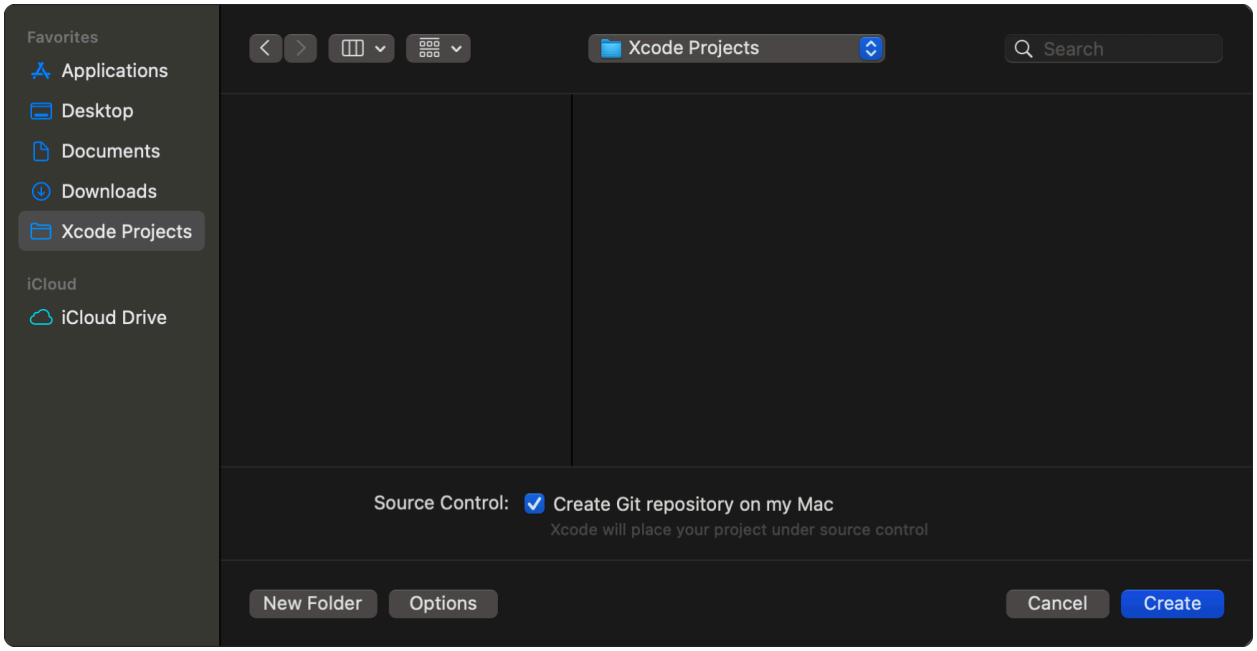
SHPE-UF-Mobile-iOS Onboarding

Overview

This document explains how to start a new project or work on an existing one in Xcode, and how to manage your project's code using Git and Github. It describes the different parts of the project, like the Data Layer and UI Layer, and how to organize your work and collaborate with others.

Setting Git with a new Project

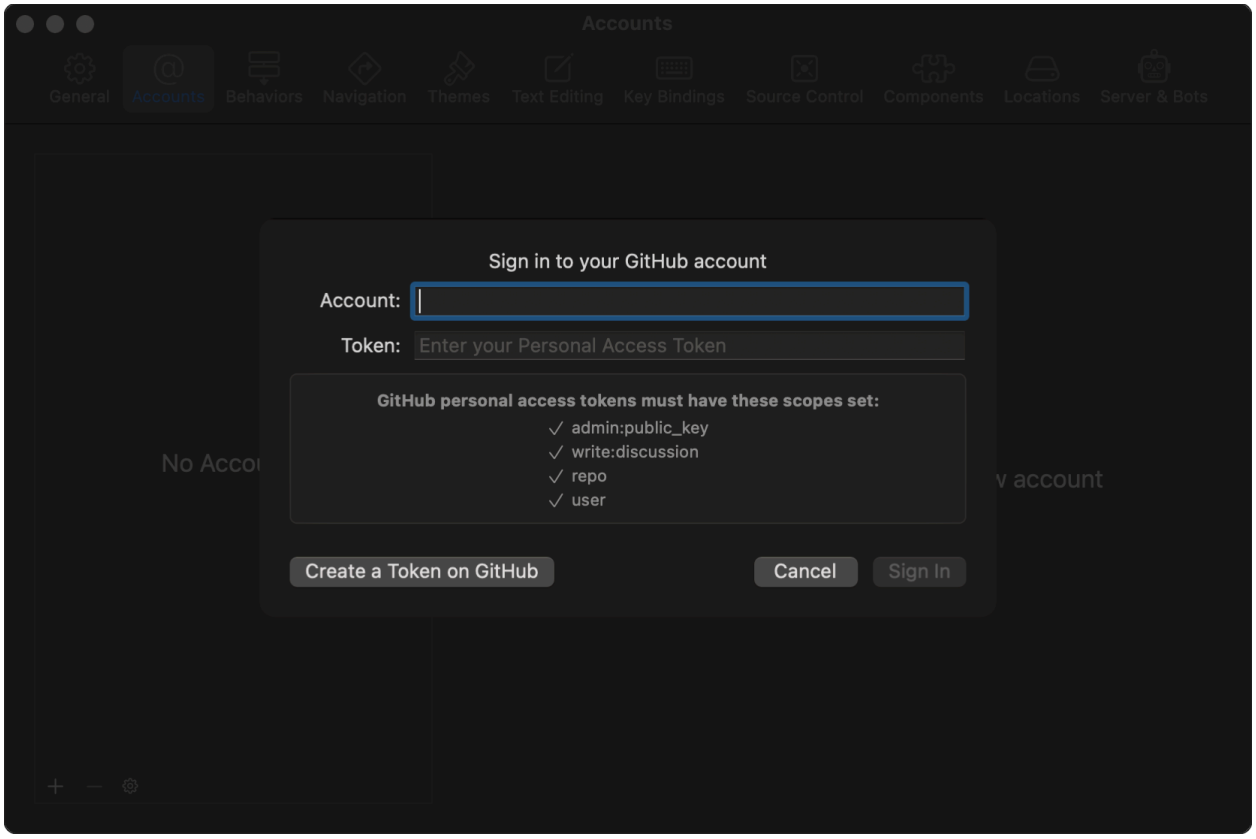
When you begin a new Xcode project, you'll choose where to save your project files. During this step, you have the option to establish a local Git source control repository by selecting "Create Git repository on my Mac" and then clicking the Create button.



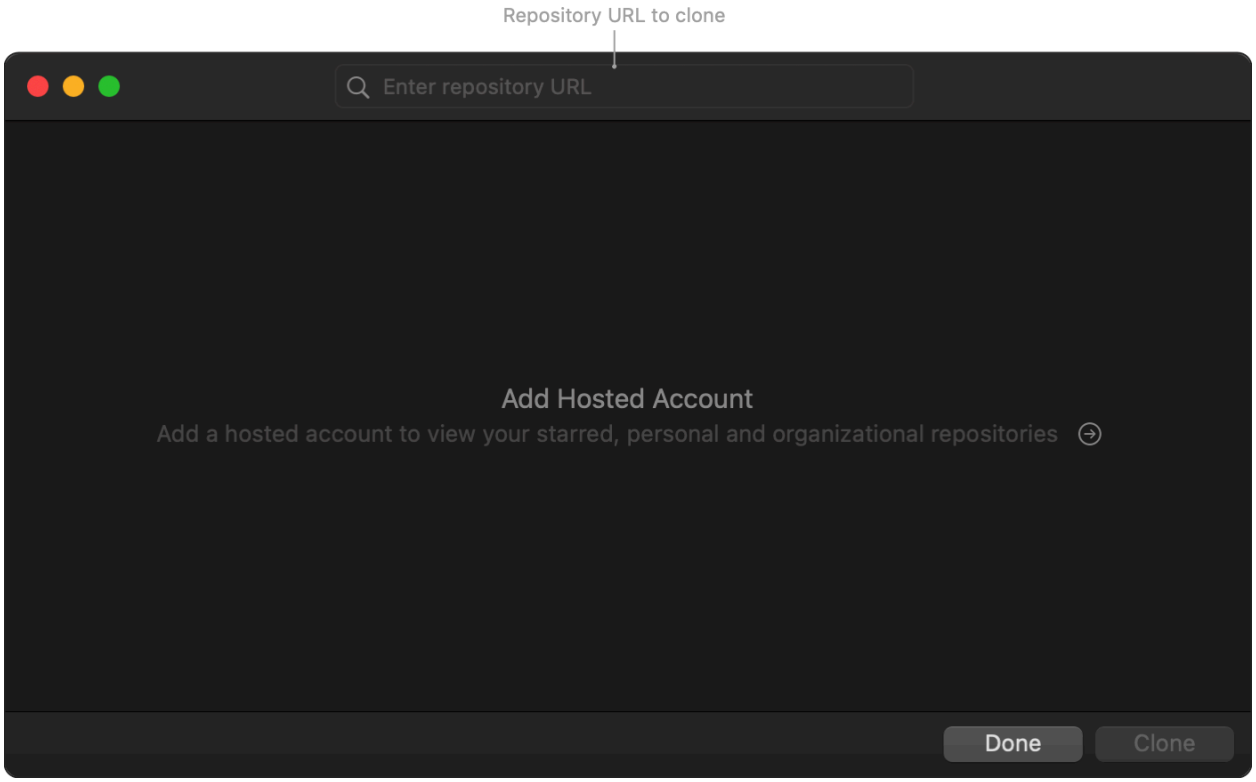
This action sets up your project in your chosen folder, initiates a local Git repository, and saves an initial batch of files.

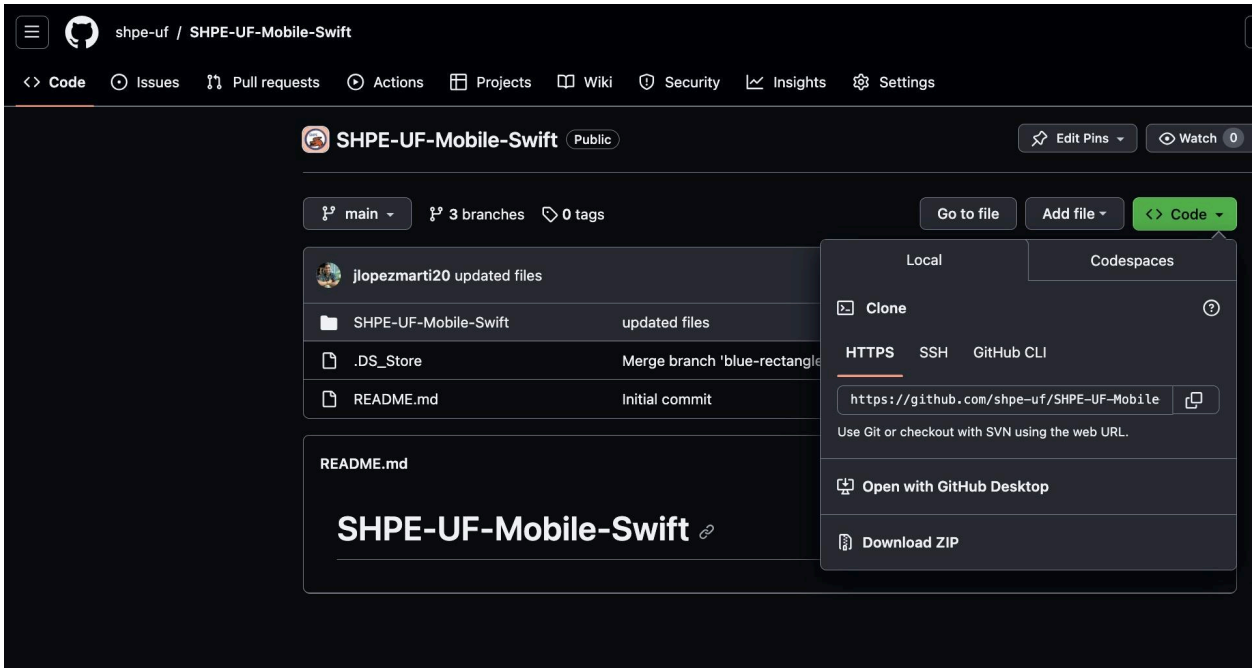
Running the app on your local device

For this project, since we'll be working with an existing remote Git repository, you should clone it to create a local copy and a connection for syncing updates. For Github repositories that need authentication, you'll need to set up your account information in Xcode.



To clone a repository, select Source Control > Clone.





Go to the repository. Under “Code” copy the URL for the repository and paste it into the repository URL field, and press Enter.

Xcode creates a copy of the project from the repository, saves it in the location you specify, checks out the branch you select, and opens the project.

How it works

1. Main App (SHPEUFAppView)
2. Pages (ui) & Authentication (auth)
3. Model Objects (model)
4. Realm Sync (MongoDB)
5. MongoDB (MongoDB)
6. Testing

Data Layer

The data layer is essentially the backbone of your application’s data management, responsible for retrieving, storing, and managing data from various sources. A single repository class should represent each different type of data handled in the app. For example, a **MoviesRepository** class for data related to movies, or a **PaymentsRepository** class for data related to payments.

Repository classes are mainly responsible for exposing data to the rest of the app, among other things. State Holder classes such as ViewModels (found in the UI Layer) should use repository classes as the only entry point to the data layer. Remember, your View should be as simple as possible.

UI Layer

The UI layer is responsible for reflecting and handling the UI State. UI state is the data that the user should see at a given moment. For example, if a user clicks a **Read Movie Description** button on a Movie App**,** the user should see the description for that movie. This is UI State.

The UI layer is broken down into UI elements and State holders. UI elements are activities that display the data (buttons, lists, cards, text boxes). State holders are responsible for the production of UI state and contain the necessary logic for that task.

To learn more about the State and ViewModel, Watch this video: (MVVM Architecture)

Here is a quick walkthrough of the main folders in our UI

- **pages:** contains all the screens in the app as well as their respective ViewModels (SignIn, Register, Home, Points)

Testing

Testing files will be the last to folders at the bottom:

- **SHPE-UF-Mobile-SwiftTest:** Unit testing (Business Logic)
- **SHPE-UF-Mobile-SwiftUITest:** UI testing (UI element)

Making changes to the frontend

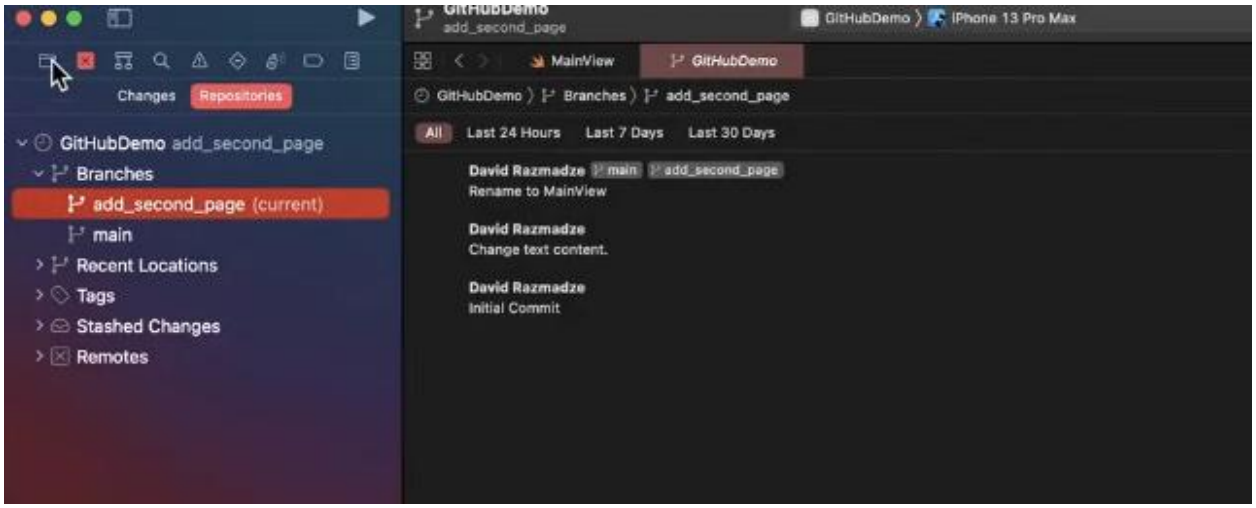
If you add any new files, make sure everything stays organized and follows the proper file structure. Try to create **reusable** components and only create new components when needed.

Managing Code with Source control & Github

Git excels in its ability to manage multiple features and timelines simultaneously through the use of branches, which serve as separate lines of development. This allows for efficient management of new features and bug fixes, ensuring that changes are isolated in branches until they are ready to be merged into the main codebase.

Creating and Managing Branches

When you are working on adding a new feature or fixing a bug in your source code, group all related changes in a separate branch. This way, you can manage everything related to that particular task in one place. For example, if you decide not to add a new feature after building it without using source control, it can be a significant effort to undo the work. If you organize your work in a branch, you can decide not to merge the branch and avoid having to undo work.

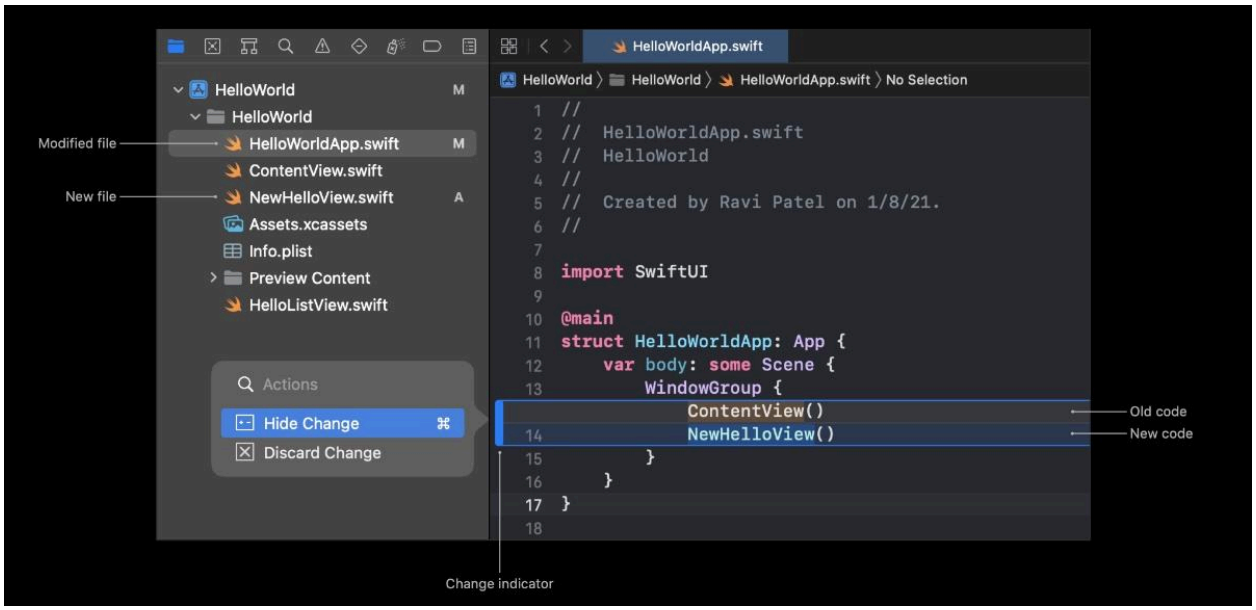


To see all your current local branches, you can go to the Branches folder under > Git > Repositories shown in the image above. For every project you start in Xcode that includes a Git repository, Xcode automatically creates a branch named '*main'*. If you want to start working on something new or make changes, you can control-click on the 'main' branch (or whichever branch you want to start from), select "Branch from [*branch name]"*, give your new branch for you and switch to it, so that any changes you make from this point on are part of that new branch.

Note: you can't switch to another branch if you have uncommitted changes, so commit your changes before switching to another branch.

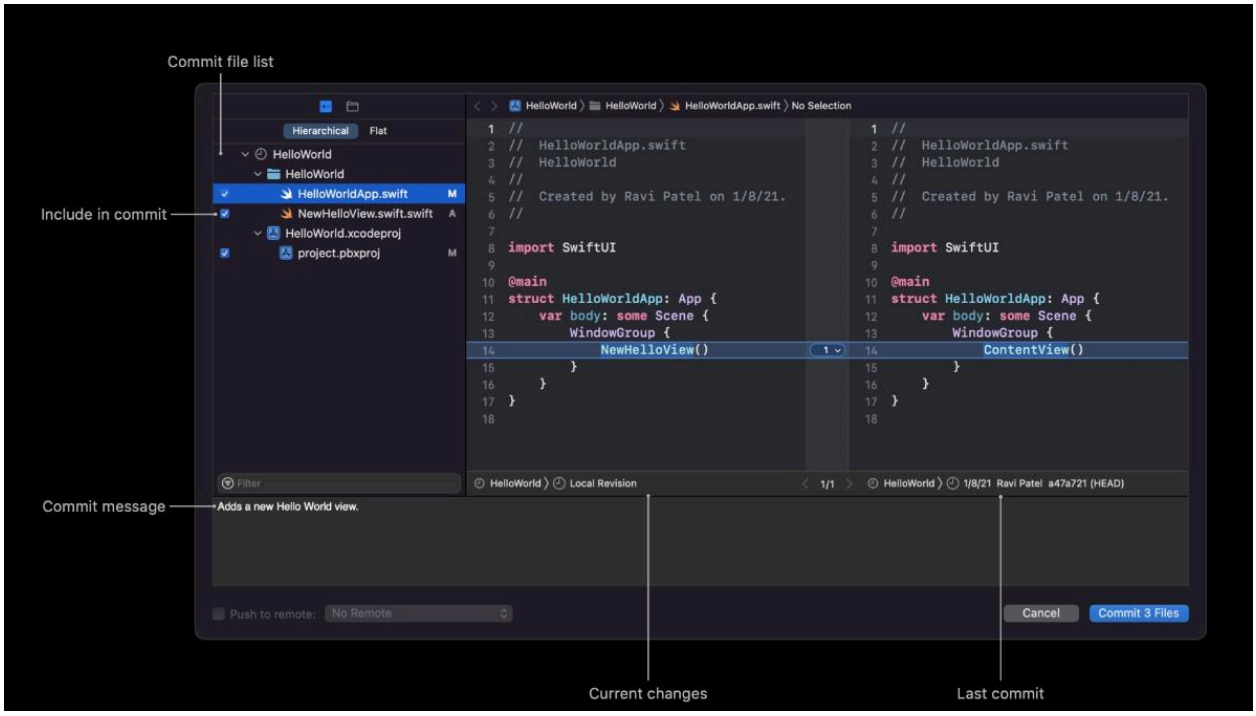
Making Commits

Commiting is like saving a snapshot of your project's current state, allowing you to document changes, save a version of the code, and revert back to it when needed.
As you make changes to your source code in a source control repository, Xcode tracks those changes and highlights them in the Project navigator and in the source editor. While you're working on a change, look at the Project navigator to see your changes in the current branch.



The Project navigator annotates files with changes since the last commit, using an A for new files or an M for modified files.

When you're done making changes, commit them to save them permanently in your source control repository. Choose Source Control > Commit, and review your changes.

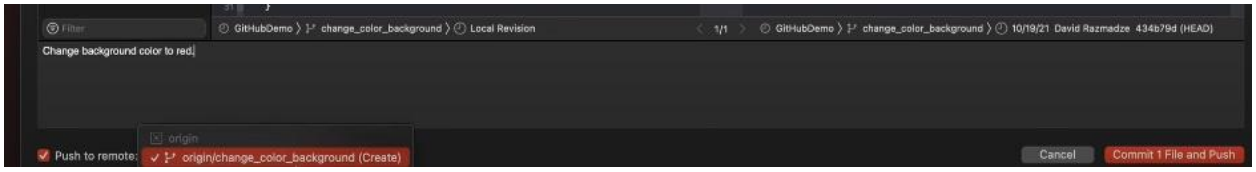


Document your changes with a commit message that describes what your commit accomplishes. Click the "Commit [number] files" button to commit your changes, or click

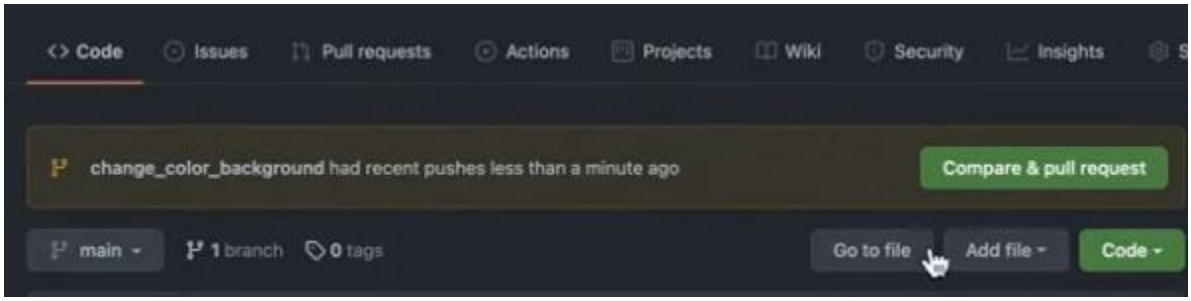
Cancel to continue working without committing to the repository. Xcode shows the commit message you provide when you look at the source control history.

Managing Pull Requests and Merging

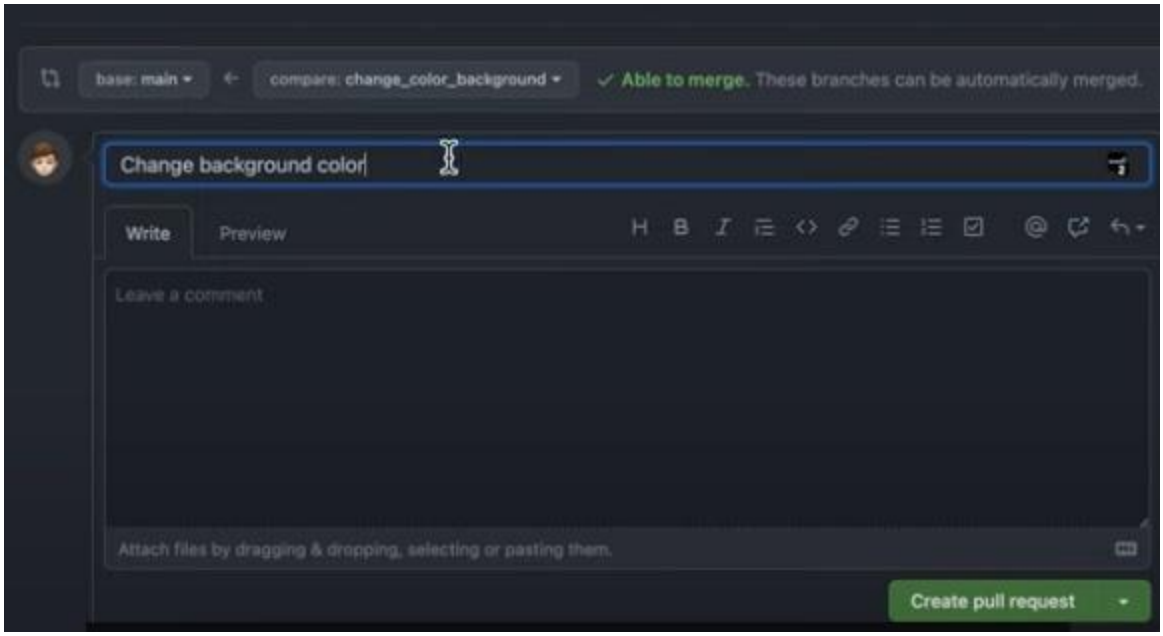
If you do not select the box, "*Push to remote",* you will only commit to your local. If you want to push your code to send it to your Github repository, you must select the box.



After committing and pushing your changes, go to Github to open a pull request, it will be labeled as "Compare & pull request".

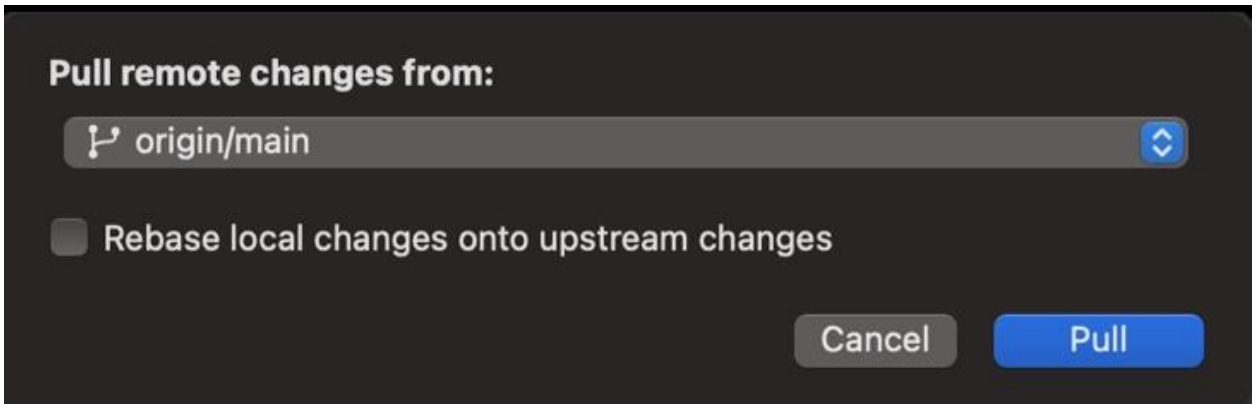


Add a brief description and create a pull request.



Once you review and are satisfied with the new feature, you can merge it back into the main branch. Click on *Merge pull request* > *confirm* and then you can delete that branch. If you go back to Xcode and go to the main branch, select Source control > Fetch changes.

Next to the “main” you can see two things that can be pulled in. To get changes from the remote repository you can click source Control > Pull. In the dialog that appears, select the branch with the changes you want to apply to your local repository, and click Pull.



That will update your project with the latest changes that another collaborator has pushed to the remote repository.

For more using Git/Github, watch this videos:

- Collaboration with Git
- Git and Github Beginners Guide

What's Next?

You are finally ready to contribute to the SHPE UF Android App. Lastly, make sure you are in the SHPE UF Tech Cabinet Discord as well as the SHPE-UF Mobile (Kotlin) Asana project.

If you haven't, go over the Development Process doc to learn how to assign and complete tasks using Asana, and what to expect from meetings.