Week Rap

Week 2 Patterns

- Crucial patterns for any language: Single Responsibility Pattern, Model View Controller, Lazy Loading, Concurrency, Callbacks, Encapsulation, Functional Programming, DRY, Rule of 3
- Crucial patterns for iOS: Delegation, Singletons, Custom Protocols
- .gitignore
- Networking Protocols: **HTTP, OAuth**

Week 2 Frameworks

- Objective-C System frameworks: Foundation, UIKit

Week 2 UI Techniques

- Storyboards
- Autolayout
- Labels, ImageViews, Buttons
- Collection Views
- Alert Controllers
- Animating Constraints
- General Segues
- **Custom Transitions**
- **Scroll Views**
- **Converting Rects/Points**

Week 2 Important Classes

- **NSURLSession**

Week 2 Swift features

- Optional Binding
- Type methods
- [weak self]
- Closure Expressions
- **Structs**

Using Git with a Team

Setting up your repo

1. **Don’t use Github’s app!!!!!!**
2. Create your project in Xcode
3. Init a git repo inside it’s directory
4. Setup a .gitignore and place it in the directory
5. Stage the .gitignore FIRST, with ‘git add .gitignore’
6. Stage & commit everything
7. Setup your remote repo on github, add it as a remote in your local git repo, and then push it up

So what should your .gitignore contain?

.gitignore

- There are a number of files generated by Xcode that you are going to want to put in your .gitignore in order to have a smooth source control experience
- The easiest way to setup your .gitignore file is to go to https://www.gitignore.io and have it generate a Swift .gitignore for you

Getting others on your repo

1. On github, the original owner needs to add all of the team members as collaborators (they now have read/write access)
2. Now the collaborators can clone the repo down to their local machine
3. The clone command automatically hooks up the remote repo as origin
4. Begin working

Git team workflow

1. When you are going to start work on a new feature, create a new branch
2. Do your work in that branch while periodically pulling from master to ensure your code works with the latest changes to master
3. When you are ready to push your changes to master, do one final pull from master to resolve any merge conflicts.
4. Push up to your remote feature branch, and then initiate a pull request to master
5. Have someone review the changes, and then accept or reject the pull request
6. Rinse and repeat

Do your pushing and pulling in Xcode, since Xcode has a great merge tool

Merge Conflicts

- Occasionally the merge process wont go as smoothly as we think it will.
- If you changed the same part of a file on the two branches you are merging, this will be a merge conflict.
- git will tell you theres conflicts in specific files, that the merge failed, and to fix the conflicts and then commit the results.
- Essentially git pauses the merge process until the conflicts are resolved.
- At anytime during a halted merge, you can run git status to see which files are still unresolved.

Resolving conflicts

- There are 2 ways to resolve the conflicts.
- Manually: Open each conflicted file and fix the conflicts line by line.
- Merge Tool: Use a merge tool that lets you choose which file's version of the conflicted code you want. This way is much less error prone. Xcode has a built in merge tool.

Manual Resolution

- Git adds conflict-resolution markers to the files that have conflicts.
- Heres what they will look like when you open them manually:

```
<<<<<<< HEAD
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> iss53
```

- The <<<<< HEAD denotes this is the beginning of the code that our local HEAD branch contains.
- The ====== signifies of the end of HEAD's version and the beginning of the branch we are trying to merge from.
- Finally, the >>>>>>iss53 signifies the end of the version of the code branch iss53 had
- Once we get rid of all the conflict markers (<<<<,======,>>>>>) in a file, we are ready to mark this file as resolved.
- Run git add on the file to mark it as resolved. staging the file tells git the conflicts have been resolved.

Merge Tool

- You can use a merge tool for a graphical interface based conflict resolution process
- use the git mergetool command to fire up the appropriate merge tool
- opendiff is the default merge tool if you havent configured git to use a different one.

Xcode's pbxproj and git

- pbxproj is a file that is contained in your Xcode projects
- it manages the file structure of your project
- so anytime a team member adds, removes, or rearranges files in your Xcode project, pbxproj changes
- git cannot automatically merge pbxproj, and sometimes even its merge tool will crash Xcode while you are trying to resolve the merge conflicts
- If that happens, you will have to manually resolve the conflicts by opening the pbxproj and removing the conflict markers yourself

Demo