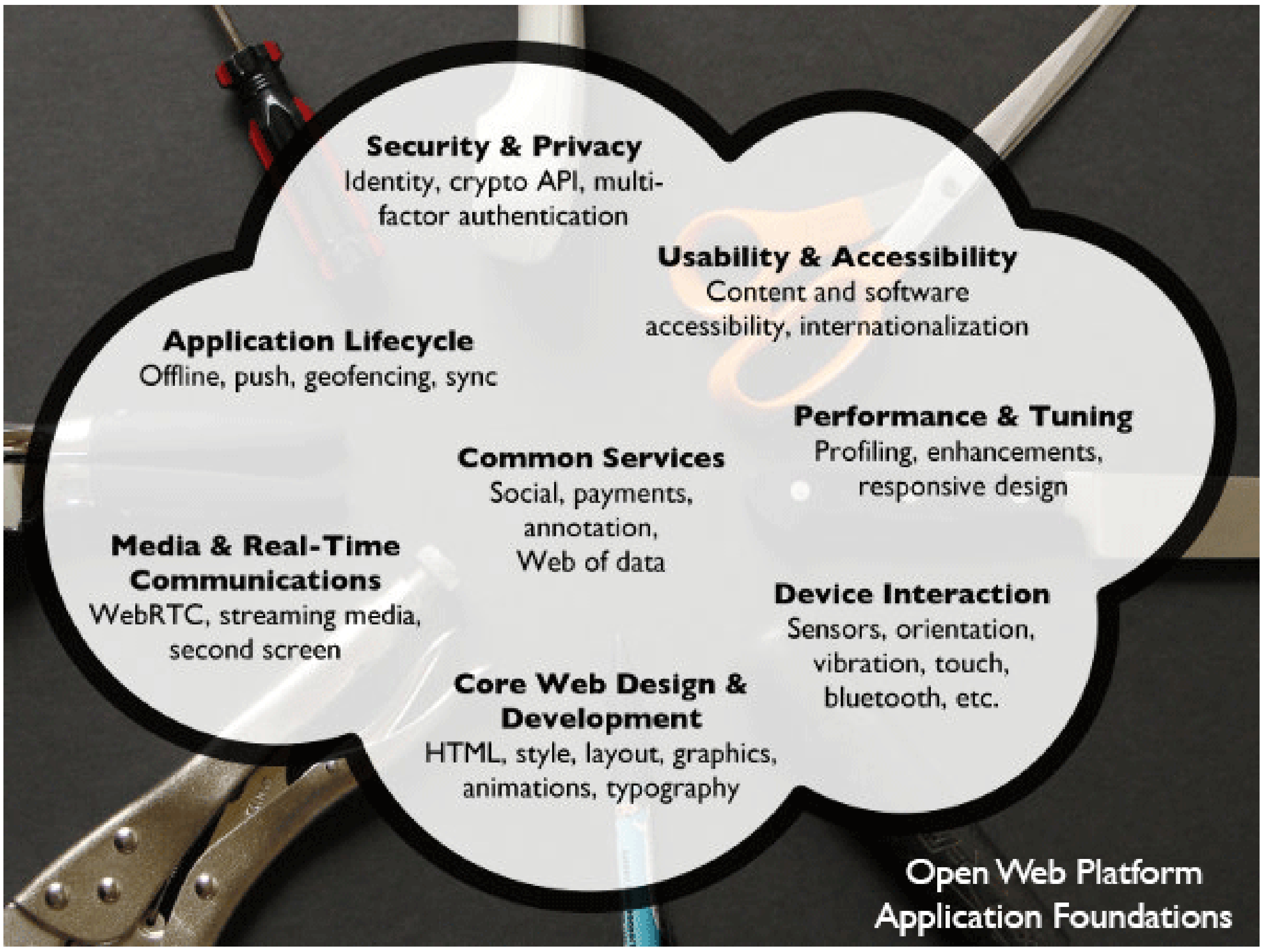
Application foundations - Part 2

Jaffe defined eight **Foundations** in that particular post, which include the following

- Security and Privacy
- Core Web Design and Development
- Device Interaction
- Application Lifecycle
- Media and Real-Time Communications
- Performance and Tuning
- Usability and Accessibility
- Services

Further information and updates can be found at the W3C's App Foundations website.

Image - Open Web Platform


Security & Privacy
Identity, crypto API, multi-factor authentication
Usability & Accessibility
Content and software accessibility, internationalization
Application Lifecycle
Offline, push, geofencing, sync
Performance & Tuning
Profiling, enhancements, responsive design
Common Services
Social, payments, annotation, Web of data
Media & Real-Time Communications
WebRTC, streaming media, second screen
Device Interaction
Sensors, orientation, vibration, touch, bluetooth, etc.
Core Web Design & Development
HTML, style, layout, graphics, animations, typography
Open Web Platform
Application Foundations

Source - W3C

Version control

- what is version control?
- setting up Git
- simple command-line usage
- Git basics

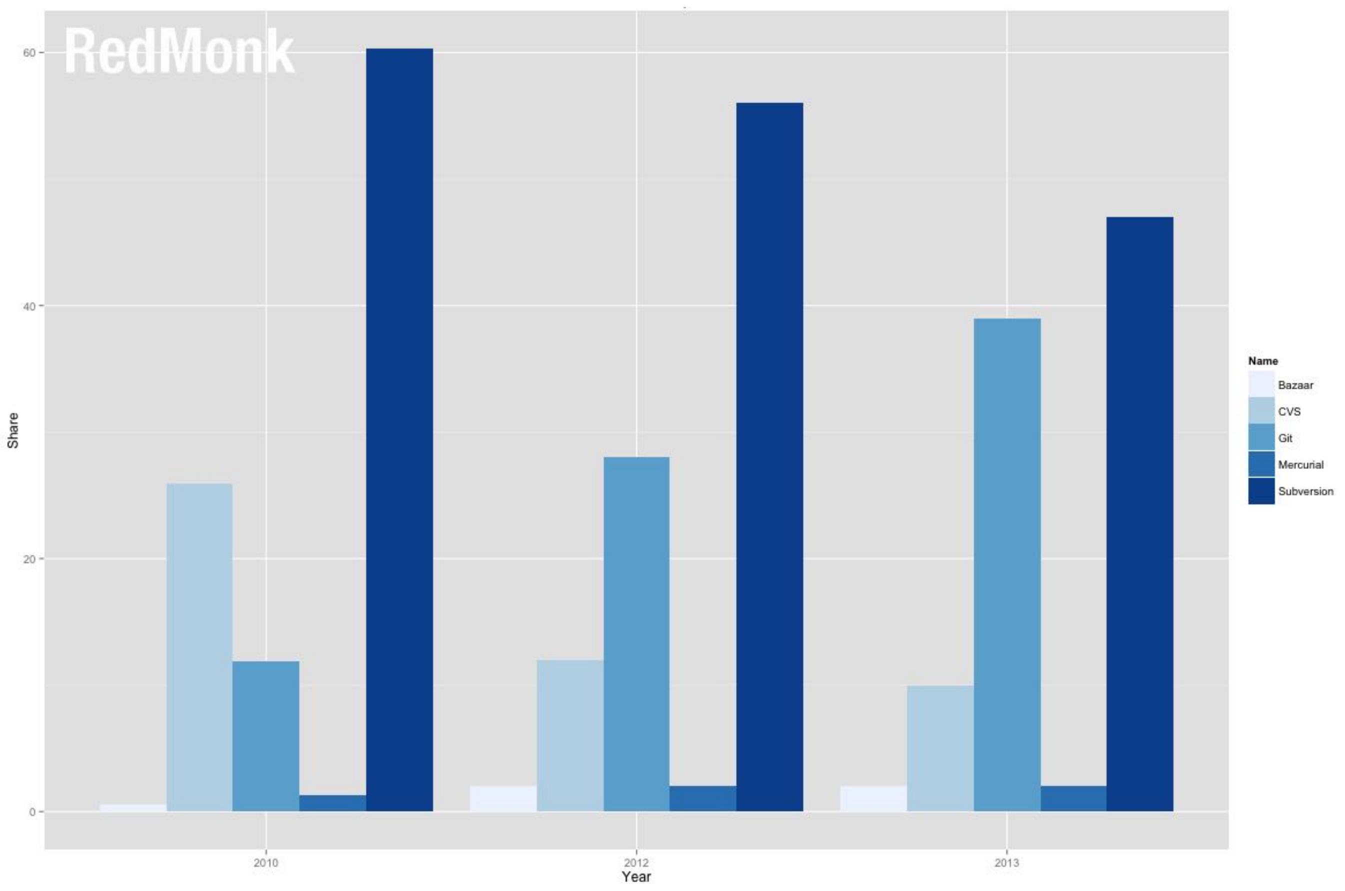
Version control - intro

- ensure we keep track of changes, updates, contributions, suggested mods...
- could try old, and error-prone, tracking of directories etc
- *version control* tool such as **Git**
- coding style known as *exploratory coding*
    - *researching, learning, checking what does and does not work correctly...*
    - *often used methodology for coders, and small groups as well*
- can lead to many changes and updates in code

Version control - what is version control?

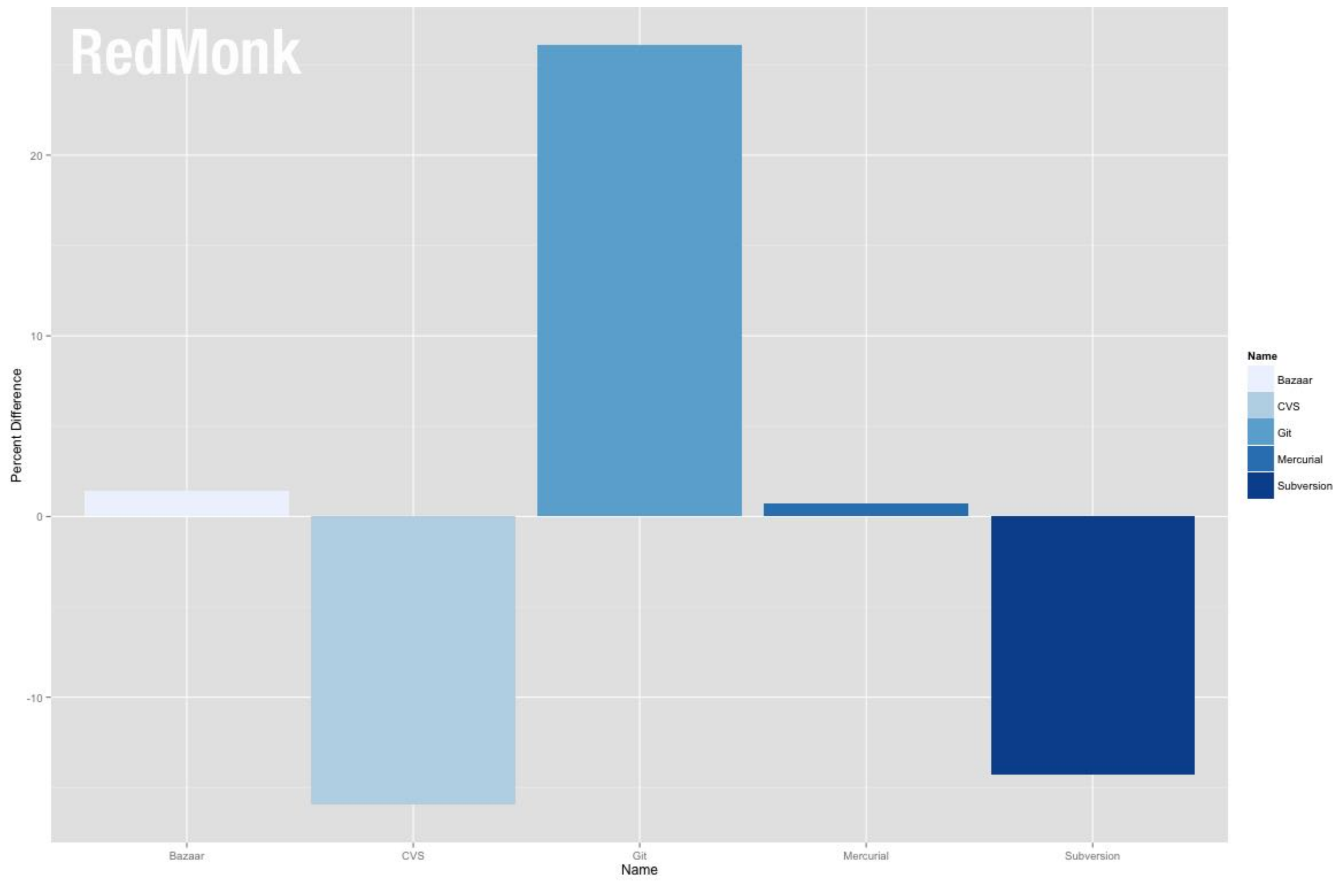
- very basic form of version control used on a regular basis
  - *copying, replicating folders, documents etc*
- compare updates between old and new copies & revert back to older version
  - *very basic form of version control*
- software development version control
  - *maintain defined, labelled points in our code*
  - *easily refer back to them or revert to an earlier state if needed*
  - *important tool for collaborative work with other developers*
- number of different, and popular, version control tools over the last few years
  - *Subversion, Mercurial, Git...*
- by 2010 Subversion held approximately 33.4% of the market for version control
  - *Git is believed to have only held approximately 2.7%, and Mercurial a paltry 0.7%*
- by 2013, Git usage was almost as high as Subversion, and it continues to grow in popularity
- Git's popularity largely due to preference for distributed version control
  - *Atlassian's switch from Subversion to Git in 2012 also helped*

Image - Version control usage (2010-2013)



Source - RedMonk

Image - Version control change in usage (2010-2013)



Source - RedMonk

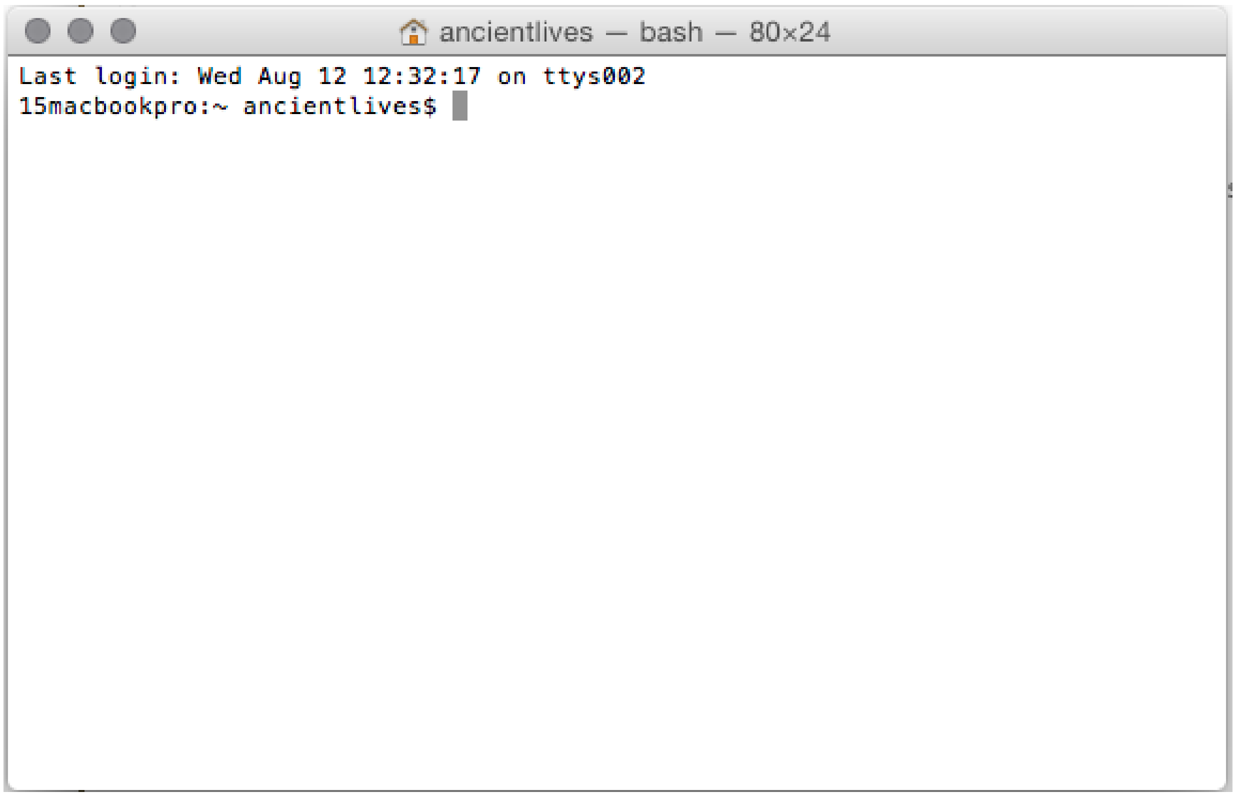
Version control - setting up Git

- simple installers available for Git
- choose platform's installer from
    - *git*
    - *follow simple instructions to install*
- full install instructions for various Linux distributions
    - *git - Linux downloads*
- for Debian/Ubuntu based APT distributions
    - `apt-get install git`

Version control - Git GUIs

- many GUIs available for working with Git
  - *Git GUIs*
- including specific GUIs for GitHub
  - *GitHub desktop clients*
- still beneficial and quicker to work with command-line
  - *quick and easy to navigate files, directories...*
  - *work with Git and version control in general*

Image - OS X Terminal application



```
Last login: Wed Aug 12 12:32:17 on ttys002
15macbookpro:~ ancientlives$ 
```

Command-line - Navigating directory and files

A few examples

- check the current directory (pwd = *print working directory*.)

```
pwd
```

- check the contents of the current directory (lists working directory)

```
ls
```

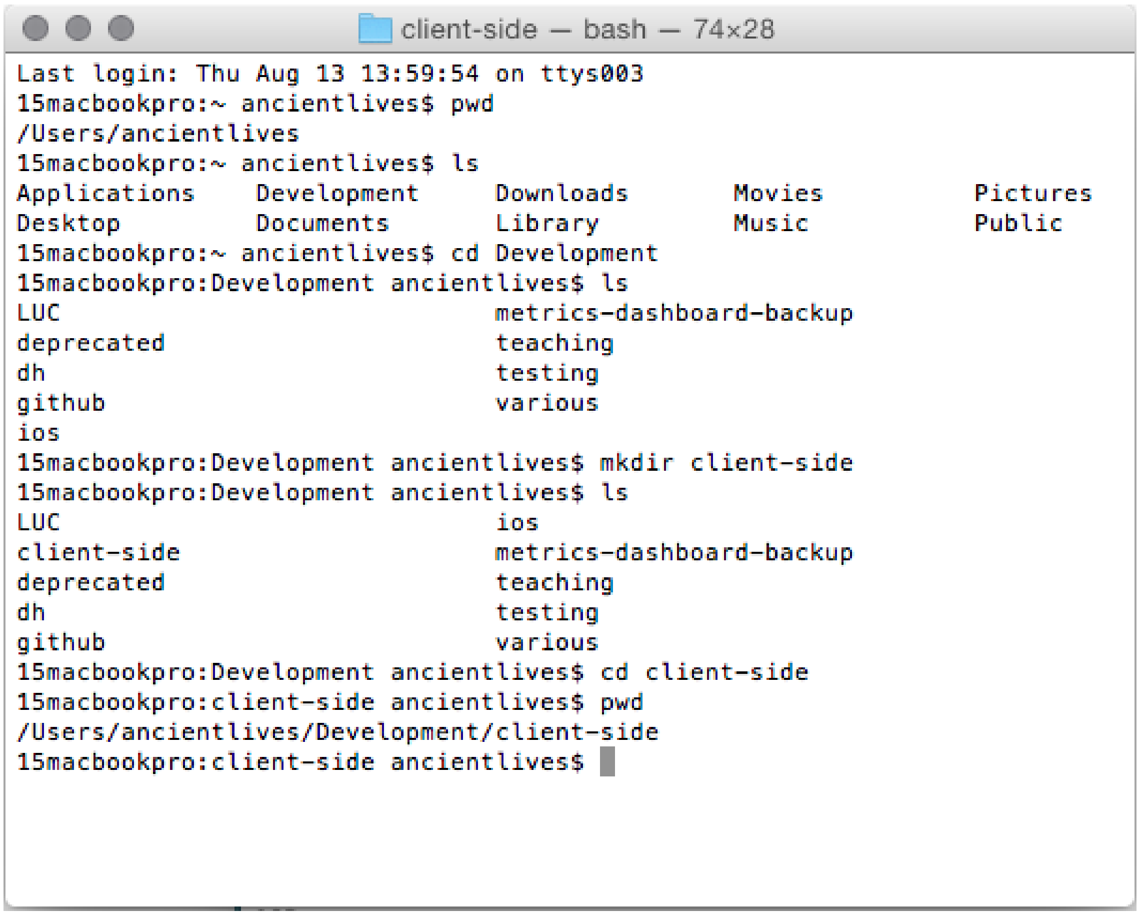
- this command allows us to change directory

```
cd
```

- in the working directory, we can create a new directory

```
mkdir
```

Image - Command-line examples



```
Last login: Thu Aug 13 13:59:54 on ttys003
15macbookpro:~ ancientlives$ pwd
/Users/ancientlives
15macbookpro:~ ancientlives$ ls
Applications    Development     Downloads       Movies          Pictures
Desktop         Documents       Library         Music           Public
15macbookpro:~ ancientlives$ cd Development
15macbookpro:Development ancientlives$ ls
LUC                             metrics-dashboard-backup
deprecated                      teaching
dh                              testing
github                          various
ios
15macbookpro:Development ancientlives$ mkdir client-side
15macbookpro:Development ancientlives$ ls
LUC                             ios
client-side                     metrics-dashboard-backup
deprecated                      teaching
dh                              testing
github                          various
15macbookpro:Development ancientlives$ cd client-side
15macbookpro:client-side ancientlives$ pwd
/Users/ancientlives/Development/client-side
15macbookpro:client-side ancientlives$ 
```

Git basics - Part I

Configure user/developer details

- set details for *username* and *user email*
    - *global flag can set these details for all work within our local instance of Git*

```
git config --global user.name "424dev"
```

- set preferred email address

```
git config --global user.email "424dev@gmail.com"
```

- override for a specific repository in Git by omitting `--global` flag

```
git config user.name "424dev-single"
```

and the same principle applies for email.

- check correct username for current repository

```
git config user.name
```

Git basics - Part 2

Tracking projects

- start tracking project with Git
  - *create new working directory (eg: at `root` of our home directory)*

```
cd ~/
```

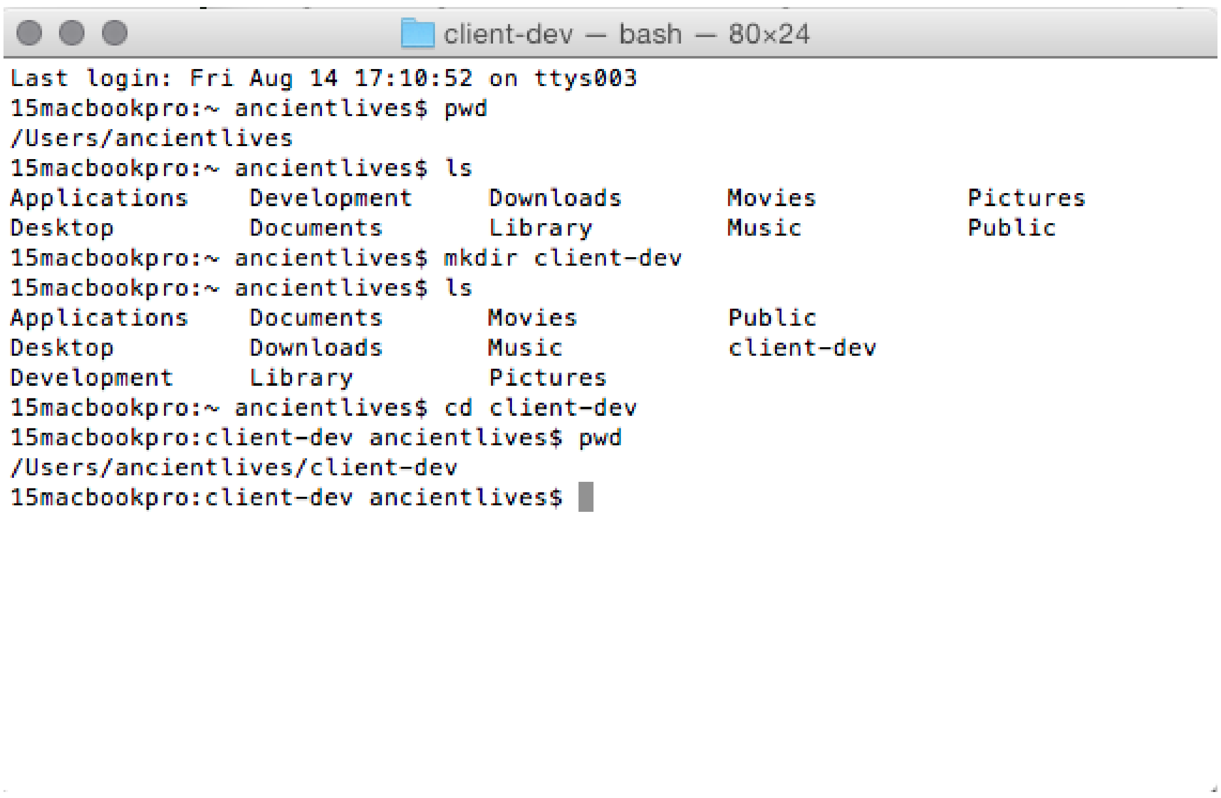
- ensures we have changed to our home directory. Then check working directory,

```
pwd
```

and then make a new directory for our client-side development.

```
mkdir client-dev
```

Image - creating a *client-dev* directory



```
Last login: Fri Aug 14 17:10:52 on ttys003
15macbookpro:~ ancientlives$ pwd
/Users/ancientlives
15macbookpro:~ ancientlives$ ls
Applications    Development     Downloads       Movies          Pictures
Desktop         Documents       Library         Music           Public
15macbookpro:~ ancientlives$ mkdir client-dev
15macbookpro:~ ancientlives$ ls
Applications    Documents       Movies          Public
Desktop         Downloads       Music           client-dev
Development     Library         Pictures
15macbookpro:~ ancientlives$ cd client-dev
15macbookpro:client-dev ancientlives$ pwd
/Users/ancientlives/client-dev
15macbookpro:client-dev ancientlives$ 
```

Git basics - Part 3

Add version control using *Git* to working directory

- initialise our new repository in the working directory

```
git init
```

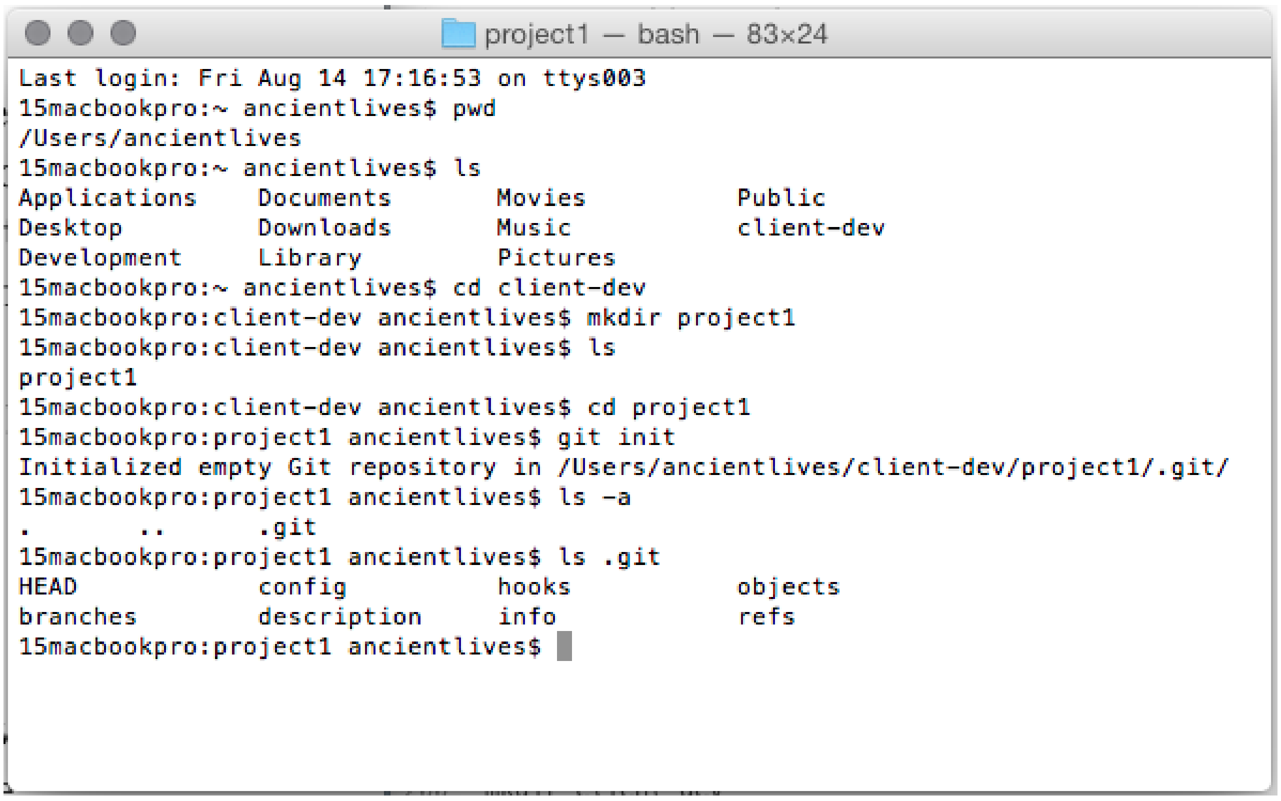
- check hidden `.git` directory has been created

```
ls -a
```

- and show contents of the `.git` directory

```
ls .git
```

Image - Initialise new Git repository



project1 — bash — 83×24

```
Last login: Fri Aug 14 17:16:53 on ttys003
15macbookpro:~ ancientlives$ pwd
/Users/ancientlives
15macbookpro:~ ancientlives$ ls
Applications    Documents       Movies          Public
Desktop         Downloads       Music           client-dev
Development     Library         Pictures
15macbookpro:~ ancientlives$ cd client-dev
15macbookpro:client-dev ancientlives$ mkdir project1
15macbookpro:client-dev ancientlives$ ls
project1
15macbookpro:client-dev ancientlives$ cd project1
15macbookpro:project1 ancientlives$ git init
Initialized empty Git repository in /Users/ancientlives/client-dev/project1/.git/
15macbookpro:project1 ancientlives$ ls -a
.       ..      .git
15macbookpro:project1 ancientlives$ ls .git
HEAD            config          hooks           objects
branches        description     info            refs
15macbookpro:project1 ancientlives$
```

Git basics - Part 4

Start using our new repository

- create an initial `index.html` file in project's working directory
    - *create using preferred text editor or command-line, eg:*

```
touch index.html
```

- save new file, and check *Git* is correctly tracking our project

```
git status
```

- outputs current status of working branch, defaults to `master`
    - *outputs we have `untracked files`*
    - *files will include new `index.html`*
- add any new untracked file/s

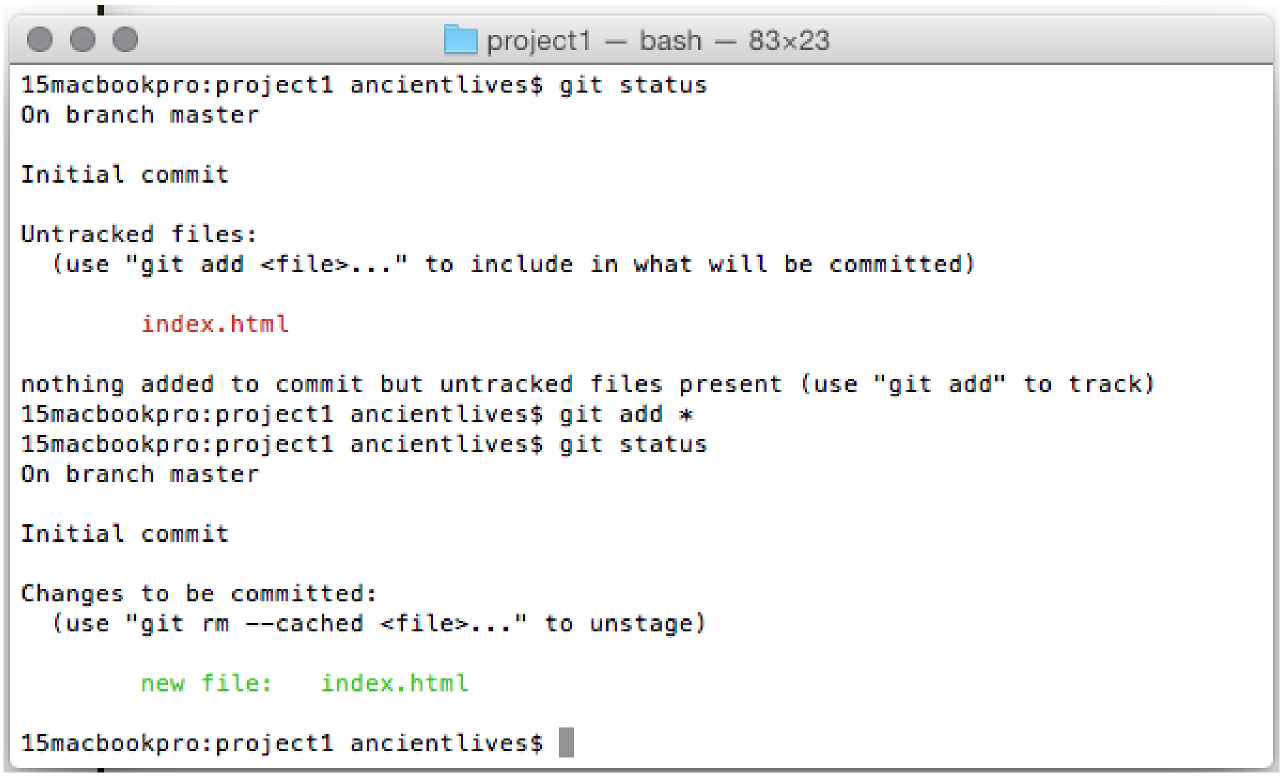
```
git add index.html
```

or

```
git add *
```

for all untracked files.

Image - Git status and add



```
15macbookpro:project1 ancientlives$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        index.html

nothing added to commit but untracked files present (use "git add" to track)
15macbookpro:project1 ancientlives$ git add *
15macbookpro:project1 ancientlives$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   index.html

15macbookpro:project1 ancientlives$
```

Git basics - Part 5

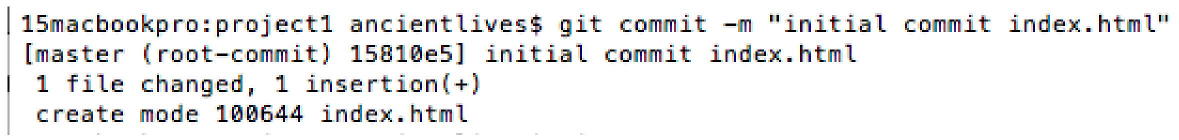
After adding our new `index.html` file to the repository, we can commit these changes to the initial state of the repository.

We use the following command

```
git commit -m "initial commit index.html"
```

- `–m` flag permits useful message for commit
    - *message added within quotation marks*
    - *should be useful and present tense*

Image - First commit and message



```
15macbookpro:project1 ancientlives$ git commit -m "initial commit index.html"
[master (root-commit) 15810e5] initial commit index.html
 1 file changed, 1 insertion(+)
 create mode 100644 index.html
```

Git basics - Part 6

- initial commit has saved a defined point in our work
    - *one we can revert to if needed*
- check `git status` and there should be nothing else to commit
    - *working directory should be clean*
- *Git* has set our files ready for tracking
- repeat this process as we make further changes and updates
    - *freeze defined points within our project*
- check recent commits, and view a record

```
git log
```

Git basics - Part 7

Git revisions

- we've seen *Git*'s simple linear commits
    - *presumes file has one parent*
    - *child commits detail this linear revision of content*
- a *Git* commit can store multiple parents and children
- eg: commit history might include
    - *revisions to a single file*
    - *addition or deletion of new files*
    - *merging of files to different branches*
    - *further additions...*

Git basics - useful commands

Git command	**Expected Outcome**
git config user.name "..."	sets username for current repo
git config --global user.name "..."	sets username for all repos (unless overridden per repo)
git config user.email "..."	sets user's email address for current repo
git config --global user.email "..."	sets user's email address for all repos (unless overridden per repo)
git init	initialise a Git repository in the current working directory
git status	output current status of repo in current working directory
git add "..."	define specific file in current repo for next commit
*git add **	define all files in current repo for next commit
git commit -m "..."	commit defined files (set using `git add`) with message
git log	output commit history for current repo

References

- Jaffe, Jim., *Application Foundations For The Open Web Platform*. W3C. 10.14.2014. http://www.w3.org/blog/2014/10/application-foundations-for-the-open-web-platform/