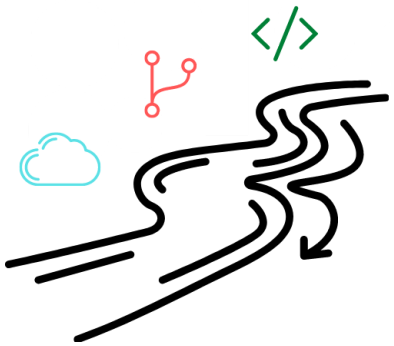
The Developer Workflow

When working on a project, once you have your developer environment set up and have a solid understanding of the SDLC, it's time to write code! As mentioned in the Developer Environment Prereq, it's most common to use an Integrated Developer Environment, or an IDE, when developing a project from start to finish. Here at the MoraL Code, we lean towards the IDEs made by a company called JetBrains, who are known for their Java IDE IntelliJ. However, feel free to use whatever IDE or developer environment you prefer!



Version Control

Version Control is one of the most important aspects of being a Software Engineer. It's a system that records changes to a file or set of files over time so that you can recall specific versions later. When working individually, version control provides you with the ability to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, and more! When working as part of a team, it makes collaborating and contributing to projects much easier. In addition to the individual abilities, team members can work on different aspects of a certain project without getting in each other's way.

Git and GitHub

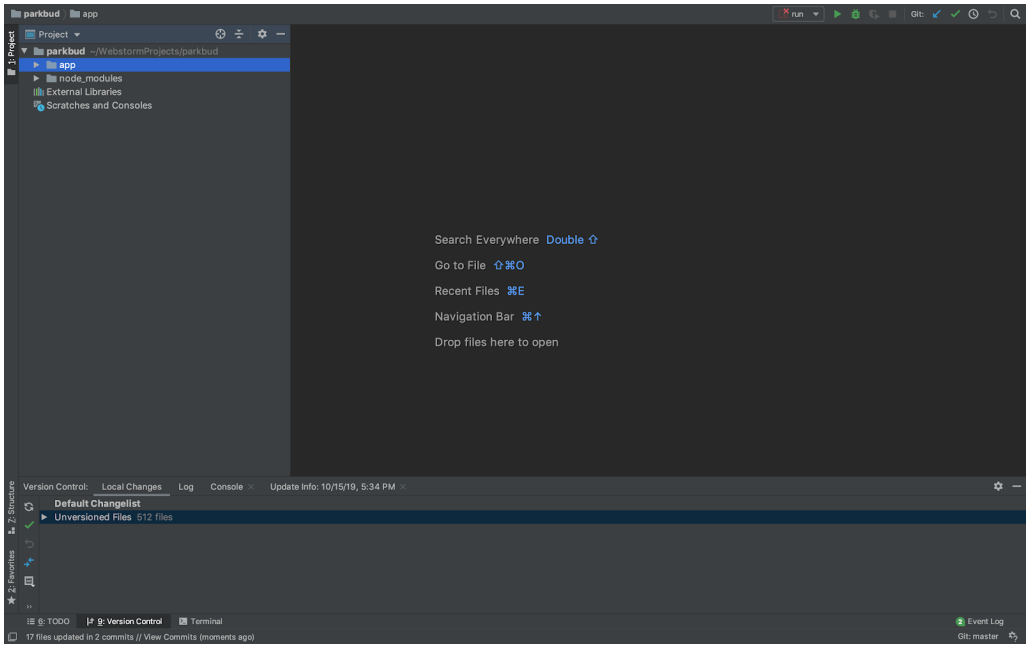
While there are tons of version control systems out there, the industry standard is Git. Git is a version control system used on the terminal. GitHub allows an easier, web-based method of using Git without having to learn all of the terminal commands. Just like with any version control system, you can revert to previous versions of your files, collaborate with others, and view/edit different versions of the same file. GitHub is also a great place to work on open source projects, create a portfolio, or gain some extra practice.

Open Source	**Portfolio**	**Extra Practice**
Contribute to the codebase of companies, freelancers, and hobbyists! (*PS: It doesn't matter if your code isn't actually accepted into the codebase, as long as you show that you're interested in/attempted contributing to open source!*)	Show off your projects, whether they're from class, a hackathon, or part of a side project. Some interviewers like to look at your GitHub account to see examples of your work.	If a specific topic catches your eye or if you'd like to expand on something from class, GitHub is the perfect place to explore existing code and practice. Just use GitHub's search bar to explore different topics and keywords!

The Developer Workflow

A major part of software engineering is establishing a developer "flow". This is a series of steps you'll take whenever working on code for a specific project or feature. Here's a comprehensive look at The MoraL Code developer flow (all of the following content is specific to JetBrains IDEs)!

We'll start off with the view you'll see once you've cloned a repository:



Creating a Feature Branch

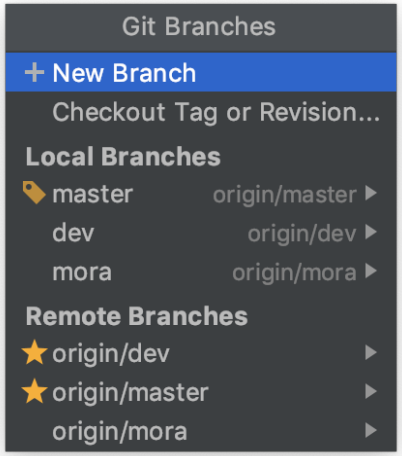
We'll be creating a feature branch for each issue we work on throughout development.

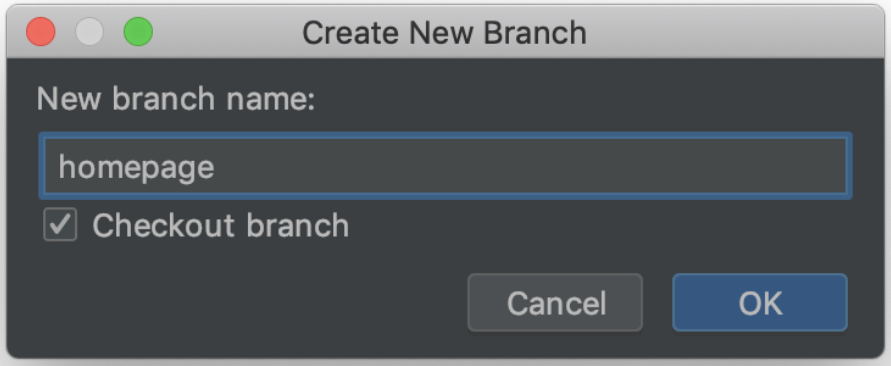
> "The basic idea of a feature branch is that when you start work on a feature (or UserStory if you prefer that term) you take a branch of the repository to work on that feature."
>
> - Martin Fowler

How do you create a feature branch with IntelliJ? The key is the bottom-right corner of the screen. You should see "Git: main". This means that we're connected to Git and we're in the main branch.

The main branch is where all of the finished code goes. When working on a feature, we don't want our code to be public and available to users until we're done writing and testing it. That's where the feature branch comes in! Before you start writing your code, create a feature branch for it. You can name it after the feature/issue you're working on (ex: homescreen) or you can name it after yourself (ex: mora).

To create your branch, click on the “Git: main” button and select “ + New Branch”



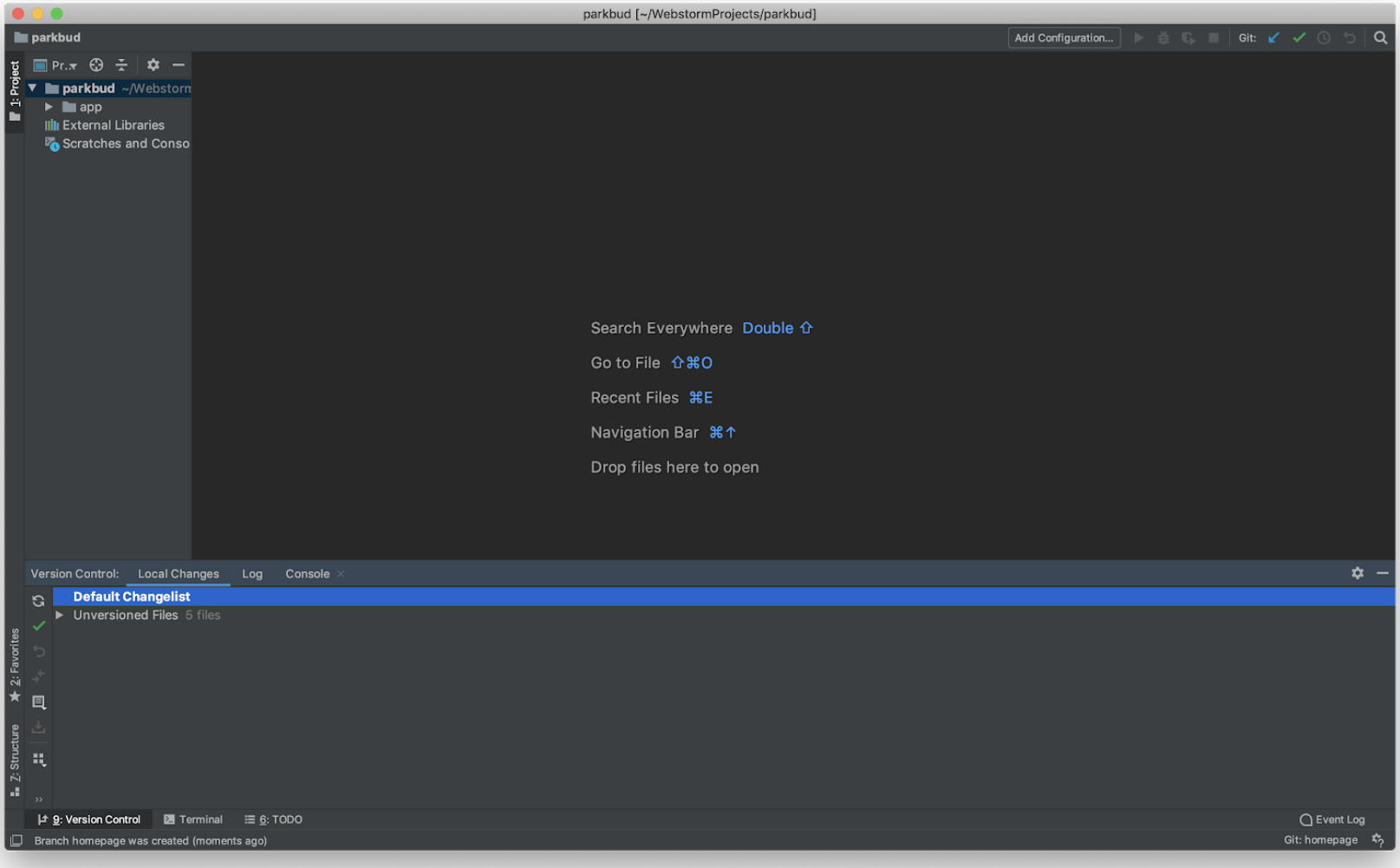


This will create a new local branch. Make sure “Checkout branch” is selected! This way IntelliJ automatically switches you to the new feature branch you just created.
**Remote branches are the ones that are saved in Git--they can be viewed by the team on GitHub, whereas local branches are the versions of remote branches you have saved on your laptop. Local branches “track” (aka are based on/are associated with) remote branches, but the changes you make to your code will only appear in the local branch. We’ll learn how to upload our changes to the remote branch a little later.*

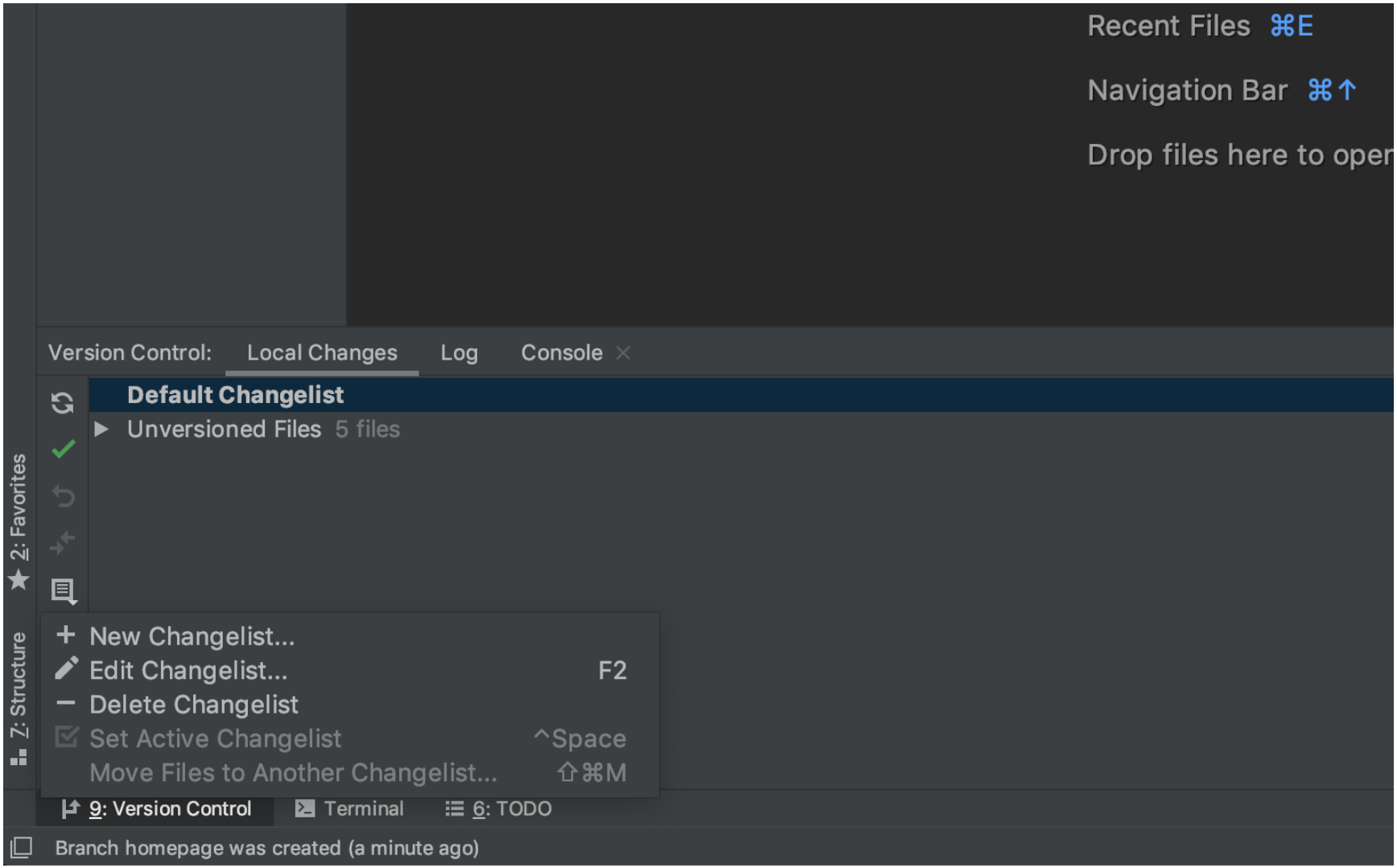
Coding With Changelists

Once you’re in your feature branch, you’re ready to start working on the code for your feature!
**It’s always important to make sure you’re in the correct branch. We’ll never directly modify the code in the main branch!*

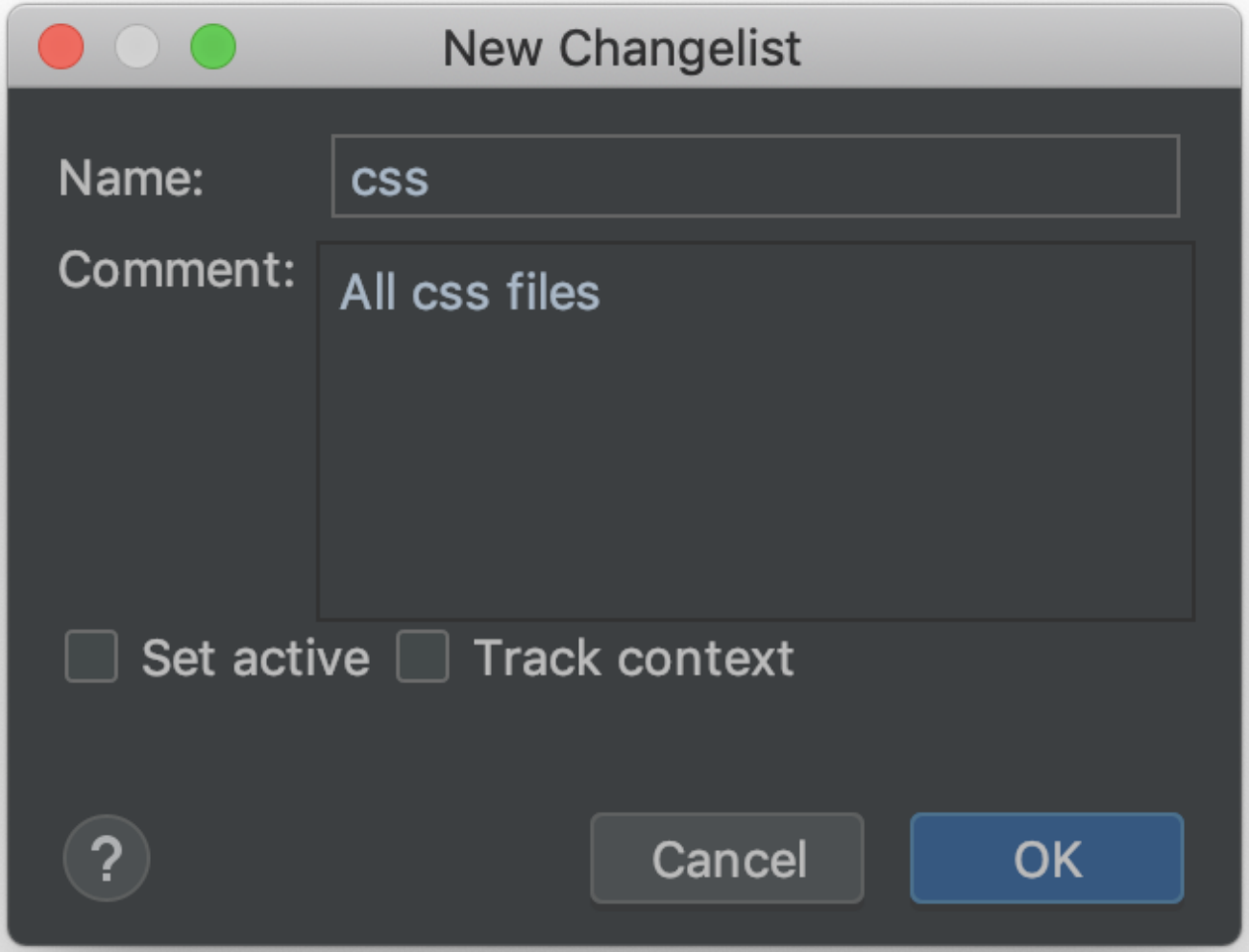
When working on a feature, all of the changes you make are kept in the “Default Changelist”.

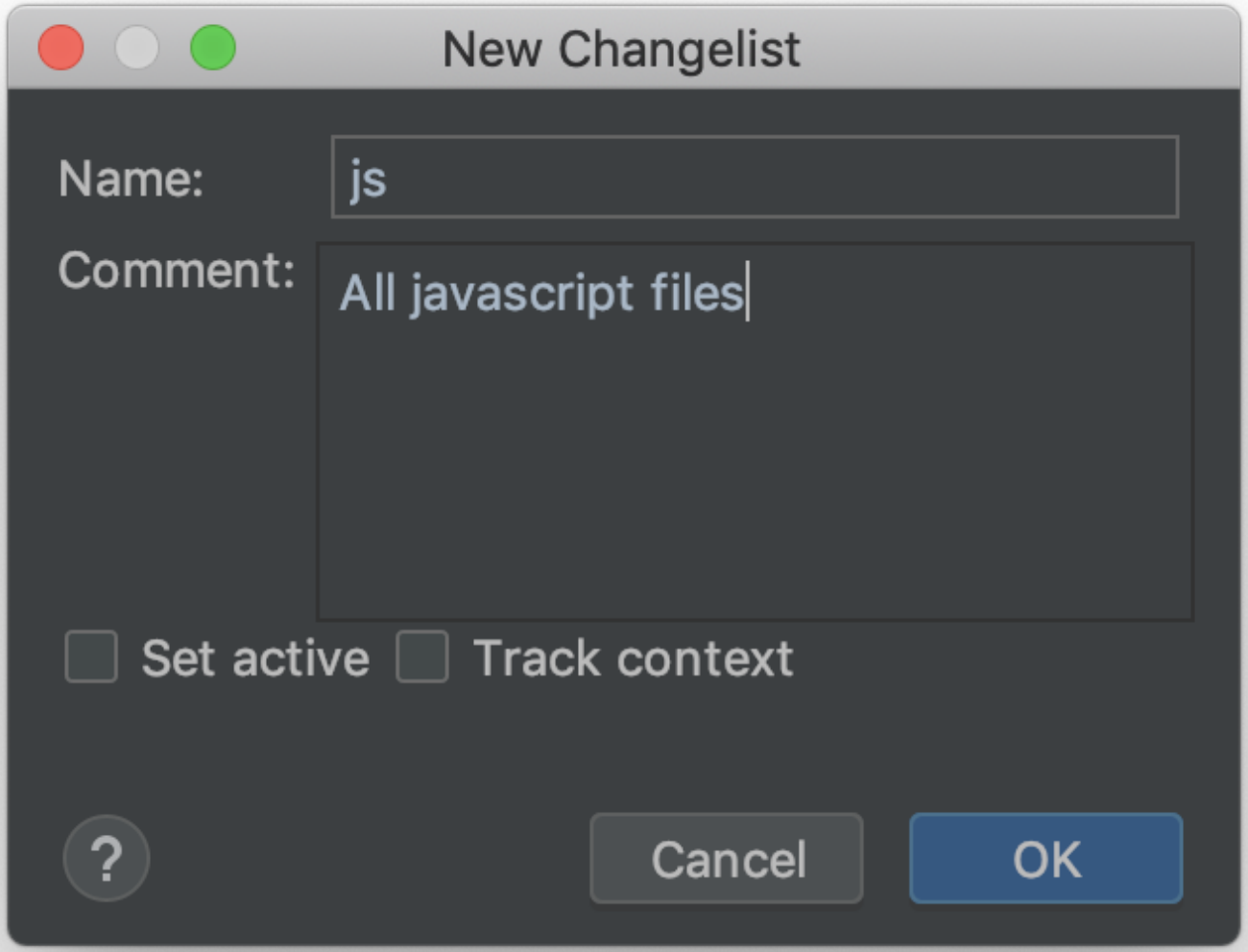


If you want, you can expand on this and use custom changelists to keep track of what you're doing. This is completely optional, but it's helpful to break up your code into smaller parts. Changelists allow you to split up your code into smaller categories based on the file. You can create custom changelists by clicking on the changelists icon and selecting "+ New Changelist…"

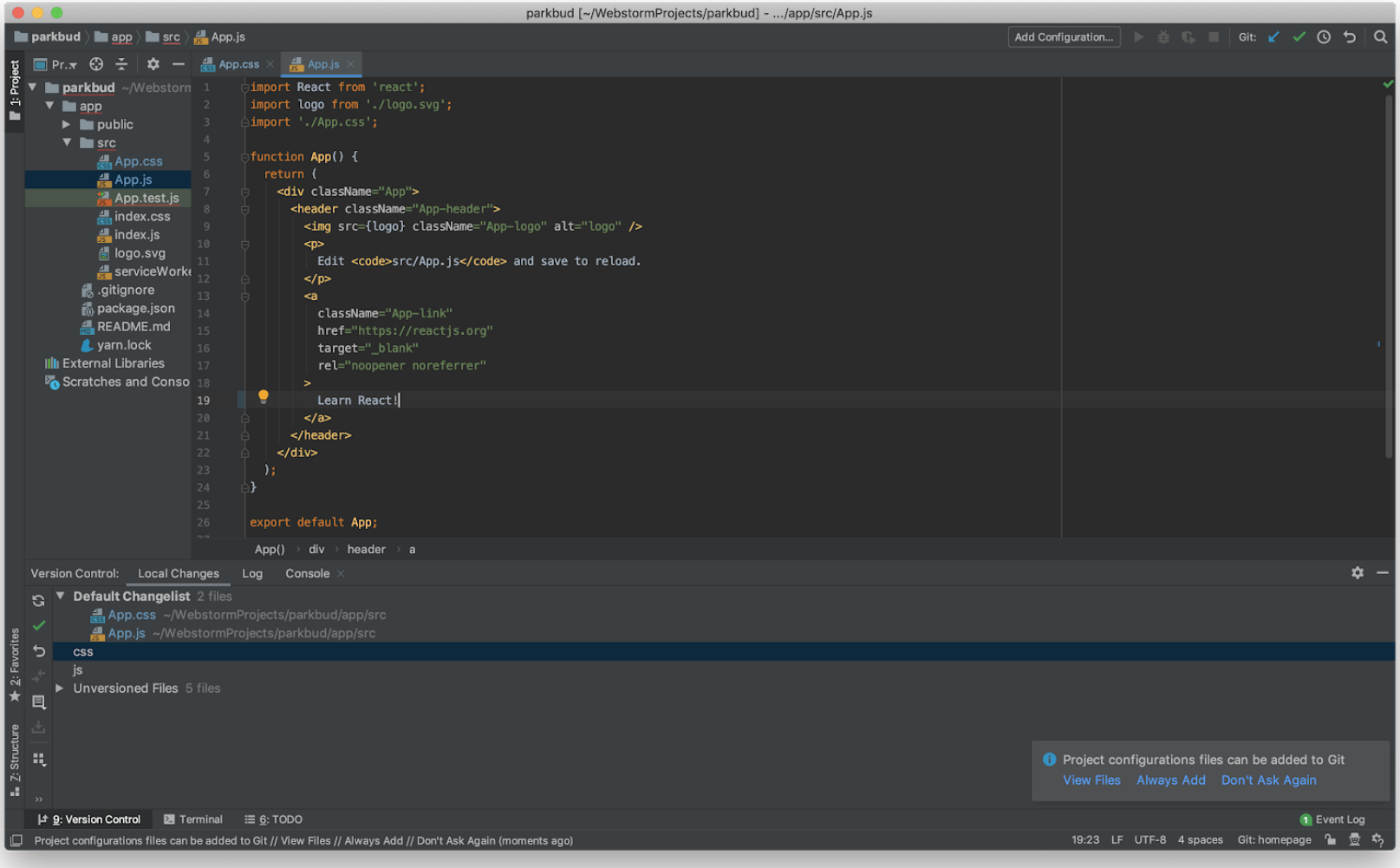


For example, here are two changelists that you might create when working in the homepage branch:

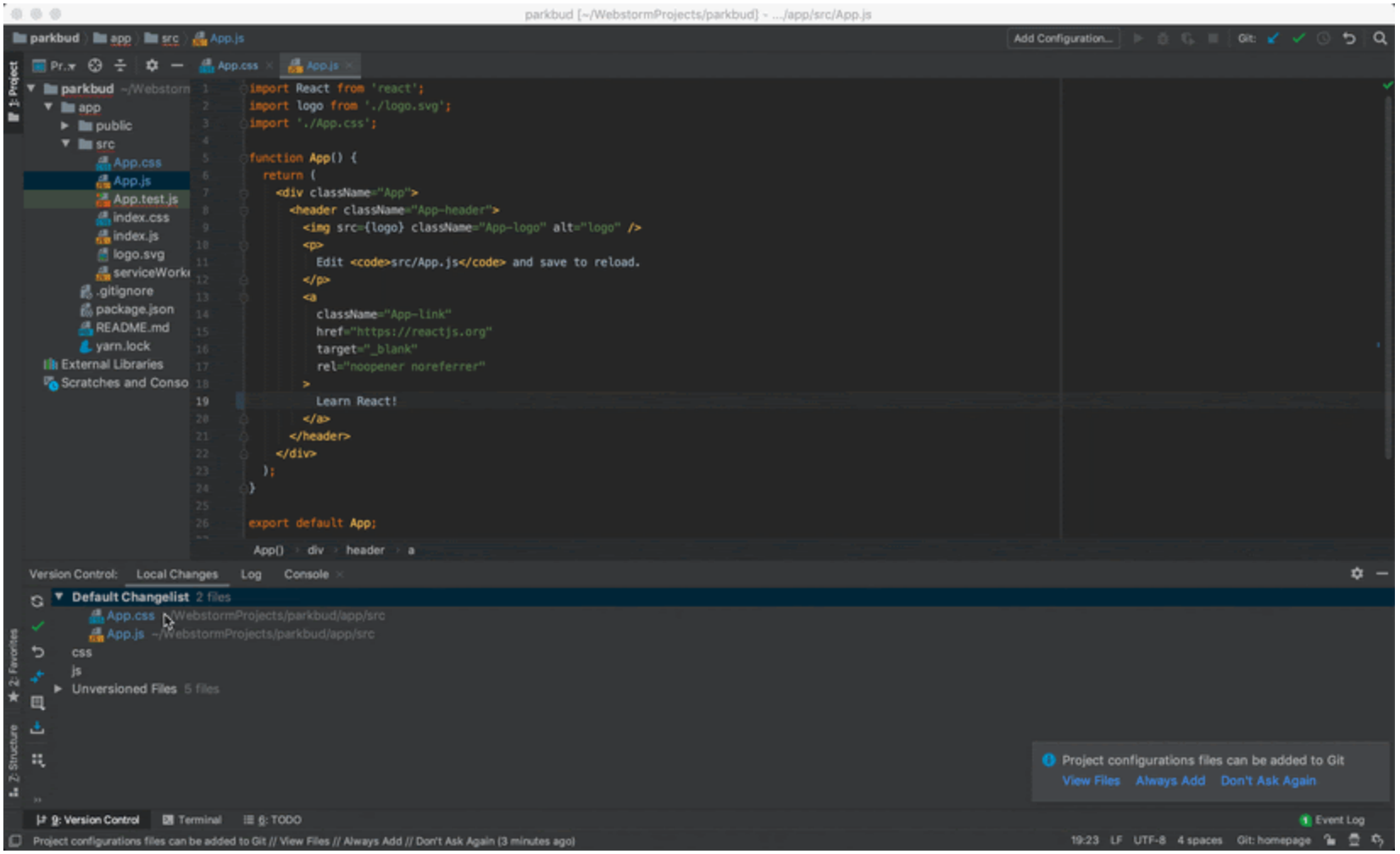

New Changelist
Name:
css
Comment:
All css files
Set active
Track context
?
Cancel
OK


New Changelist
Name:
js
Comment:
All javascript files
Set active
Track context
?
Cancel
OK

Now, when you make changes, they'll automatically go to the Default Changelist



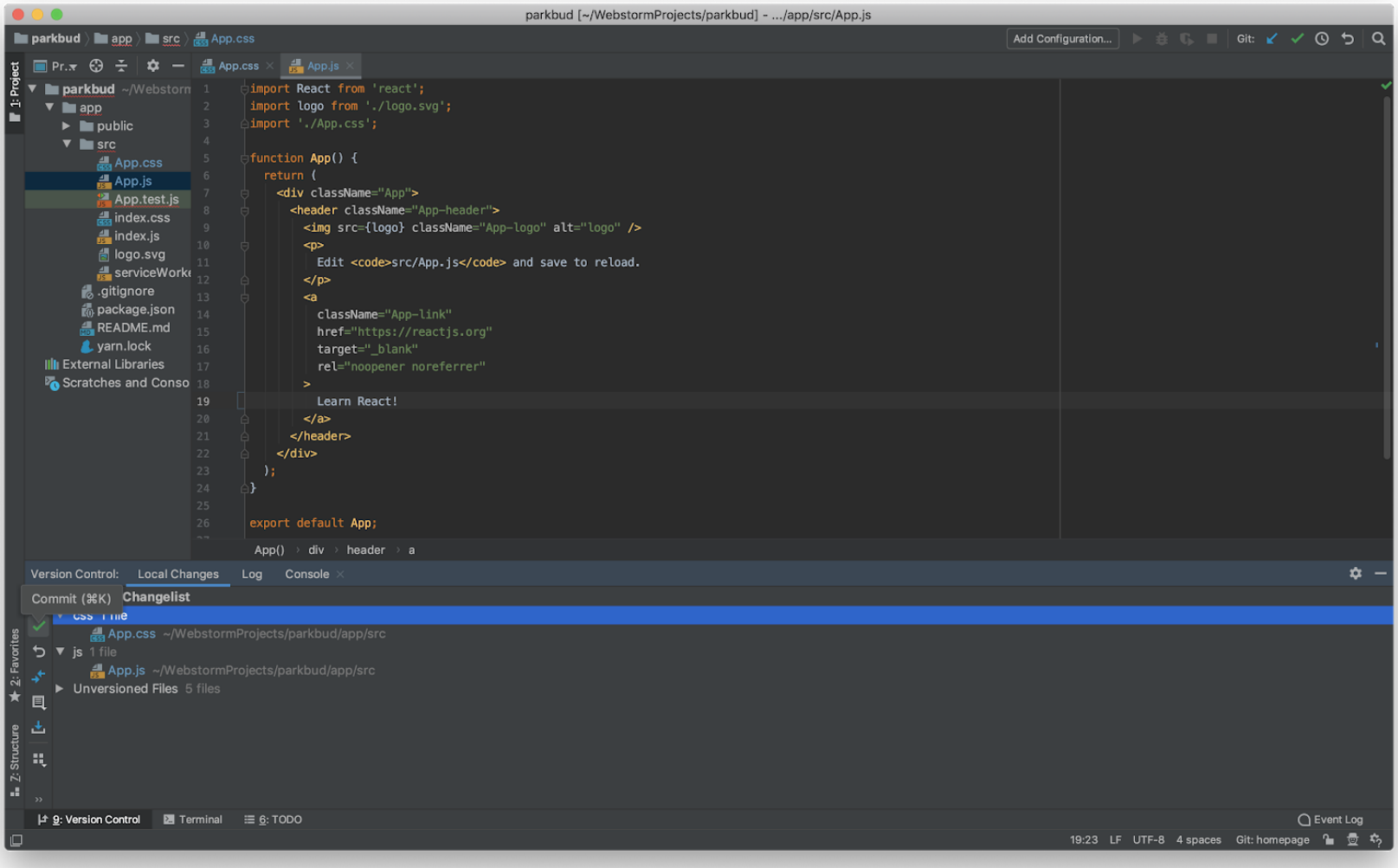
You can reorganize the files into specific changelists by dragging them or by right clicking them!



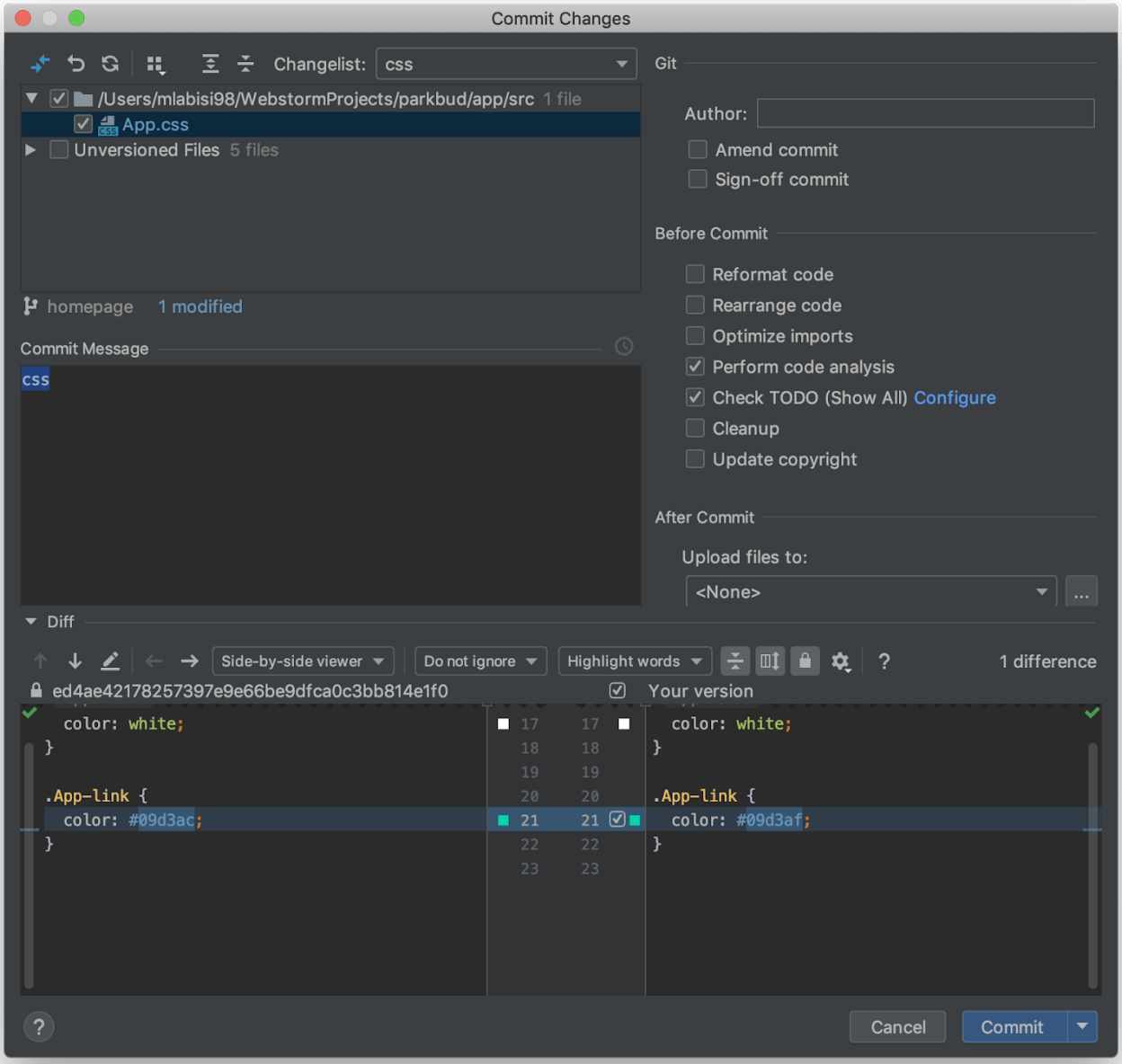
Committing Your Changes

Every once in a while, it's a good idea to commit your changes. Commiting a change is like hitting the save button on Microsoft Word while you're writing an essay (that was back in the day lol now everything autosaves).

So once you feel like you've made a good amount of progress, you can commit your changelist by selecting it (you can select multiple changelists by Ctrl-Clicking them) and then clicking the green commit button:



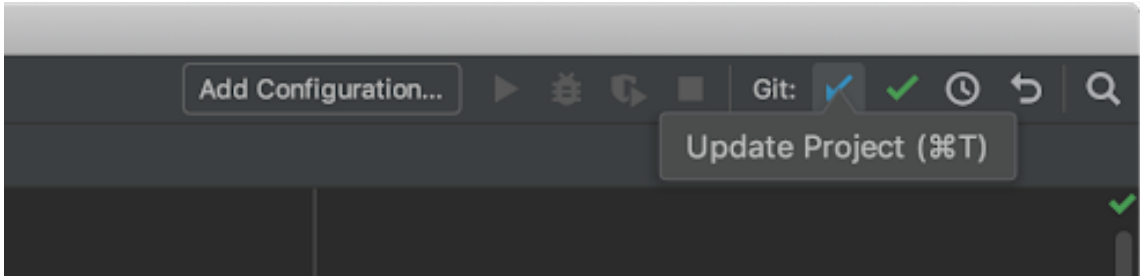
The “Commit Changes” dialog will come up. You should go ahead and add a quick description of the changes you made in this changelist. Then click “Commit”. If some dialog pops up asking you to check the warnings, just ignore it.


Commit Changes
Changelist: css
/Users/mlabisi98/WebstormProjects/parkbud/app/src 1 file
App.css
Unversioned Files 5 files
homepage 1 modified
Commit Message
css
Git
Author:
Amend commit
Sign-off commit
Before Commit
Reformat code
Rearrange code
Optimize imports
Perform code analysis
Check TODO (Show All) Configure
Cleanup
Update copyright
After Commit
Upload files to:
<None>
Diff
Side-by-side viewer
Do not ignore
Highlight words
1 difference
ed4ae42178257397e9e66be9dfca0c3bb814e1f0
Your version
color: white;
}
.App-link {
color: #09d3ac;
}
color: white;
}
.App-link {
color: #09d3af;
}
Cancel
Commit

Once you commit your change, it'll disappear from your changelists. Don't worry, your change has been registered! It's just waiting to be pushed to the remote branch! Remember, the changes you make in the local branch won't get shared on GitHub until you push them! We'll learn how to do that right now!

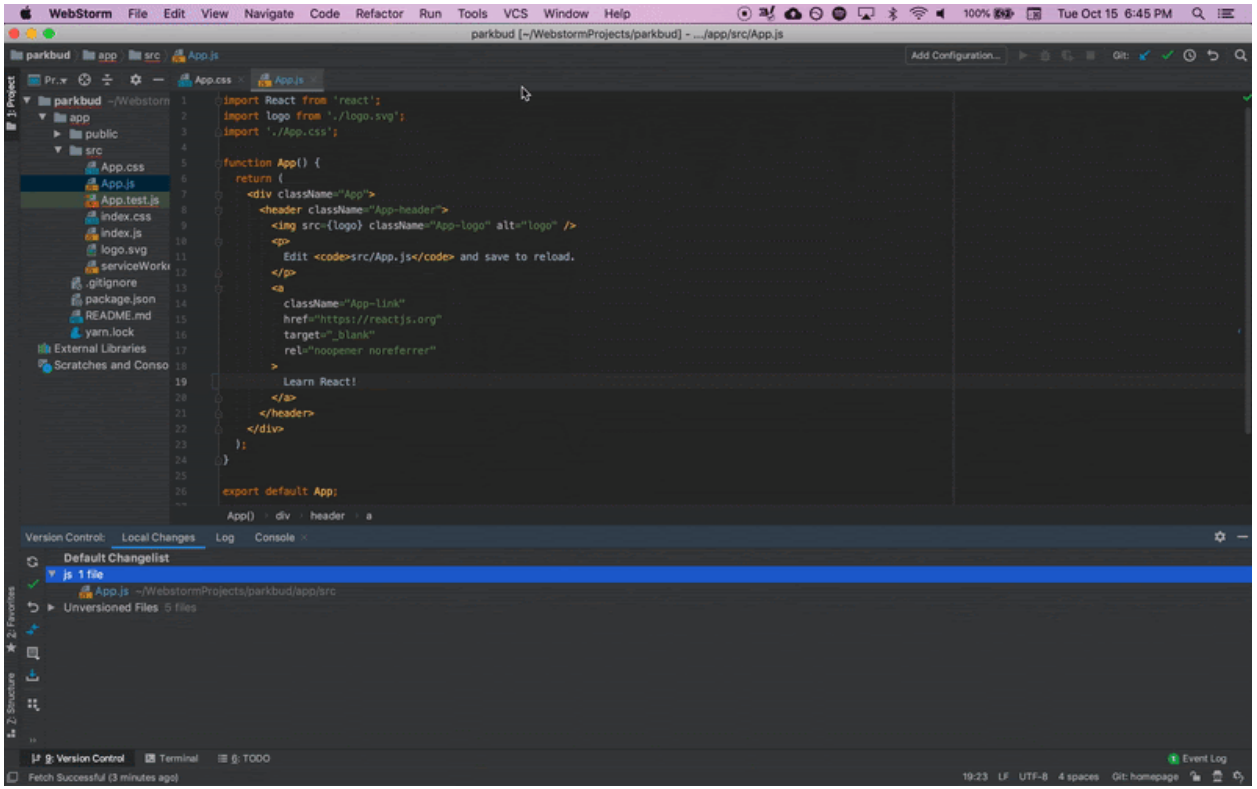
Pushing Your Changes

Before pushing your changes, sync with the remote version of your local branch in order to make sure your branch up-to-date. Click the blue arrow in the top-right corner.

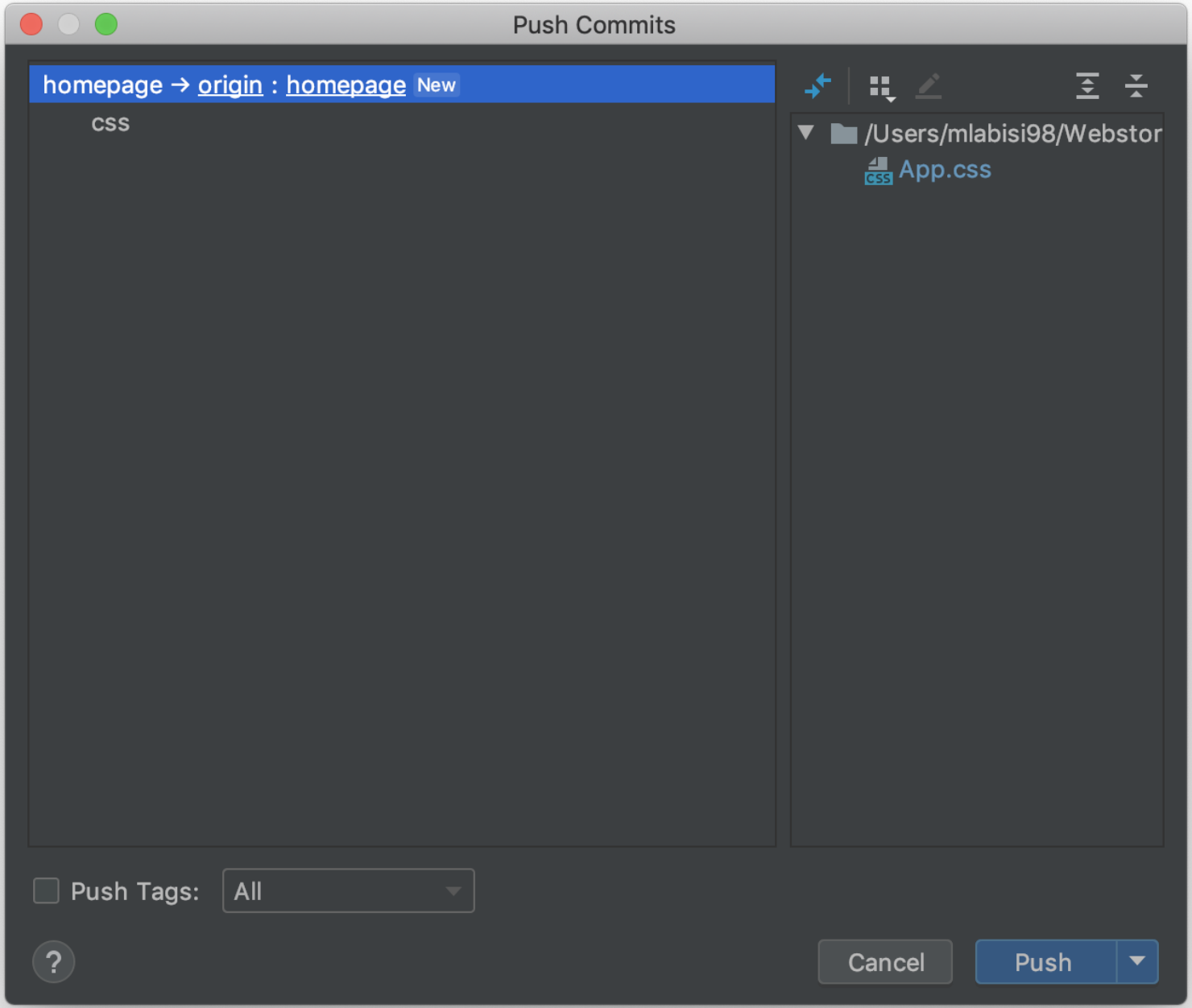


If you encounter any errors/conflicts, this means that two people are working on the same line of code. If this happens, just ask for help in the group chat!

After your local branch has been synced with the remote, you're ready to push your committed changes. Here's how you do it:

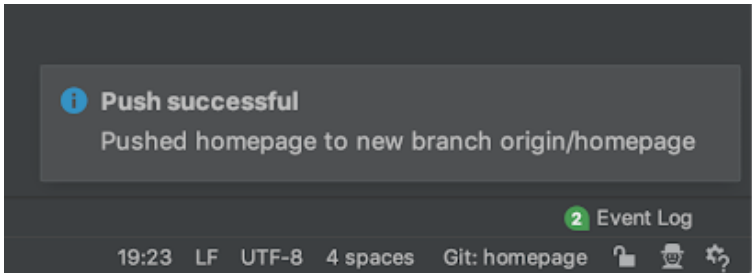


Take a look at the “Push Commits” dialog.


Push Commits
homepage → origin : homepage New
css
/Users/mlabisi98/Webstor
App.css
Push Tags:
All
Cancel
Push

The highlighted text here (`homepage → origin : homepage New`) means that you're pushing the local branch homepage to the origin (aka remote) branch homepage. The New tag means that this is a new branch that doesn't exist on GitHub yet. In the future when you push updates, this New tag will go away. If the remote branch has the wrong name for some reason, then you can click on the name and fix it. Then, click "Push" and you're good to go! Your code is now live on GitHub!

**Once again, make sure that you aren't pushing to the main branch! We won't be directly modifying the code in the main branch.*



Publishing Your Code via Pull Request

Although your code is all uploaded to your feature branch on GitHub, your code is not deployed (aka available to external users). In order to start the deployment process, you'll need to create a pull request for your branch. A pull request is another way of saying "I would like to merge (aka upload) the code from my feature branch to another branch."

The Deployment Pipeline

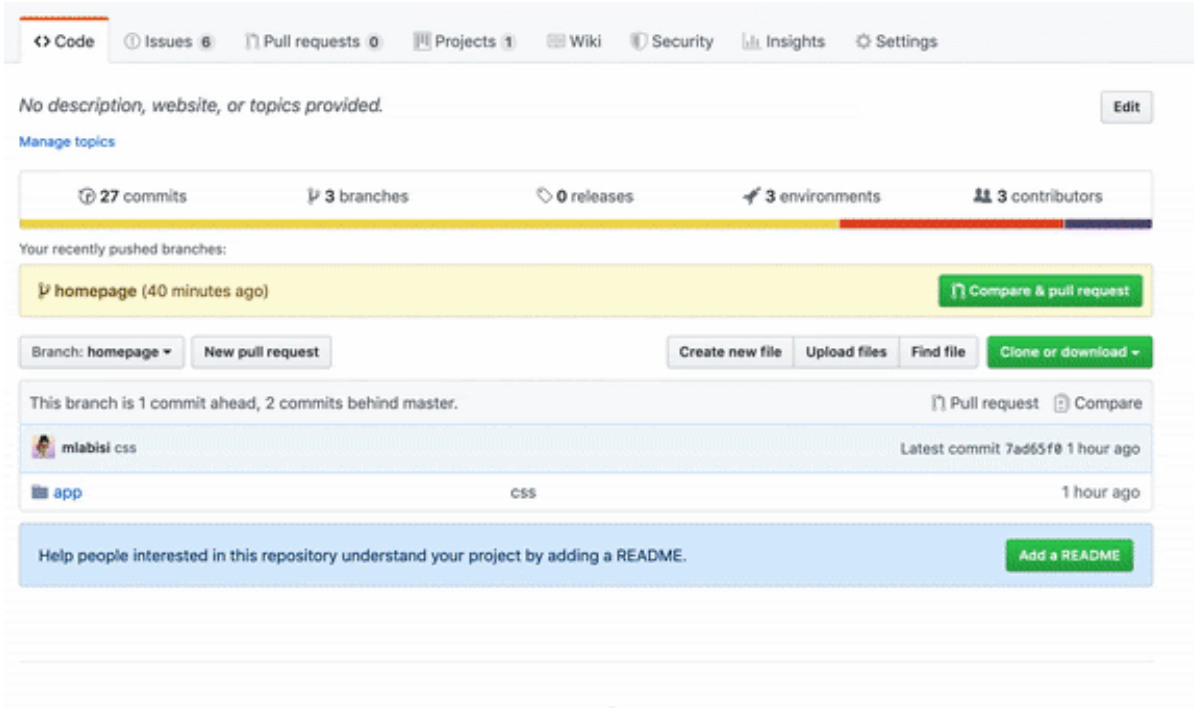
All code has to go through the following three-step process:

1. Create a pull request from your feature branch to the dev branch.
2. Once the code in the pull request has been validated and everyone has approved it, accept the pull request and merge the code.
3. Create a pull request from the dev branch to the main branch. The code has now been deployed to production!

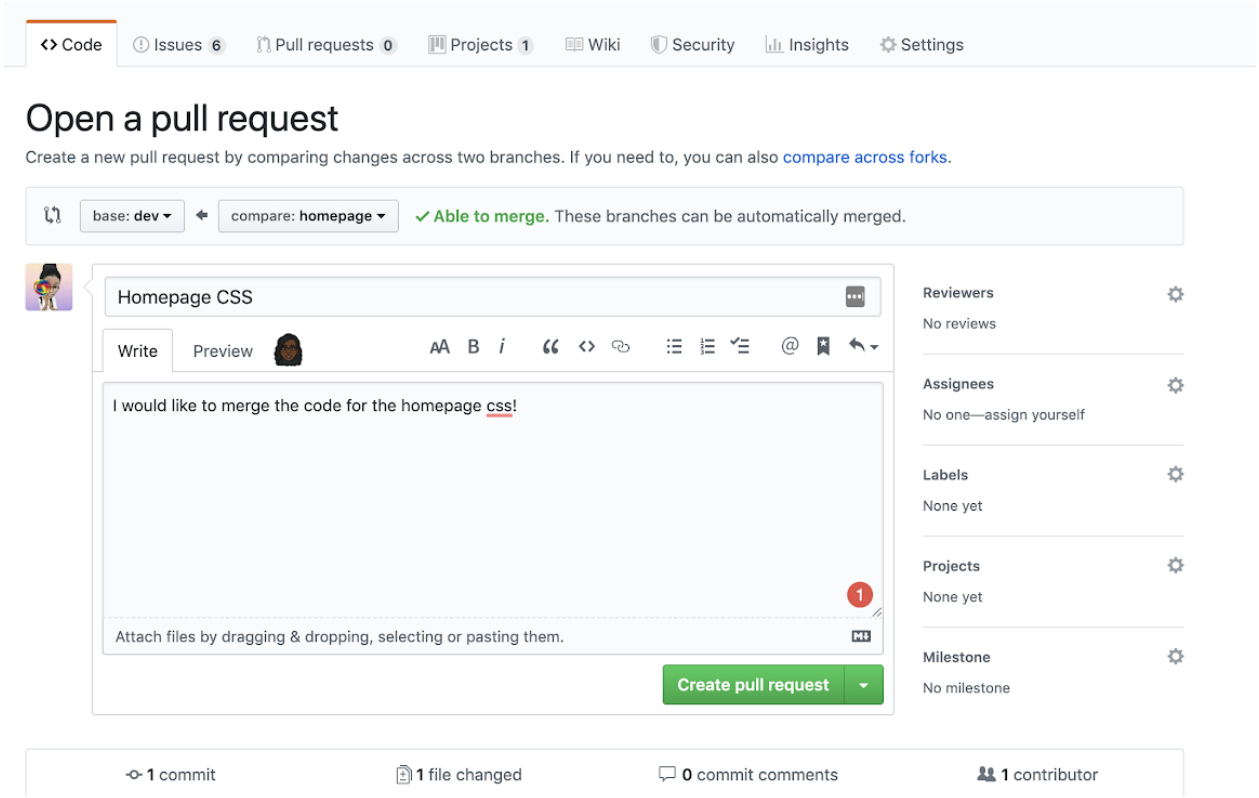
Another word for this three-step process is the pipeline. Pipelines are a key part of a software engineering methodology called "Continuous Integration", which is discussed in the Software Development Life Cycle Prereq.

Creating a Pull Request

Here's how to create a pull request. Make sure to choose the correct **base** branch. This is the branch you want to merge your code to.



When you open your pull request, make sure to write a descriptive title and description. Also, make sure to add everyone as a reviewer. Then click “Create pull request”.


Code
Issues 6
Pull requests 0
Projects 1
Wiki
Security
Insights
Settings
Open a pull request
Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.
base: dev
compare: homepage
Able to merge. These branches can be automatically merged.
Homepage CSS
Write
Preview
I would like to merge the code for the homepage css!
Attach files by dragging & dropping, selecting or pasting them.
Create pull request
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
1 commit
1 file changed
0 commit comments
1 contributor