Git is a tool for you to code with your teammates by hosting the ground truth code on a remote server (hosted by **GitHub**), and storing version history in the form of **commits**. Each codebase is called a **repository**, which, in this case, is your website.

Basic Commands

`git pull` delivers all the changes *from* the remote repository *to* your local computer. In other words, if your teammates made any changes to the ground truth code, and set their updated version of the code as the new ground truth code, you will receive all of their changes.

`git add <filepath>` adds all the modifications you made to the file at the filepath on your local computer to the **staging area**, which just stores all the files that will be included in the next commit. You need to add your files before you commit them.

`git add -A` adds all the modifications to *all* the files in the repository on your local computer to the staging area.

`git commit -m "your commit message"` creates a single **commit** that includes all the modifications currently in the staging area. You can think of a commit as a packaged set of changes you've made to the repository. Commits will be stored on your local computer until you update the ground truth code on the remote server (GitHub). Each commit has a unique hashed ID that looks like a random sequence of numbers/letters.

`git push` delivers all the changes *from* your local computer *to* the remote repository. This will update the ground truth code on the server.

Sometimes, `git push` will fail if there is a conflict between the code on your local computer and the ground truth code on the remote server. This is called a **merge conflict**. Sometimes, Git's algorithm can smartly merge code without any issues. Then all you have to do is run `git push`.

Other times, you'll need to manually handle the merge conflict. If you see a merge conflict error when trying to push, open Visual Studio Code and navigate to the files that have a red "!" mark in the file navigator. You will have three options:

1. Accept Current Change: Overwrites the code from the server with your own code

2. Accept Incoming Change: Overwrites your own code with the code from the server
3. Accept Both Changes: Combines your code and the server's code together

Once you fix all the conflicts, you'll have to `git add` the changes again, `git commit` again, and `git push` again.

Branches

Think of a **branch** as a separate version of the repository. If you have only one ground truth, everyone on the team will need to work with the same ground truth. What if one teammate wants to pursue a feature that will take a long time, but doesn't want to push broken/unfinished code to the server? That teammate wouldn't be able to store changes in the remote server. Uh oh, one day their computer drops into the Charles. All their changes are lost!

When you create a branch, you copy the current state of the remote repository, and create a new ground truth to work with. Teammates can freely push changes to this new branch without having to worry about modifying the main ground truth or other branches!

In fact, when you first create a repository, a branch called **main** is automatically created (used to be called "master", which is now depreciated). The main branch is where, namely, your "main" ground truth code is stored—your working product. Now, if you want to develop a feature that will take a long time and isn't really related to what your other teammates are doing, you can create a branch. For example, if you want to code up a minigame on your website, you can create a branch called "game".

`git branch <branch name>` will create a new branch (with the current state of the remote repository) with that branch name.

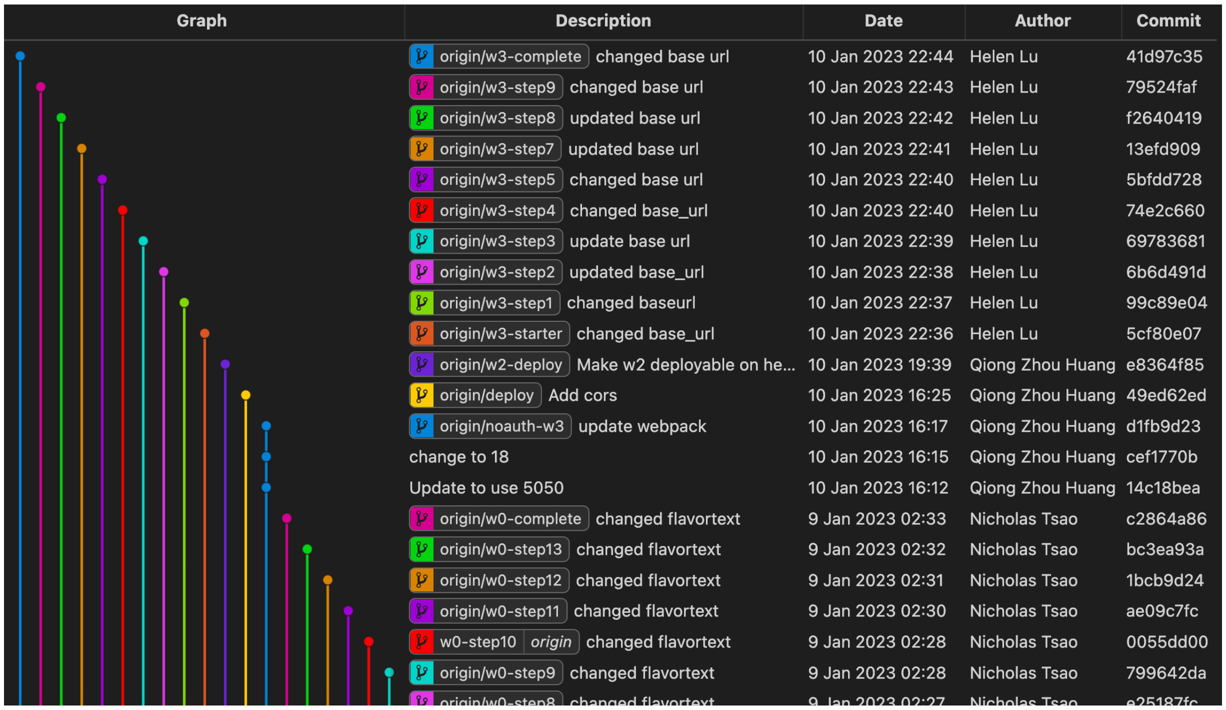
`git checkout <branch name>` will switch to the branch with that branch name. This is what you've been using in lectures to navigate between different steps and workshops!

`git push --set-upstream origin <branch name>` tells the remote server that you've created a new branch. After doing this, you'll be able to see the branch on GitHub, and other teammates can work on your branch.

To merge a branch into the main branch, you'll have to checkout main first, then run **`git`**

`merge <branch name>`. Then you can deal with merge conflicts the usual way, then push.

Creating branches on a small team isn't necessary, but if you want to work on extended features, or features you don't really know will work yet, they can be useful! They also make your version history look very cool :'), see the version history graph of catbook-react below:



Graph	Description	Date	Author	Commit
	origin/w3-complete changed base url	10 Jan 2023 22:44	Helen Lu	41d97c35
	origin/w3-step9 changed base url	10 Jan 2023 22:43	Helen Lu	79524faf
	origin/w3-step8 updated base url	10 Jan 2023 22:42	Helen Lu	f2640419
	origin/w3-step7 updated base url	10 Jan 2023 22:41	Helen Lu	13efd909
	origin/w3-step5 changed base url	10 Jan 2023 22:40	Helen Lu	5bfdd728
	origin/w3-step4 changed base_url	10 Jan 2023 22:40	Helen Lu	74e2c660
	origin/w3-step3 update base url	10 Jan 2023 22:39	Helen Lu	69783681
	origin/w3-step2 updated base_url	10 Jan 2023 22:38	Helen Lu	6b6d491d
	origin/w3-step1 changed baseurl	10 Jan 2023 22:37	Helen Lu	99c89e04
	origin/w3-starter changed base_url	10 Jan 2023 22:36	Helen Lu	5cf80e07
	origin/w2-deploy Make w2 deployable on he...	10 Jan 2023 19:39	Qiong Zhou Huang	e8364f85
	origin/deploy Add cors	10 Jan 2023 16:25	Qiong Zhou Huang	49ed62ed
	origin/noauth-w3 update webpack	10 Jan 2023 16:17	Qiong Zhou Huang	d1fb9d23
	change to 18	10 Jan 2023 16:15	Qiong Zhou Huang	cef1770b
	Update to use 5050	10 Jan 2023 16:12	Qiong Zhou Huang	14c18bea
	origin/w0-complete changed flavortext	9 Jan 2023 02:33	Nicholas Tsao	c2864a86
	origin/w0-step13 changed flavortext	9 Jan 2023 02:32	Nicholas Tsao	bc3ea93a
	origin/w0-step12 changed flavortext	9 Jan 2023 02:31	Nicholas Tsao	1bcb9d24
	origin/w0-step11 changed flavortext	9 Jan 2023 02:30	Nicholas Tsao	ae09c7fc
	w0-step10 *origin* changed flavortext	9 Jan 2023 02:28	Nicholas Tsao	0055dd00
	origin/w0-step9 changed flavortext	9 Jan 2023 02:28	Nicholas Tsao	799642da
	origin/w0-step8 changed flavortext	9 Jan 2023 02:27	Nicholas Tsao	e25187fc

Each of the lines is a separate branch! (This Visual Studio Code extension is called Git Graph. You can install it for free by searching "Git Graph" in the extensions tab on the left sidebar).

Honorable Mentions

`git fetch` syncs your local computer with the latest changes on the remote server.

`git reset --hard` resets *all* the modifications you made to your code since the last commit.

`git diff` shows all the modifications you made to your local code *before* adding them to the staging area. This is useful for seeing all the changes you've made.

`git log` lists all the commits (the version history), from most recent to oldest.

`git clone <HTTPS link>` copies the remote repository on GitHub onto your local computer, syncs your local computer with the remote repository. Access the HTTPS link by going to the repository on GitHub, clicking “Code”, and clicking the copy button.

`git revert <commit ID>` will revert your code back to the commit with that ID.

`HEAD` is synonymous with the commit that you’re currently on (aka what version of the code you’re viewing). NOTE: If you have a **detached HEAD** error, please post on Piazza and we will try to help! This can be dangerous because you can lose all your changes.

Phew, that was a lot! Obviously, you don't need to know all these commands. However, referencing this guide can be helpful as you work through your projects. You can also find the Git documentation here. Google is your friend for Git! As always, if you have questions, please post on Piazza or visit us at office hours! The most updated schedule can always be found at https://weblab.mit.edu/schedule/.