# Starting out with Git (Github flavored)

## Create a Github account

<https://github.com>

## Get the Github desktop application

<https://desktop.github.com>

## Send Dave Cavins your GH username

## Open your favorite code editor

If you don’t have a favorite, here are some good options:

VS Code: <https://code.visualstudio.com/>

Atom: <https://atom.io/>

Brackets: <https://brackets.io/>

## What is Git?

Git is software that tracks changes to a project that can be shared and synchronized among contributors. These changes might be code (tracked very well) or other files like images or Excel docs (not great, but at least they’re kept up to date). Git uses a local repository (like on your computer) where you work, and all of the local repositories are synced against the remote, which in our case is stored at Github.

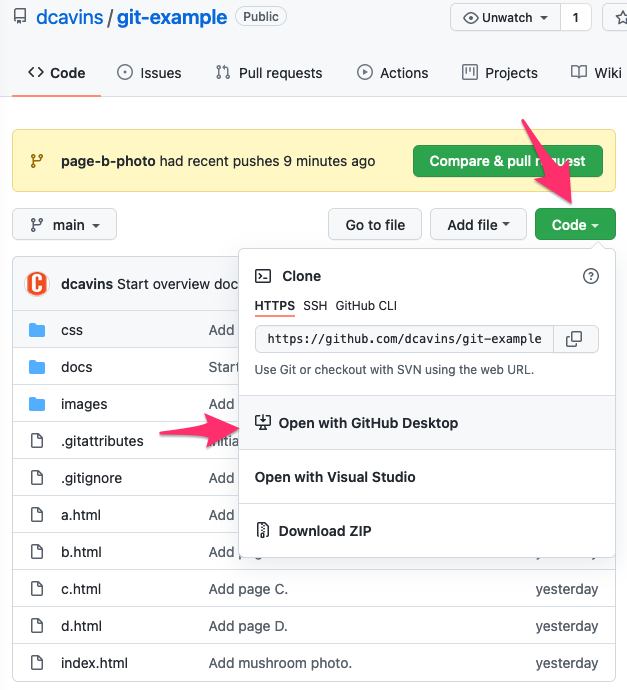
The Github desktop UI can handle most of the common git operations. Here’s what it looks like:

Website

Description automatically generated with medium confidence

## Working with an existing repository

First, you will need to **clone** the repository to a local, working copy. We’ll start from the Github repo page at <https://github.com/dcavins/git-example>:



## Commits

Commits are groups of changes that you bundle for saving and submission to the remote repository. They should be narrow in scope (when possible) because if something goes wrong, and you’re working your way back through git commit history, you’re going to have a lot easier time seeing what was changed in the step that broke your repo. Try to make commits on a topic or task rather than a time span.

## Branches

Git branches are probably the second most-important feature of git (after commits). Branches allow you to make an alternate universe of your code, which is great when making big changes, or changes that will take time. (Or you’re just slow.) You can create a new branch to work in, and at any time, switch back to the main branch to get the current state of the project from the remote repository. When you change branches, the files in the directory that is git-tracked will actually be changed by the procedure.

## Putting Branches Back Together

There are two main strategies for comparing and updating one branch from another, **Merging** and **Rebasing**. The main difference is order in which the “new” commits are applied.

When you merge branch B into branch A, the new commits from branch B are tacked onto the end of branch A (often as a large, merge commit). When you rebase branch B using branch A as the rebase source, the new commits in branch B are shifted to the end, or another way to think of it is that the new commits from branch A are shifted earlier in time, compared to the work in branch B.

Diagram, schematic

Description automatically generated

## Pushing to Github

My preferred procedure for making a push to the main branch is as follows:

* You’re working in a feature branch, right? Great, commit your changes.
* Switch back to the main branch and sync it with the remote repo.
* Switch to your feature branch and rebase it, using the newly updated main branch as the source.
* Deal with conflicts.
* Merge the conflict-free feature branch into the main branch.
* Push to Github!
* If you don’t need your feature branch anymore, you can delete it, since all of the nice, atomic commit history is now in the main branch.

## Pull Requests (PRs)

PRs a great option when you are not the super expert on a repo. You can create a feature branch and make all of the changes that you think need to be made, test your work locally, then publish that feature branch to Github and ask the super expert of the repo to review your changes and merge them to the main branch if approved. If you are not 100% sure of your changes, this is the approval routine for you!