PRE-WORK: MOBILE DEVELOPMENT

GETTING STARTED WITH MOBILE DEVELOPMENT

Congratulations on joining us at General Assembly for the Mobile Development course! We are excited to have you and are looking forward to working with you all soon!

In order to help ensure your success in the course, we've made a pre-course checklist. Complete all the items on the list and when you arrive on your first day, you and your computer will be prepared for the first lesson. This should take you around 2–3 hours total to complete.

We can't tell you how excited we are to have you join the General Assembly community!

Happy Coding!

BEFORE CLASS CHECKLIST (2–3 HOURS)

COMPLETE ALL ITEMS BEFORE YOUR FIRST CLASS

☐ Upgrade your Mac to OS 10.10 (Yosemite)

☐ Install Xcode v 6.0+ from the Mac App Store

☐ Sign up for **a free Apple Developer Account**

☐ Complete required pre-course tutorials

PRE-COURSE TUTORIALS

Required Pre-Work:

• Complete the following lesson, **"Understanding Mobile Devices,"**

• Complete the **Git and GitHub tutorial** (attached)

Additional Pre-Work: Want to get a jump start on iOS, Swift and Mobile Development? Check out the resources linked to below. These aren't required but feel free to read through as many or as few as you like.

- **Xcode Overview:** Think of this as the user manual for Xcode—this will likely come in handy as you work through the assignments.
- **Swift Overview:** Learn about some of the features built into the Swift programming language.
- **Xcode Cheat Sheet:** A great cheat sheet to keep bookmarked for when you get deeper into iOS development.

SOFTWARE INSTALLATIONS

Please bring a laptop to every class. A Mac running OS 10.9 or higher is required for this course.

SOFTWARE		NOTES
If you haven't already, update your operating system: install OS 10.10 (Yosemite) via the Mac App Store	~60 min	OS 10.9 or 10.10 is required to install Xcode v 6.0
Install Xcode v 6.0+ from the Mac App Store	~ 60 min	Xcode is the IDE (integrated development environment) you'll be using to create iOS apps.

Git and GitHub

Overview

Our Mobile Development course will be using Git as our versioning control system. We will use GitHub as our interface to that system. This document will cover

1. Version control
2. The GitHub app
3. Common terms

Git is a tool software developers use to manage changes in code. Those changes may be your own, or may be made by members of your team.

November 30	fix navigating the page controller	Jeff Algera	162e7da224a799918970bec6b8775eeb80362ffb
November 30	User management	Jeff Algera	3279948c9c362b5905ba73e6aac88237095cdaba
November 21	Update from FB	Jeff Algera	a393c006ad955fd8b89198555de170f96dab0334
November 20	Linking FB friends	Jeff Algera	38244b656f5753d4a8018adc6f3f1f9e28ad180b
November 19	Loading data from FB	Jeff Algera	59472f4eabb9cb1dc6af4bea9450d205b7f753d7
November 19	Cleanup user manager	Jeff Algera	c103e4de342df61a6b1980de23f2715095fcafb2

A history of changes made in a Git repository

GitHub is a company that provides a really excellent interface for managing Git repositories.

Secret lair of GitHub in San Francisco.

[Video] Introduction to Git and Github

What Git and GitHub do, why you should use them, and how to get started:

References

GitHub http://github.com

Git according to Wikipedia http://en.wikipedia.org/wiki/Git_(software)

GitHub instructional videos published on YouTube https://www.youtube.com/user/GitHubGuides

Version Control

Version control provides a number of benefits to the developer. Here are some of those benefits:

1. History
2. Version Control
3. Branching
4. A Distributed System

Benefit 1: History

Imagine working on a piece of code. Let's call this version 1:

```
var eureka = "The meaning of life is the following number"
var meaning_of_life = 42
print("Guys, I've figured it out.  \(eureka) is \(meaning_of_life)!")
```

Then after a few days, you change your mind and update the code. Let's call this version 2:

```
var eureka = "The meaning of life is the following number"
var meaning_of_life = 99999
print("Guys, I've figured it out.  \(eureka) is \(meaning_of_life)!")
```

Some time passes, you close Xcode, shut down your computer and then you realize:

> The first version was right! What did I have?

With Git, you can have a historical record of all changes made in your code. You can then reference those changes at a later time.

```
var meaning_of_life = 42
```

Benefit 2: Working with a team

Now imagine you are working with a team of three programmers on an awesome new app. There is one Xcode project, three programmers and one manager.
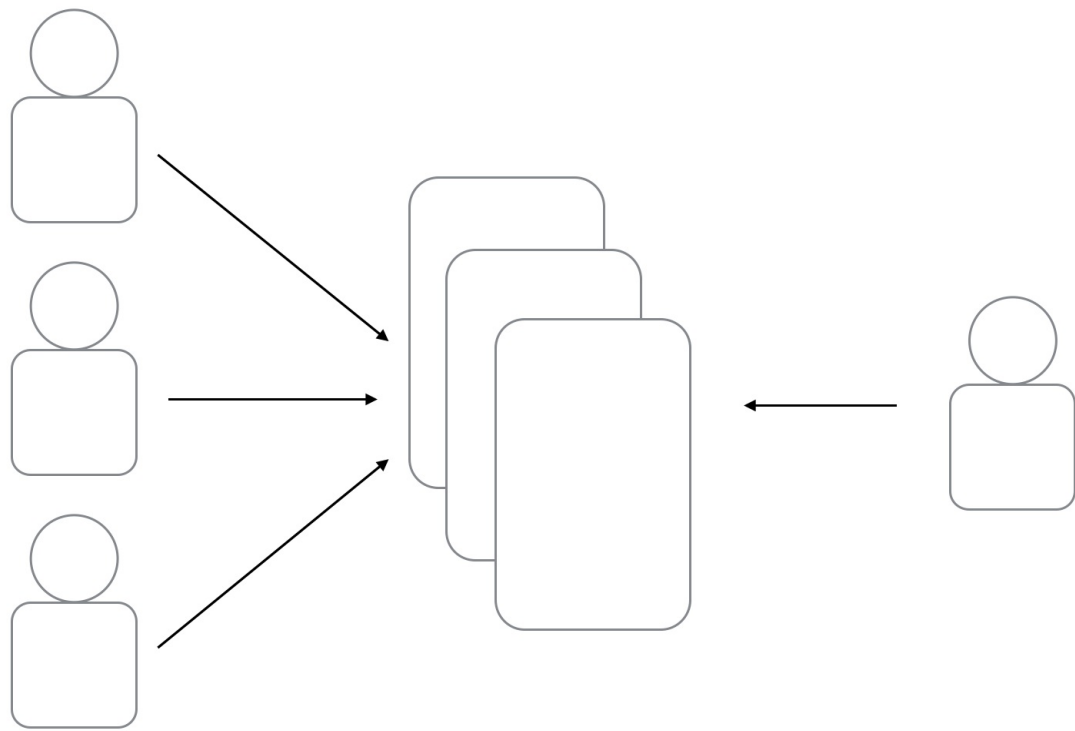
| Developers | Source Code | Manager |

The benefits you get with a version control system, such as Git, in this scenario:

1. Historical record of what each programmer was working on and when
2. The ability to go back in time and view changes from a different day for any of the programmers on the team
3. Automatic merging of changes from each developer into one central project
4. One central project that a manager can download and send off to the client.

Benefit 3: Branching

Branches are commonly used to segregate new code from the existing project. Let's say your manager decides she wants to incorporate a new feature into your awesome app and only wants that feature to be available to the rest of the developers when it's complete and ready.

Developers   Source Code
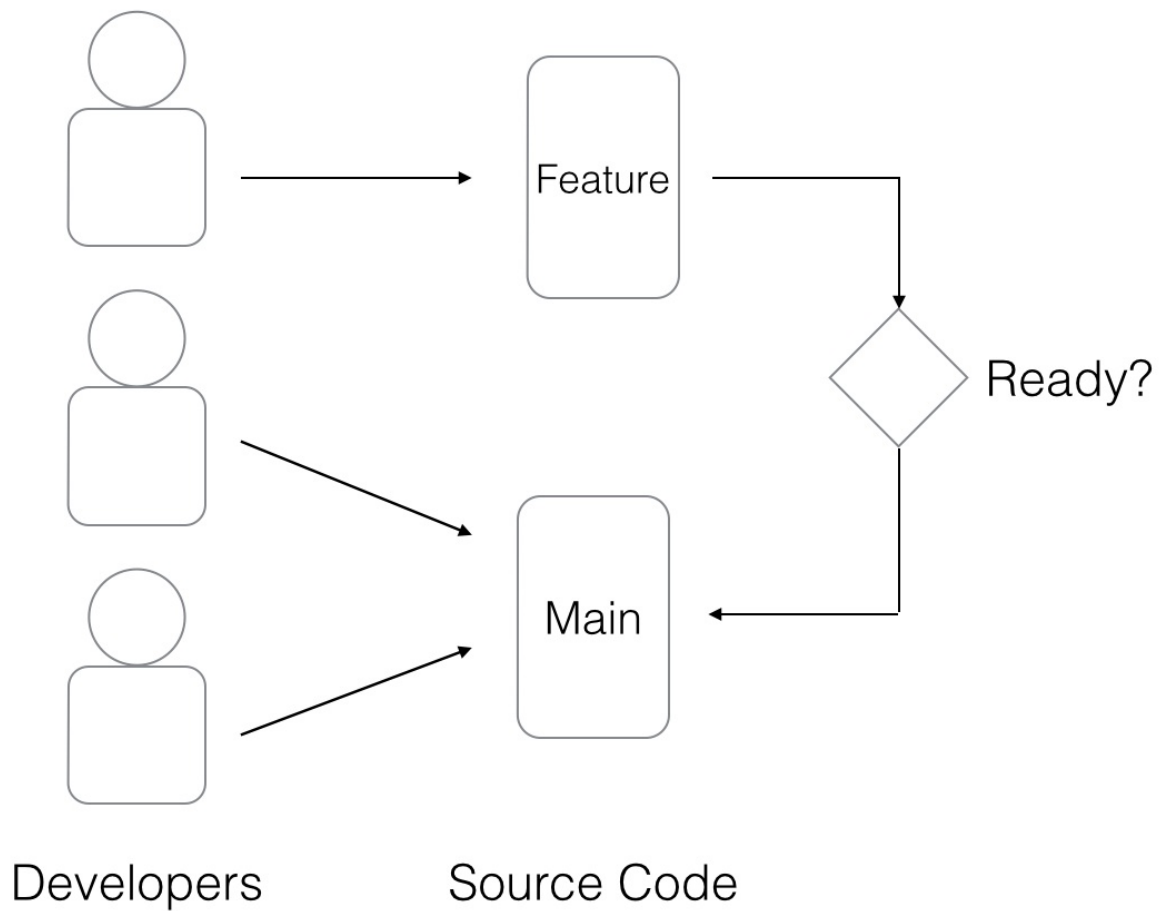
With Git you can work on your own branch and incorporate your changes at a later time, while retaining the other benefits of version control, as previously described.

Benefit 4: Local versus Remote - A Distributed System
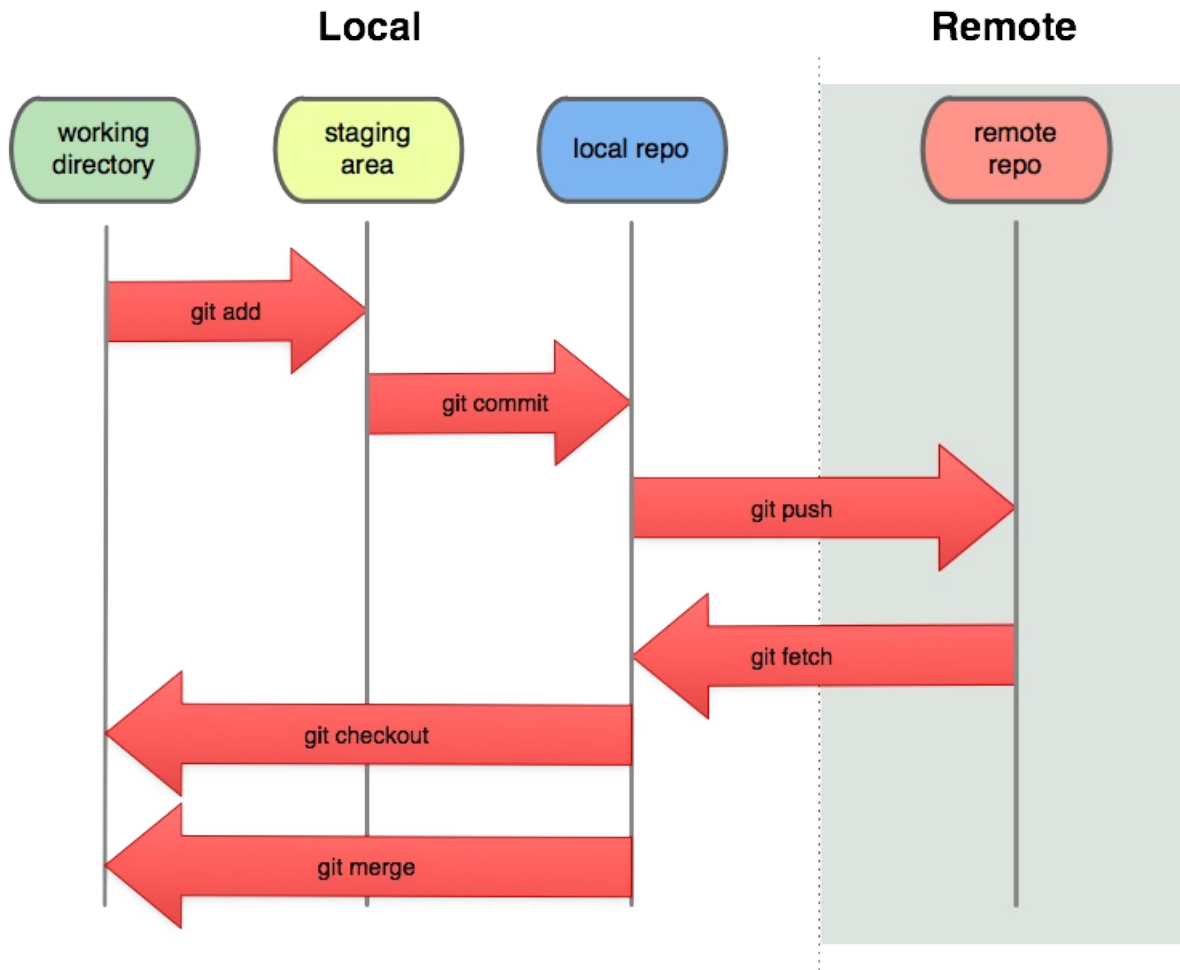
When you're working with Git, you are working with your own copy. This is what we refer to as your **local** copy. With your local copy, you can do whatever you want. Add some new buttons to your app, change todo en español, get dangerous and experiment and tinker throughout the app.

> This is a great way to learn and experiment

This is your local copy and you're not affecting others. You can commit changes to your local copy (even without Internet access) and decide to abandon it all and get a fresh copy from the remote.

When you're ready, you can publish your changes from your local copy into the remote repository.

References

Why Git? - http://www.markus-gattol.name/ws/scm.html

GitHub User Interface

GitHub provides two ways to use their service that we will use in our course. These are the web and the Mac app.

The Web

http://github.com

What you are seeing here is the remote representation of a repository. In detail:

1. Repo name and type - Describes who owns the repository, what the name of the repo is and whether the repo is public or private
2. A selectable overview of commits, branches, releases and contributors for a particular repository. Selecting any one of these options will bring a detail view of that selection.
3. A drop-down selection of the currently viewing branch. Changing the branch will update the source code to reflect that branch.
4. A selectable list of source code organized within the committed file structure.

One of the great features of the GitHub website, is the ability to drill down and view specific information about individual source code files including commit logs.
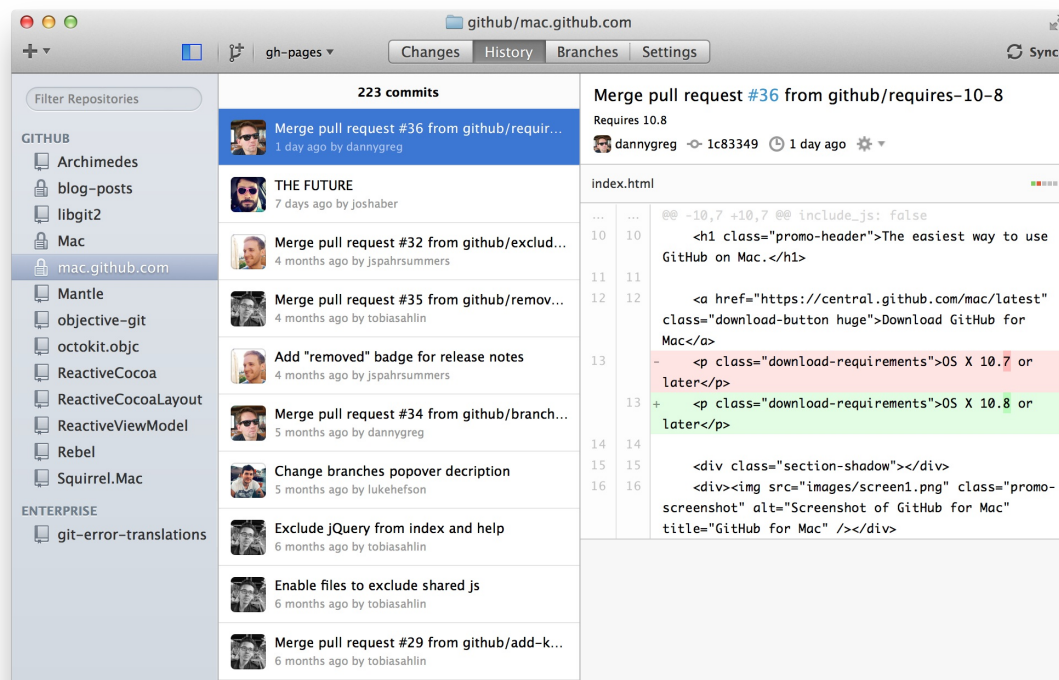
In the above graphic, we are looking at a blame of an individual source code file. Note the detail. You have a line-by-line listing of when that code was changed, by whom and a selectable commit entry to reference how that change was made. All items in blue are selectable.

The Mac App

https://mac.github.com
https://mac.github.com/help.html

Common Terms

After using Git for a number of years, developers have created a language around the tool and their process. This list can be used as a quick reference to decipher those messages and assimilate this language into your own.

Terms

- **Add** - Creating a new file within the repository
- **Blame** - A term used to find who changed what line of code and when.
- **Branch** - describes a separate line of development from another branch
- **Checkout** - retrieve the latest copy of a repository into your local branch
- **Clone** - to initially copy a remote repository onto your computer.
- **Commit** - to take your changes and submit them to the repository
- **Conflict** - when two commits affect the same code and Git has trouble automatically merging the two
- **Git** - pronounced with a hard "G", rhymes with spit
- **GitHub** - A company in San Francisco that made an excellent interface to Git repositories
- **Local** or **Local changes** - a cloned copy of a repository that may have changes not yet pushed to the remote repository.
- **Master** - Typically, the base branch of which all other brances are derived
- **Merge** - Take a series of commits from one branch and put them into another branch
- **Merge conflict** - Same as conflict
- **Publish** - To submit your changes to the remote repository. May also refer to the creation of a branch on the remote repository.
- **Push** - Same as publish without the creation connotation.
- **Pull** - To retrieve changes on a remote repository into your own local repo.
- **Repo** or **Repository** - Where the source code lives. For our purposes, this is GitHub
- **Stage** - The precursor to commit. Observe changes about to be commited into your branch before those changes go into the local copy of your repo.
- **Stash** - To quickly push aside your changes into a temporary state which can be retrieved later or abandoned.
- **Version Control** - A system that handles keeping track of changes and provides automation around merging changes from multiple streams of input, such as people.

References

GitHub Glossary - https://help.github.com/articles/github-glossary/

Git Manual (*Warning: Heavy material, approach with coffee*) http://gitref.org

Stack Overflow Q&A - http://stackoverflow.com/questions/7076164/terminology-used-by-git