Contents

1/ What is Github?

Github is a hosting platform for git repositories.

You can put your own Git repos on Github (cloud) and access them from anywhere and share them with people around the world.

Beyond hosting repos, Github also provides additional collaboration features that are not native to Git (but are super useful).

Basically, Github helps people share and collaborate on repos.

Difference between Git and Github:

+ Git is the version control software that runs locally on your machine. You don't need to register for an account.

You don't need the internet to use it. You can use git without ever touching Github.

+ Github is a service that hosts Git repositories in the cloud and makes it easier to collabarte with other people.

You do need to sign up for an account to use Github. Ít's an online place to share work that is done using Git.

Github is not your only option... There are tons of competing tools that provide similar hosting and collaboration features,

including GitLab, BitBucket, and Gerrit.

If you ever plan on working on a project with at least one other person, Github will make your life easier! Wheter you're building a hobby project with your friend or you're collaborating with the entire world, Github is essential.

2/ Cloning

So far we've created our own Git repositories from scratch, but often we want to get a local copy of an existing repository instead.

To do this, we can clone a remote repository hosted on Github or similar websites. All we need is a URL that we can tell Git to clone for use.

To clone a repo, simply run git clone <url>.

Git will retrieve all the files associated with the repository and will copy them to your local machine.

In addition, Git initializes a new repository on your machine, giving you access to the full Git history of the cloned project.

Command Syntax: git clone <url>

(Make sure you are not inside of a repo when you clone)

Note: git clone is a standard git command.

It is NOT tied specifically to Github. We can use it to clone repositories that are hosted anywhere! It just happens that most of the hosted repos are on Github these days.

3/ Permissions?

Anyone can clone a repository from Github, provided the repo is public. You do not need to be an owner or collaborator to clone the repo locally to your machine.

You just need the URL from Github.

Pushing up your own changes to the Github repo...that's another story entirely! You need permission to do that.

Git clone is a standard git command.

It is NOT tied specifically to Github. We can use it to clone repositories that are hosted anywhere!. It just happens that most of the hosted repos are on Github these days.

4/ Register for Github

Configuring SSH Keys

You need to be authenticated on Github to do certain operations, like pushing up code from your local machine.

Your terminal will prompt you every single time for your Github email and password, unless...

You generate and configure an SSH Key! Once configured, you can connect to Github without having to supply your username/password.

https://docs.github.com/en/authentication/connecting-to-github-with-ssh

5/ Creating our first Github repo

- Option 1: Existing Repo

If you already have an existing repo locally that you want to get on Github...

+ Create a new repo on Github

+ Connect your local repo (add a remote)

+ Push up your chages into Github

- Option 2: Start From Scratch:

If you haven't begun work on your local repo, you can...

+ Create a brand new repo on Github

+ Clone it down to your machine

+ Do some work locally

+ Push up your changes to Github

6/ A Crash course on git remote

- Remote

Before we can push anything up to Github, we need to tell Git about our remote repository on Github.

We need to setup a "destination" to push up to.

In Git, we refer to these "destinations" as remotes. Each remote is simply a URL where a hosted repository lives.

- Viewing remotes

To view any existing remotes for you repository, we can run git remote or git remote -v (verbose, for more info)

This just displays a list of remotes. If you haven't added any remotes yet, you won't see anything!

Command Syntax: git remote -v

- Adding a new remote

A remote is really two things: a URL and a label. To add a new remote, we need to provide both to Git

Command Syntax: git remote add <name> <url>

Okay Git, anytime i use the name 'origin', I'm referring to this particular Github repo URL.

Example: git remote add origin https://github.com/blah/repo.git

- Origin

Origin is a conventional Git remote name, but it is not at all special. It's just a name for a URL.

When we clone a Github repo, the default remote name setup for us is called origin. You can change it. Most people leave it.

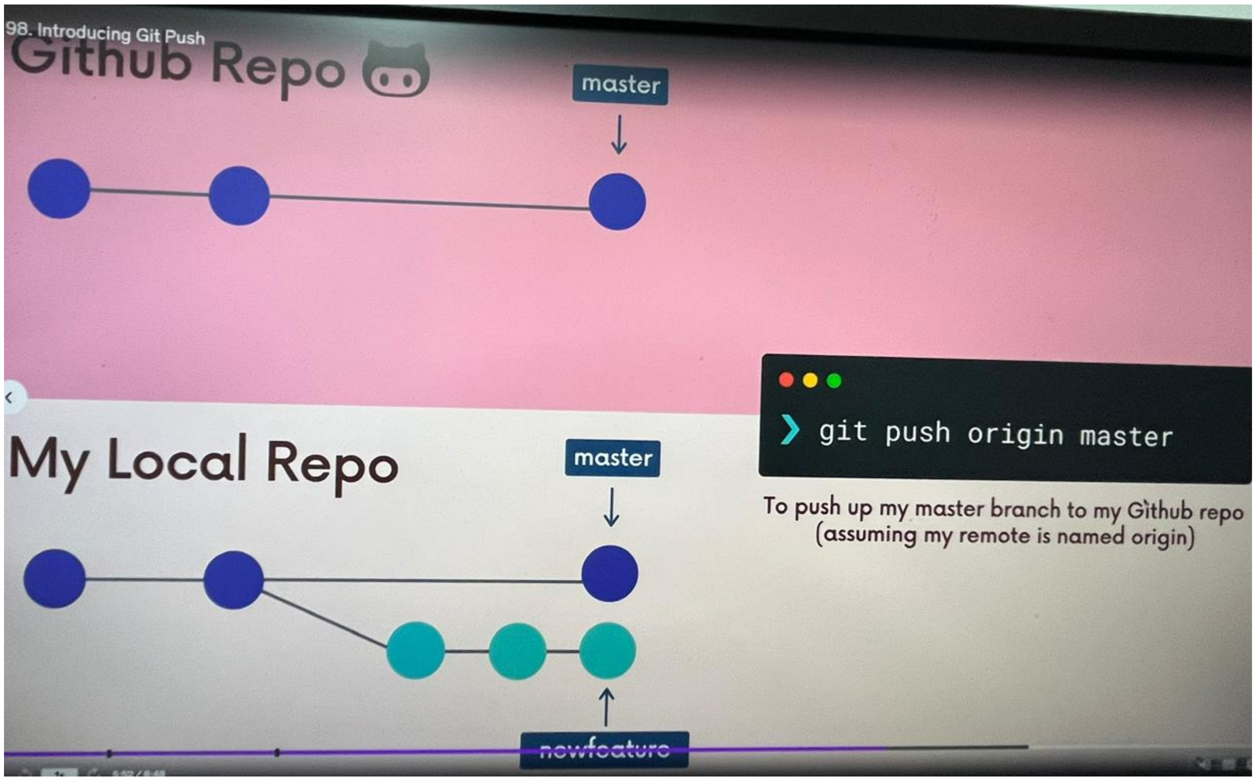
- Other commands

They are not commonly used, but there are commands to rename and delete remotes if needed.

Command Syntax: git remote rename <old> <new>

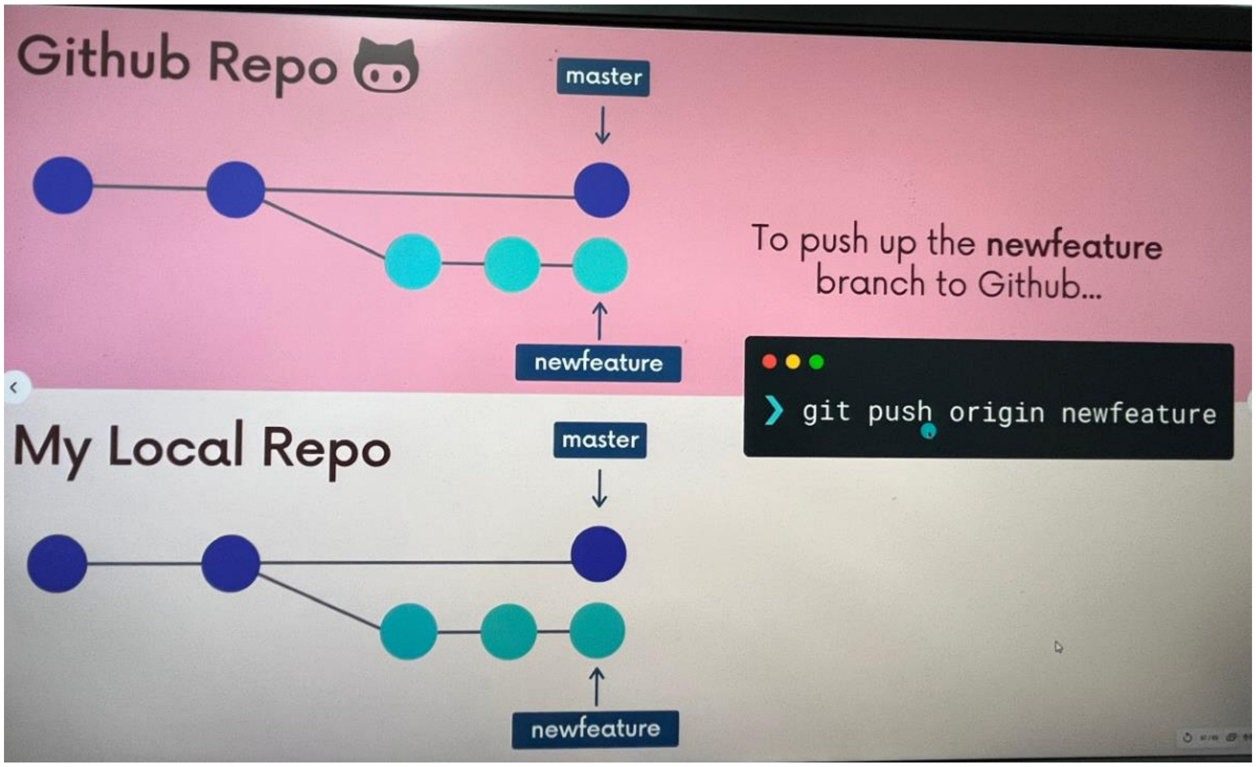
git remote remove <name>

7/ Introducing Git Push



```
❯ git push origin master
```

To push up my master branch to my Github repo (assuming my remote is named origin)



To push up the **newfeature** branch to Github...

```
❯ git push origin newfeature
```

- Pushing

Now that we have a remote set up, let's push some work up to Github!. To do this, we need to use the git push command.

We need to specify the remote we want to push up to AND the specific local branch we want to push up to that remote.

Command syntax: git push <remote> <branch>.

Example: git push origin master

git push origin master tells git to push up the master branch to our origin remote.

- Push in detail

While we often want to push a local branch up to a remote branch of the same name, we dont't have to!

To push our local pancake branch up to a remote branch called waffle we could do: git push origin pancake:waffle

Command syntax: git push <remote> <local-branch>:<remote-branch>

Example: git push origin pancake:waffle

- The -u option

The -u option allows us to set the upstream of the branch we're pushing. You can think of this as a link connecting our local branch to a branch on Github

Running **git push -u origin master** sets the upstream of the local master branch so that it tracks the master branch on the origin repo.

Command Syntax: git push -u origin master

What this means....

Once we've set the upstream for a branch, we can use the git push shorthand which will push our current branch to the upstream.

Command syntax:

git push -u origin master -> git push

8/ Fetching and Pulling

- Remote Tracking Branches

"At the time you last communicated with this remote repository, here is where x branch was pointing"

They follow this pattern **<remote>/<branch>.**

+ origin/master references the state of the master branch on the remote repo named origin.

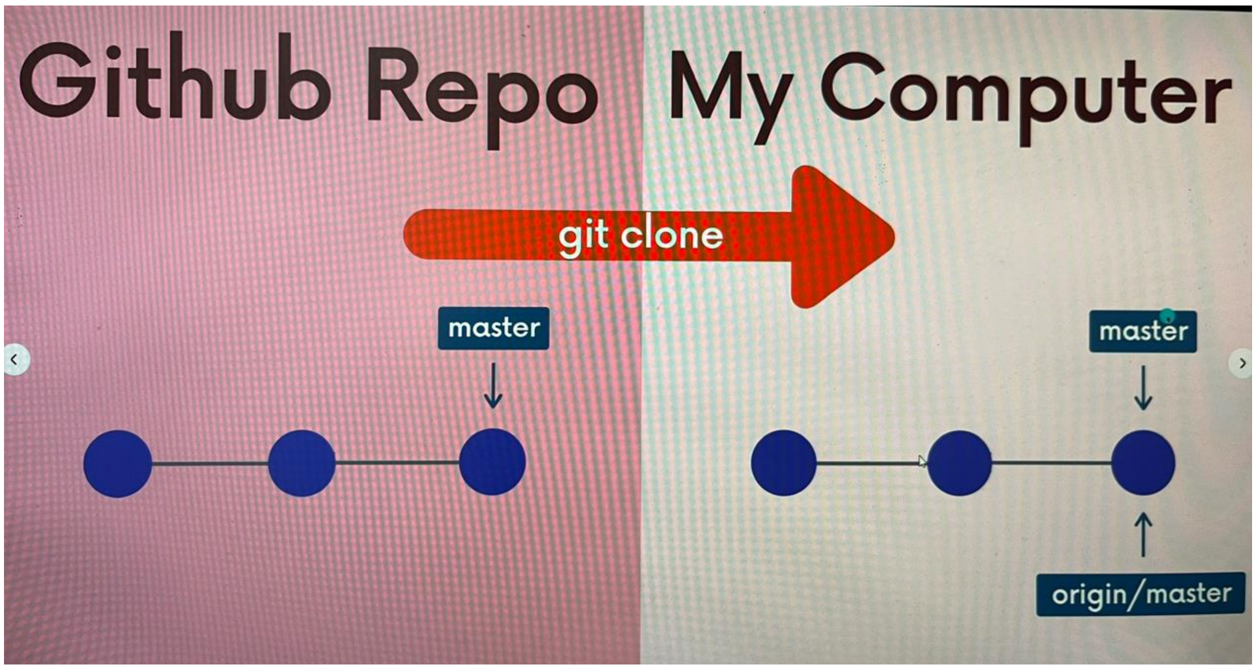
+ upstream/logoRedesign references the state of the logoRedesign branch on the remote named upstream (a common remote name).

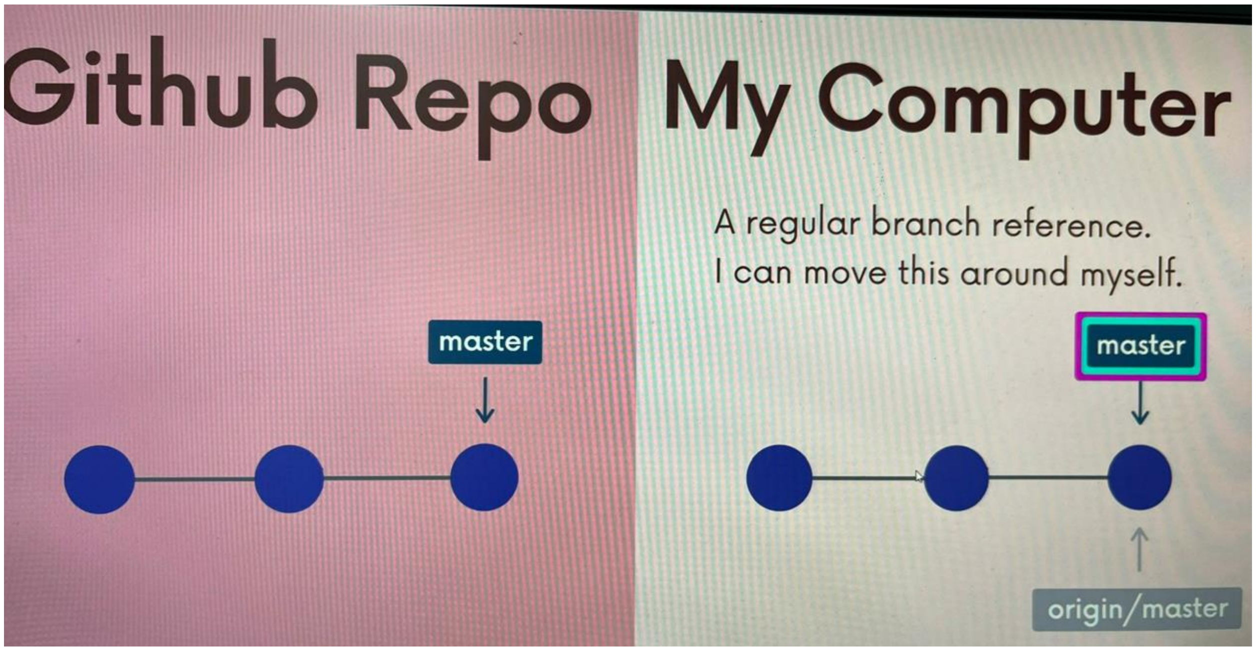
Remote branches

Run git branch -r to view the remote branches our local repository knows about

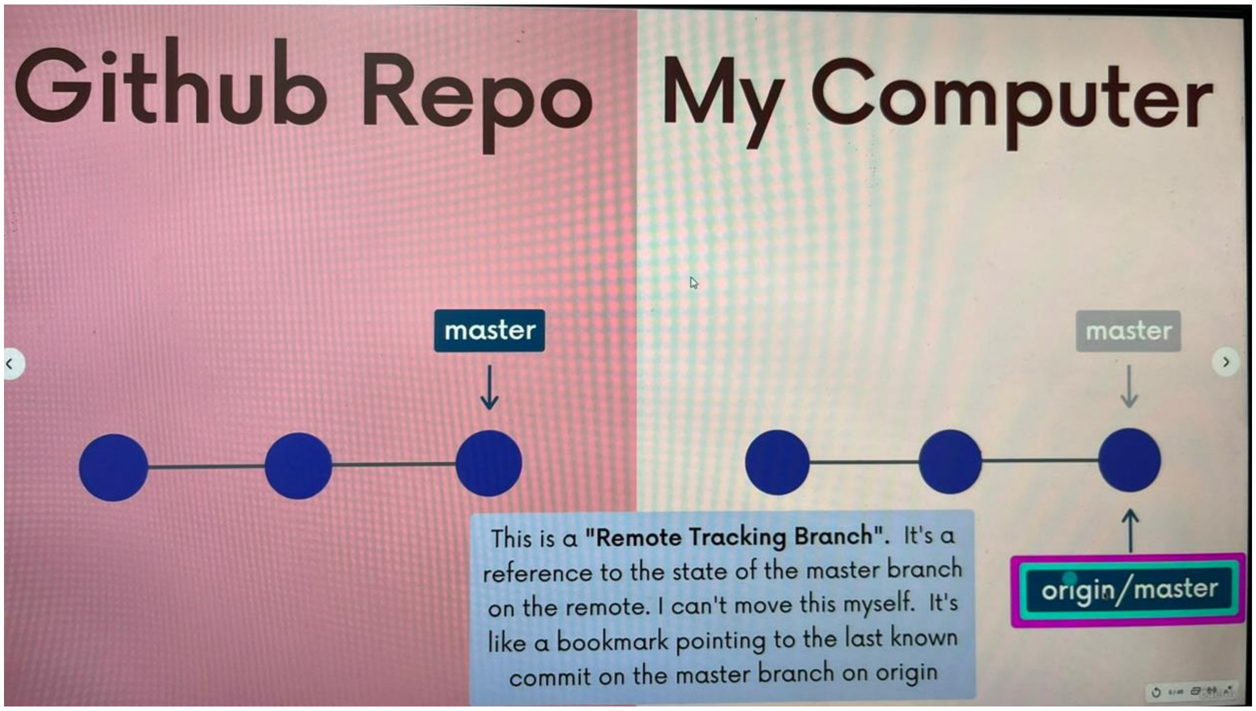
Command Syntax: git branch -r

Result: origin/master

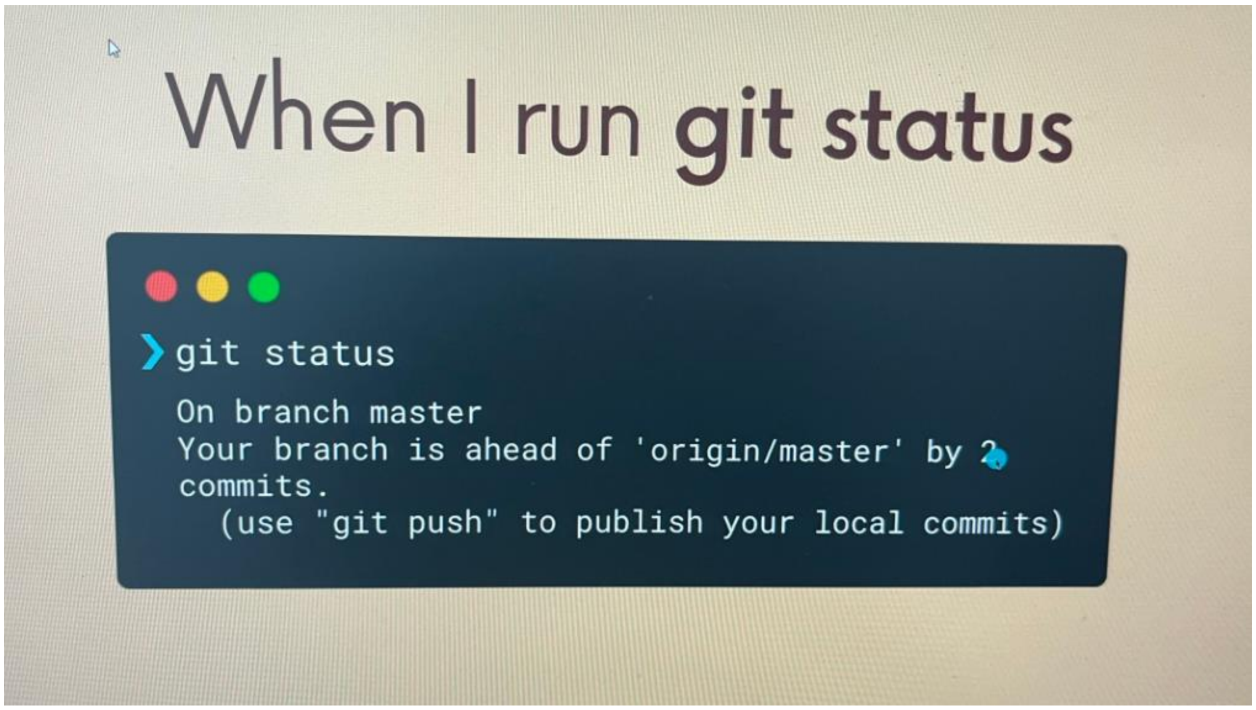




A regular branch reference.
I can move this around myself.

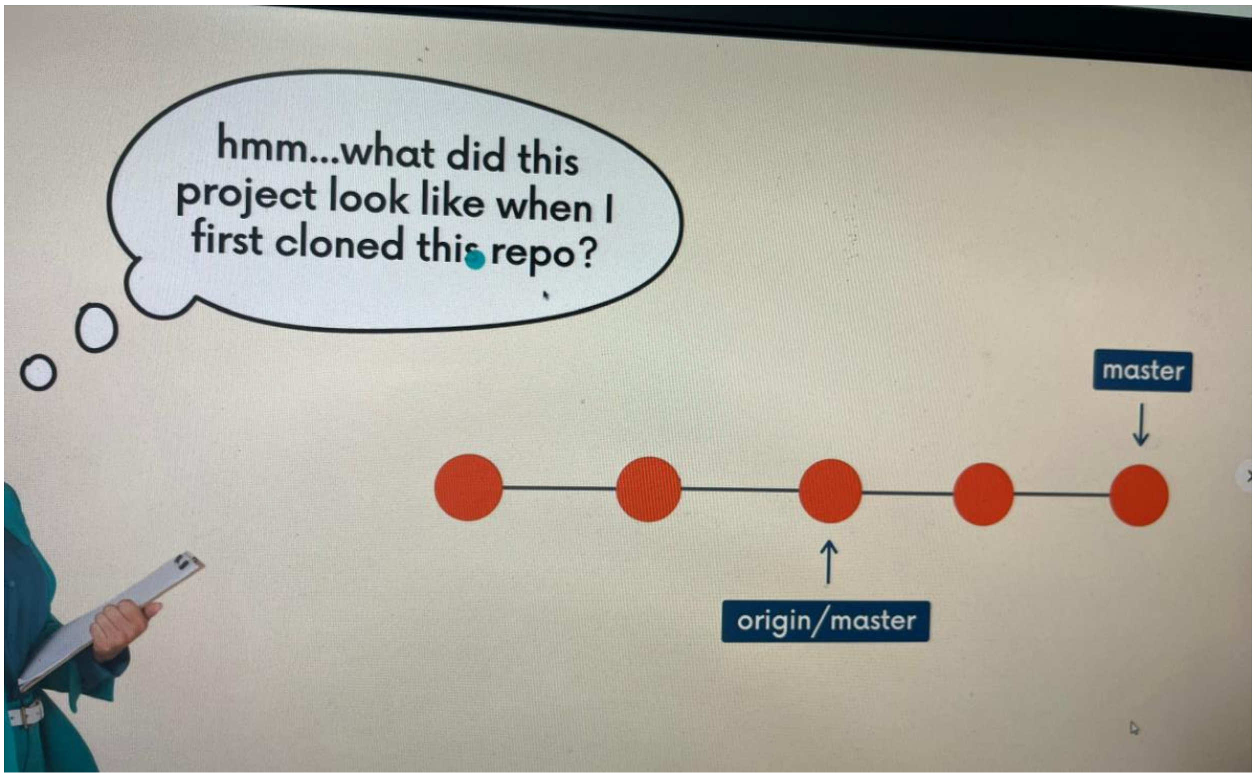


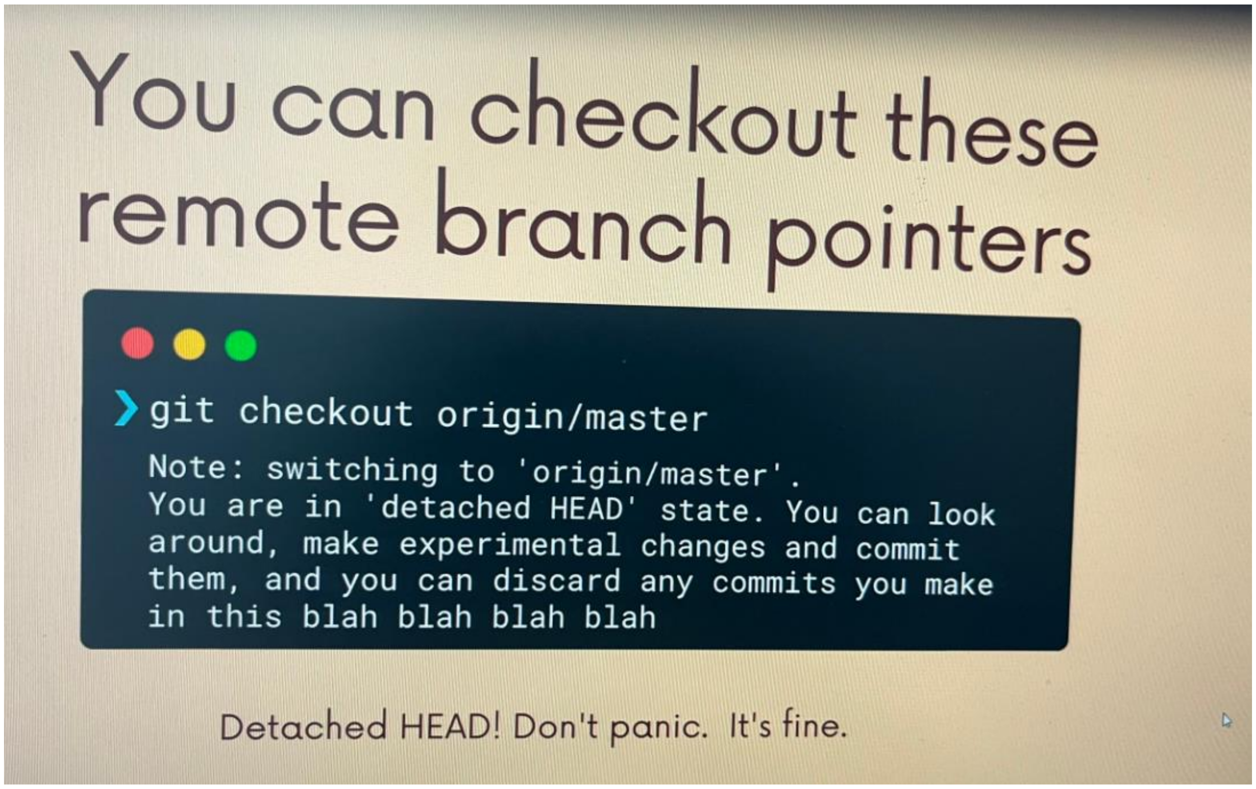
This is a **"Remote Tracking Branch"**. It's a reference to the state of the master branch on the remote. I can't move this myself. It's like a bookmark pointing to the last known commit on the master branch on origin



When I run **git status**

```
❯ git status
  On branch master
  Your branch is ahead of 'origin/master' by 2
  commits.
    (use "git push" to publish your local commits)
```


hmm...what did this project look like when I first cloned this repo?
master
origin/master



You can checkout these remote branch pointers

```
❯ git checkout origin/master
  Note: switching to 'origin/master'.
  You are in 'detached HEAD' state. You can look
  around, make experimental changes and commit
  them, and you can discard any commits you make
  in this blah blah blah blah
```

Detached HEAD! Don't panic. It's fine.

- Working with remote branches

Remote Branches

Once you've cloned a repository, we have all the data and Git history for the project at the moment in time.

However, that does not mean it's all in my workspace!

The github repo has a branch called puppies, but when i run git branch i do not see it on my machine.!!!

All i see is the master branch. What's going on?

If we use git branch -r we will see origin/puppies

I want to work on the puppies branch locally!

I could checkout origin/puppies, but that puts me in detached HEAD.

I want my own local branch called puppies, and i want it to be connected to origin/puppies, just like my local master branch is connected to origin/master.

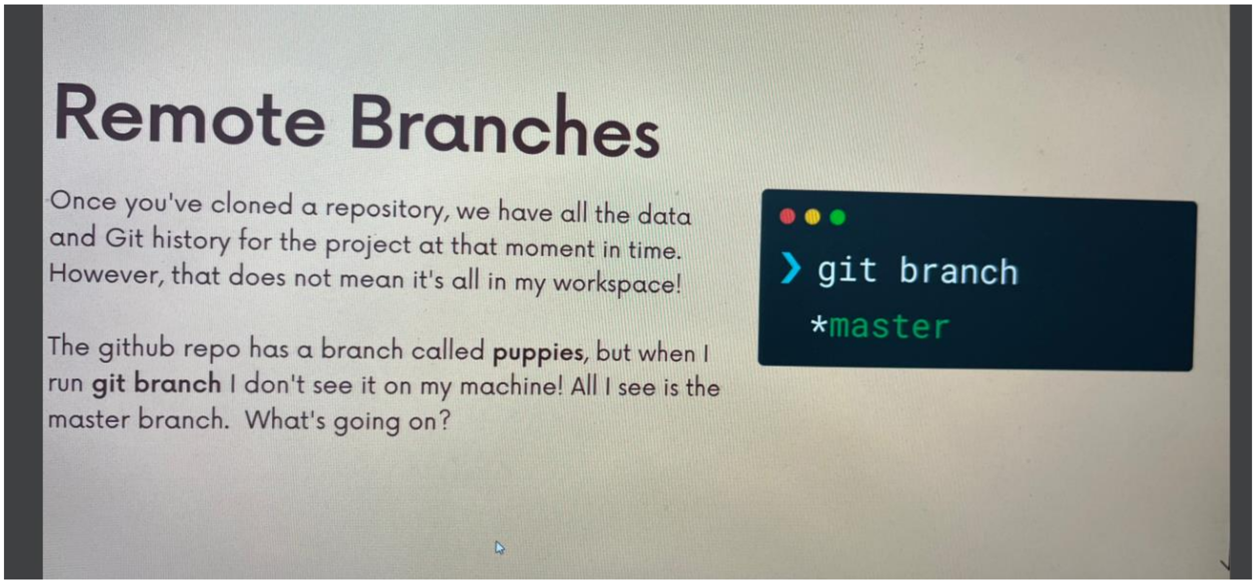
=> It's super easy!

Run git switch <remote-branch-name> to create a new local branch from the remote branch of the same name.

git switch puppies makes me a local puppies branch AND sets it up to track the remote branch origin/puppies

Command syntax: git switch <remote-branch-name>

Example: git switch puppies

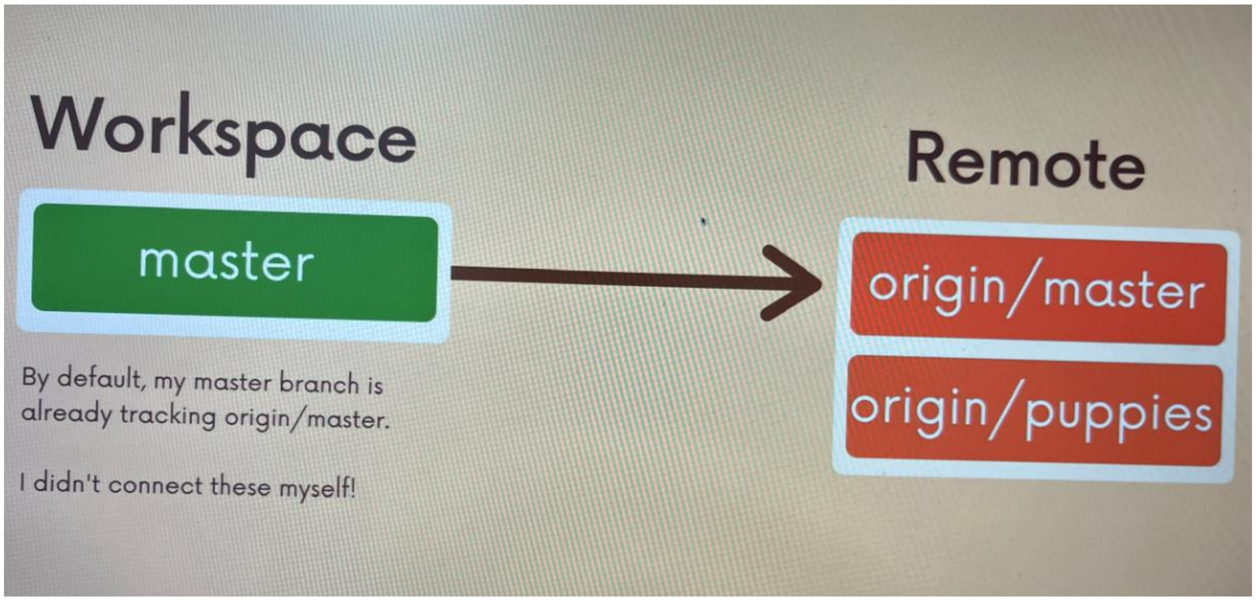


Remote Branches

Once you've cloned a repository, we have all the data and Git history for the project at that moment in time. However, that does not mean it's all in my workspace!

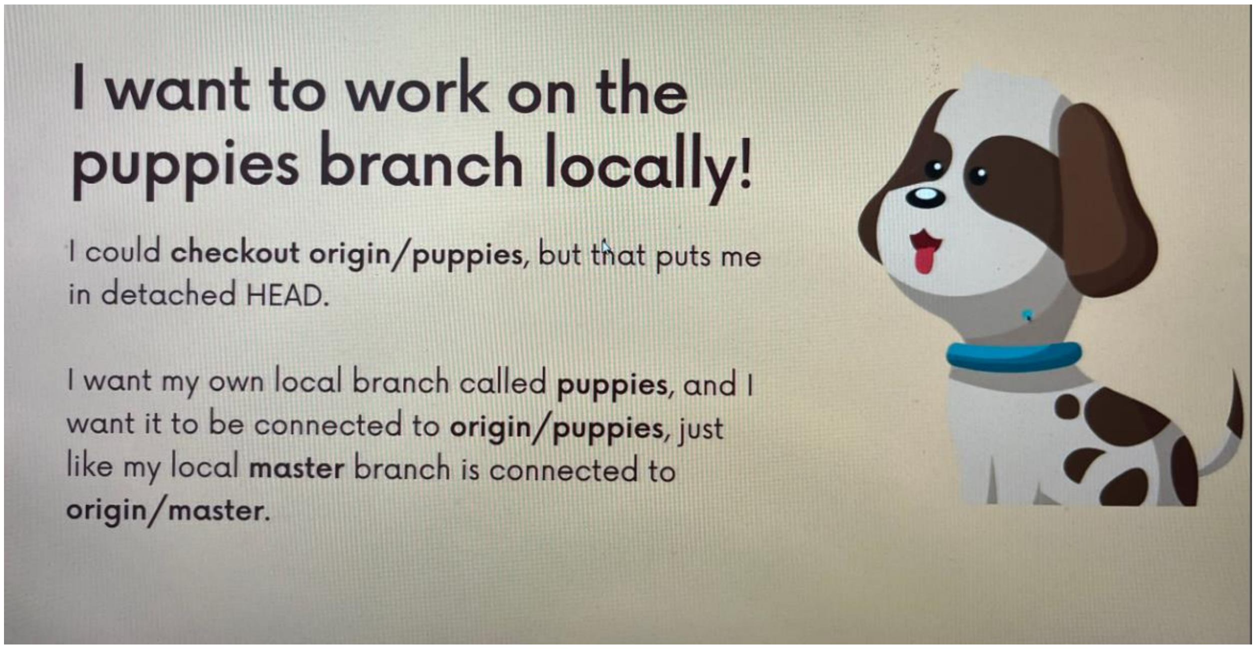
The github repo has a branch called **puppies**, but when I run **git branch** I don't see it on my machine! All I see is the master branch. What's going on?

```
❯ git branch
  *master
```



By default, my master branch is already tracking origin/master.

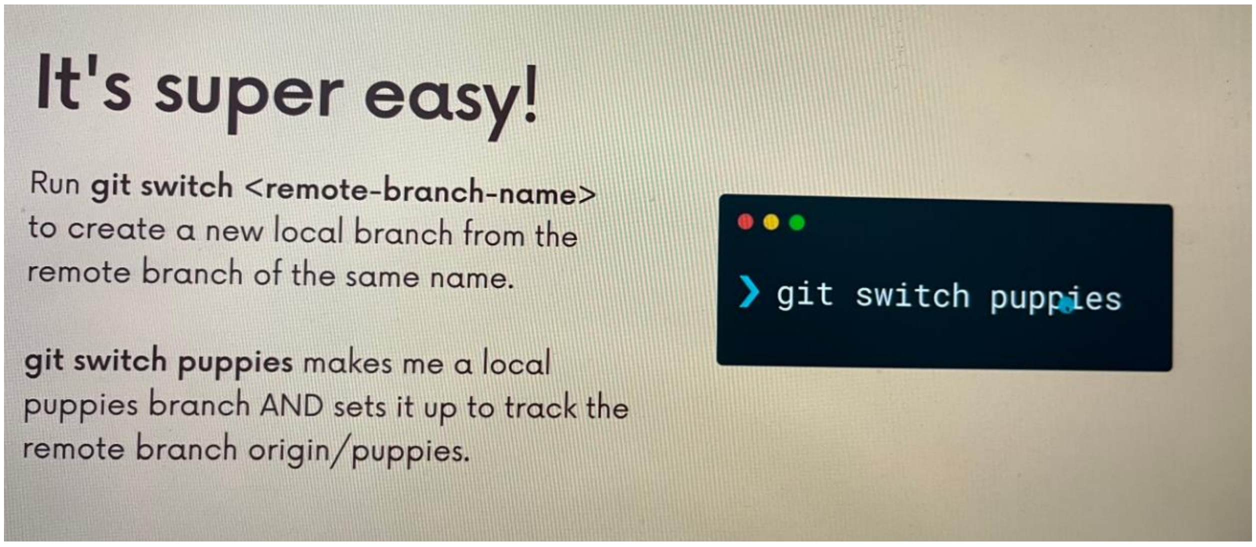
I didn't connect these myself!



I want to work on the puppies branch locally!

I could **checkout origin/puppies**, but that puts me in detached HEAD.

I want my own local branch called **puppies**, and I want it to be connected to **origin/puppies**, just like my local **master** branch is connected to **origin/master**.

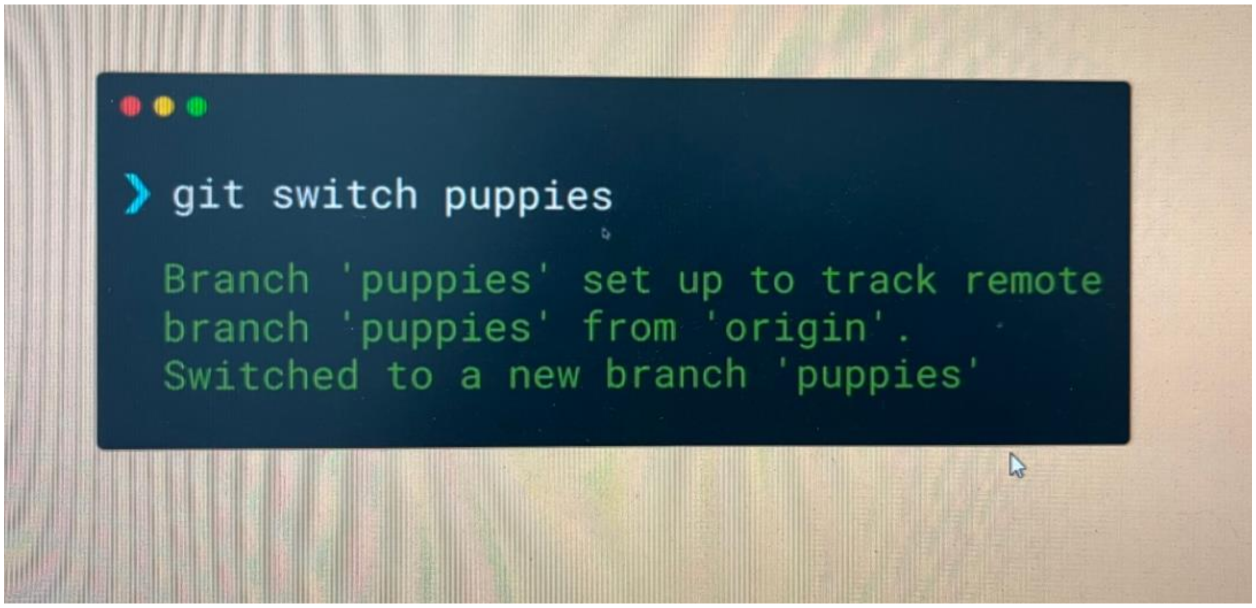


It's super easy!

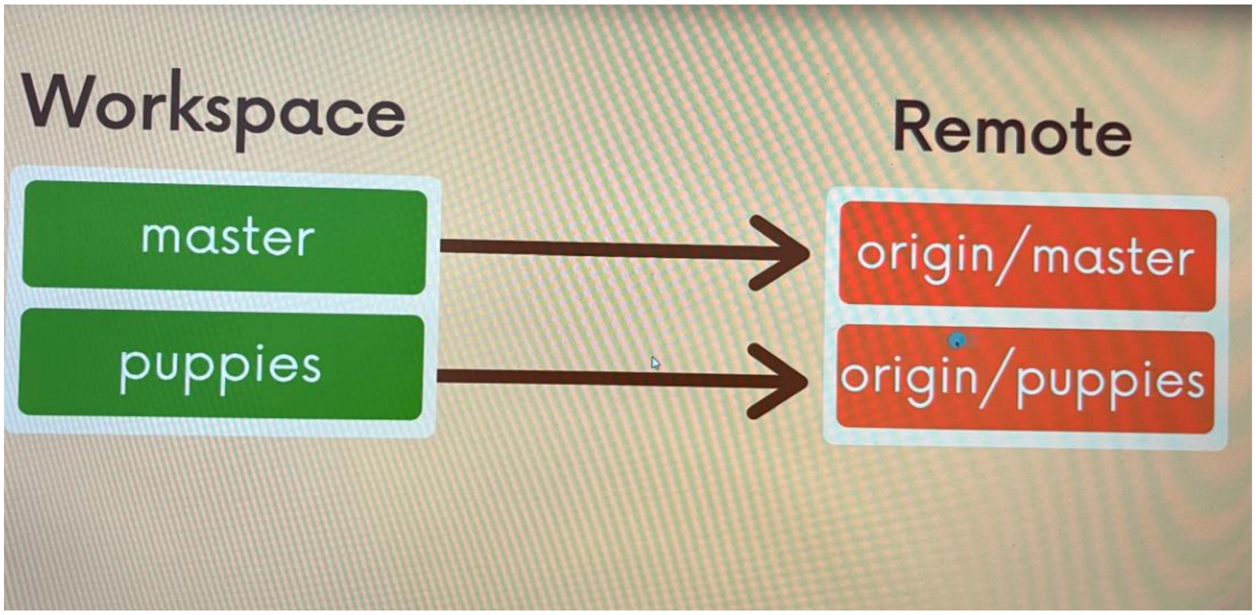
Run **git switch <remote-branch-name>** to create a new local branch from the remote branch of the same name.

git switch puppies makes me a local puppies branch AND sets it up to track the remote branch origin/puppies.

```
❯ git switch puppies
```



```
❯ git switch puppies
  Branch 'puppies' set up to track remote
  branch 'puppies' from 'origin'.
  Switched to a new branch 'puppies'
```

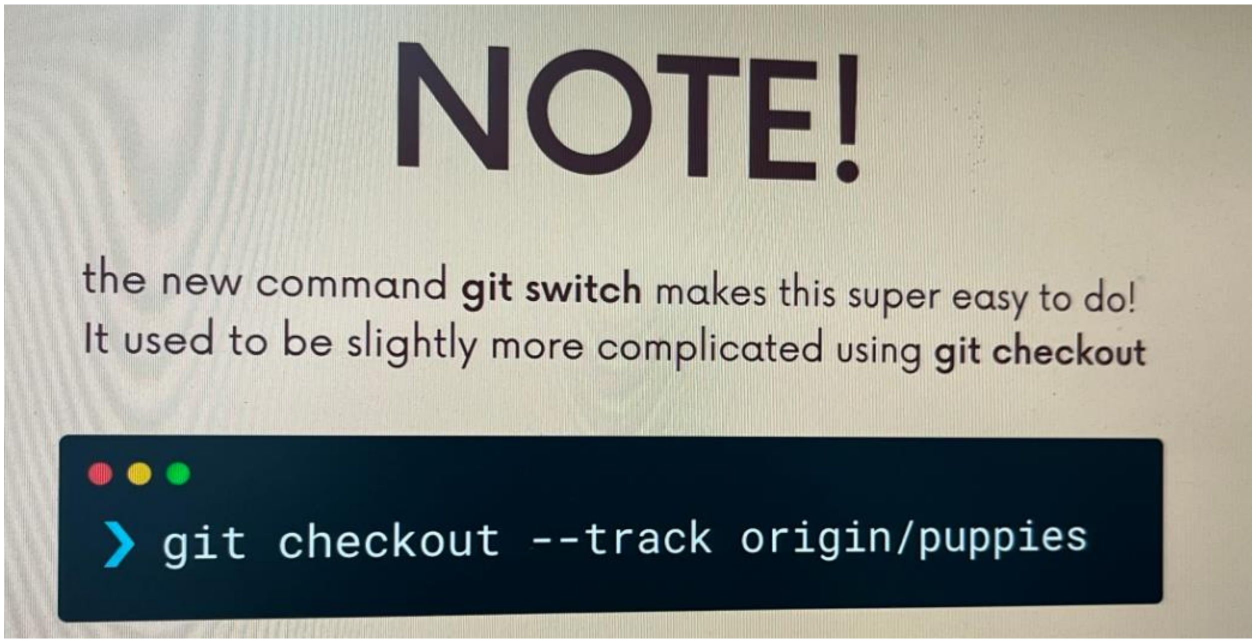


NOTE:

The new command git switch makes this super easy to do! It used to be slightly more complicated using git checkout!

Command syntax: git checkout --track origin/<branch-name>

Example: git checkout --track origin/pupies

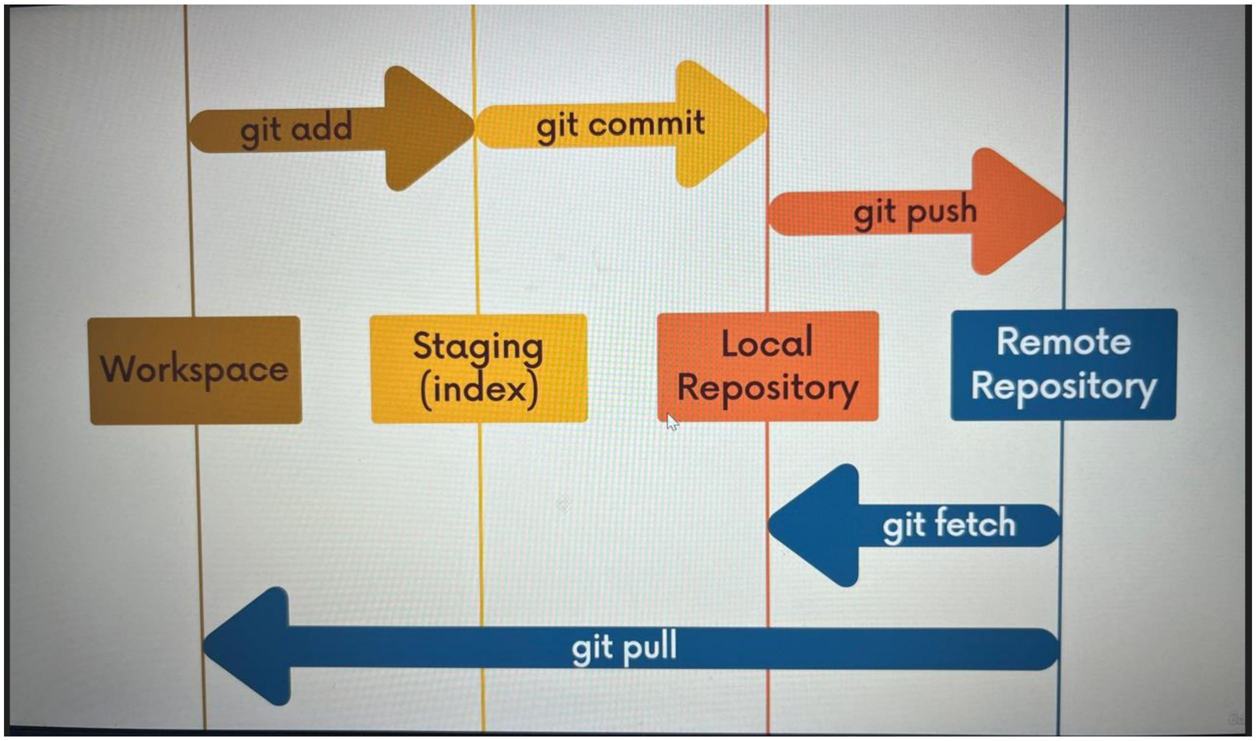


NOTE!

the new command **git switch** makes this super easy to do!
It used to be slightly more complicated using **git checkout**

```
❯ git checkout --track origin/puppies
```

- Git Fetch: The basics



Fetching:

Fetching allows us to download changes from a remote repository, BUT those changes will not be automatically integrated into our working files.

It lets you see what others have been working on, without having to merge those changes into your local repo.

Think of it as “Please go and get the latest information from Github, but do not screw up my working directory”

Git Fetch

The git fetch <remote> command fetches branches and history from a specific remote repository. It only updates remote tracking branches.

Git fetch origin would fetch all changes from the origin remote repository

Command Syntax: git fetch <remote>

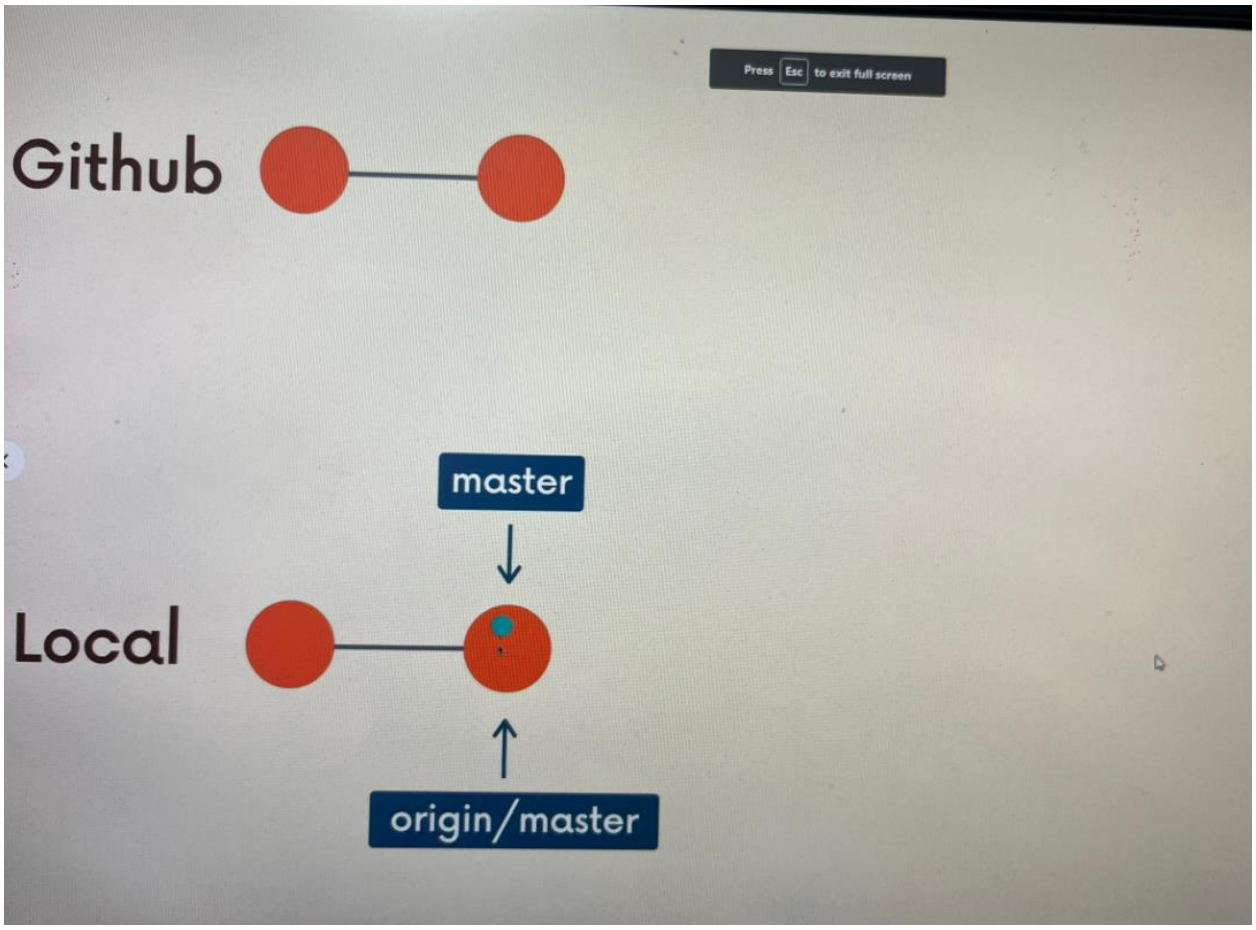
(If not specified, <remote> defaults to origin)

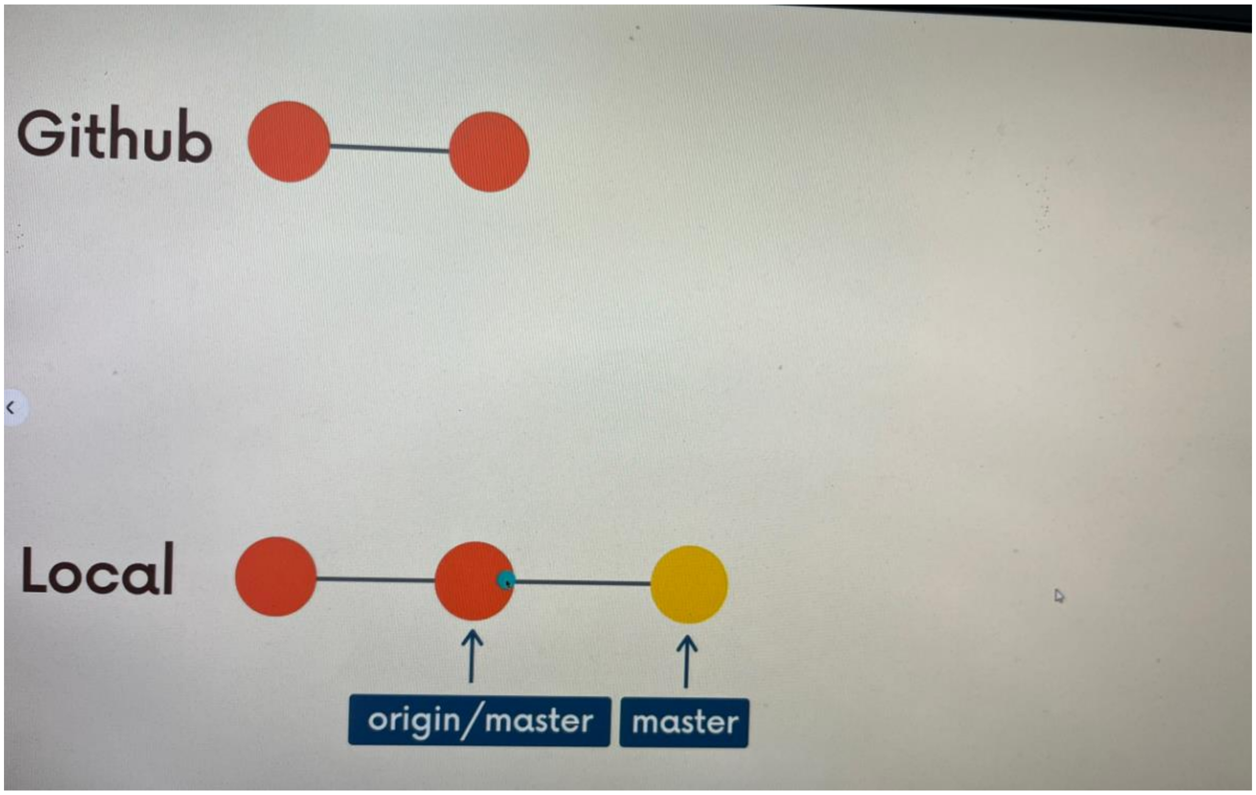
We can also fetch a specific branch from a remote using git fetch <remote> <branch>

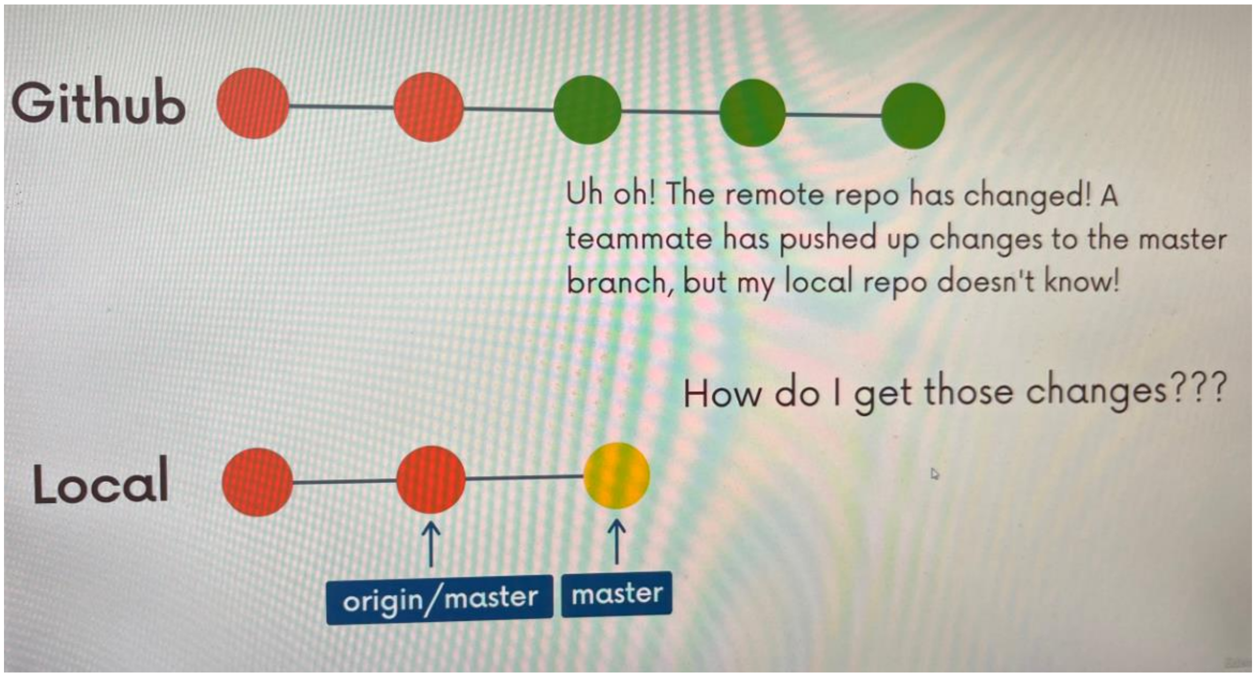
For example, git fetch origin master would retrieve the latest information from the master branch on the origin remote repository.

Command Syntax: git fetch <remote> <branch>

I now have those changes on my machine, but if I want to see them I have to checkout origin/master. My master branch is untouched.

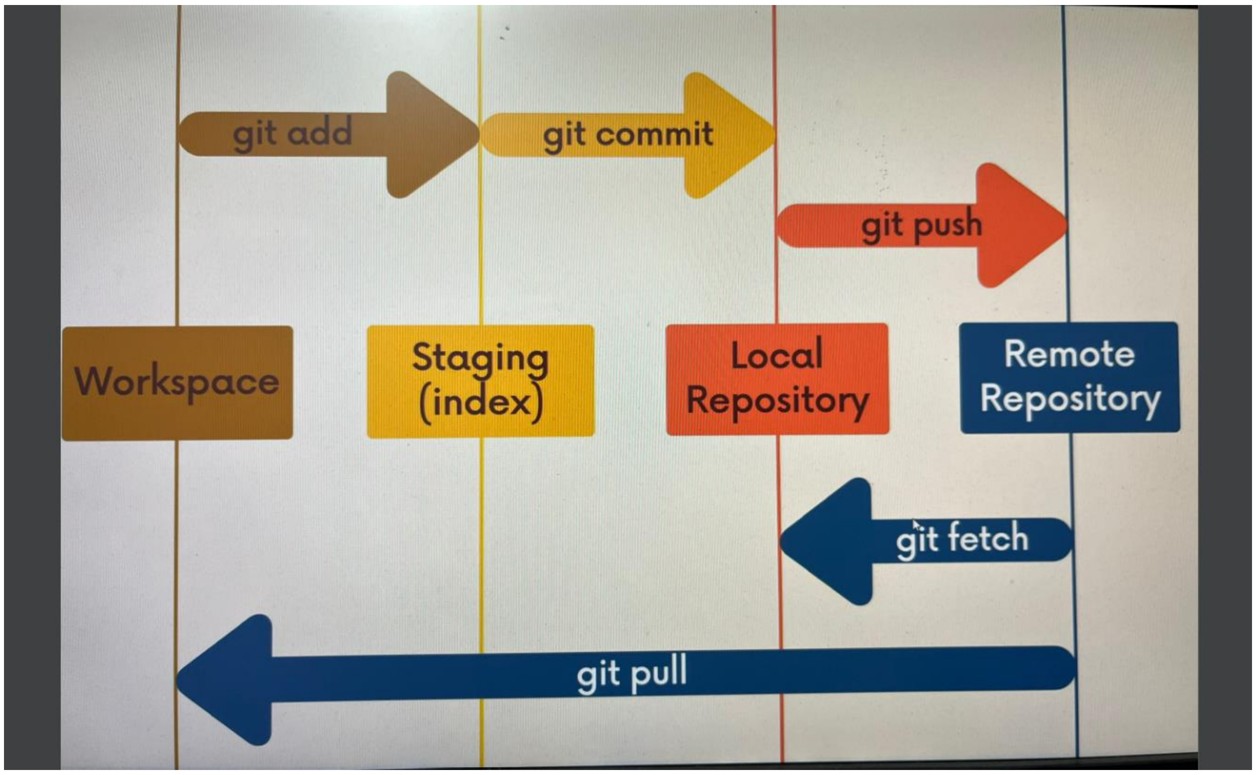

Press Esc to exit full screen
Github
master
Local
origin/master

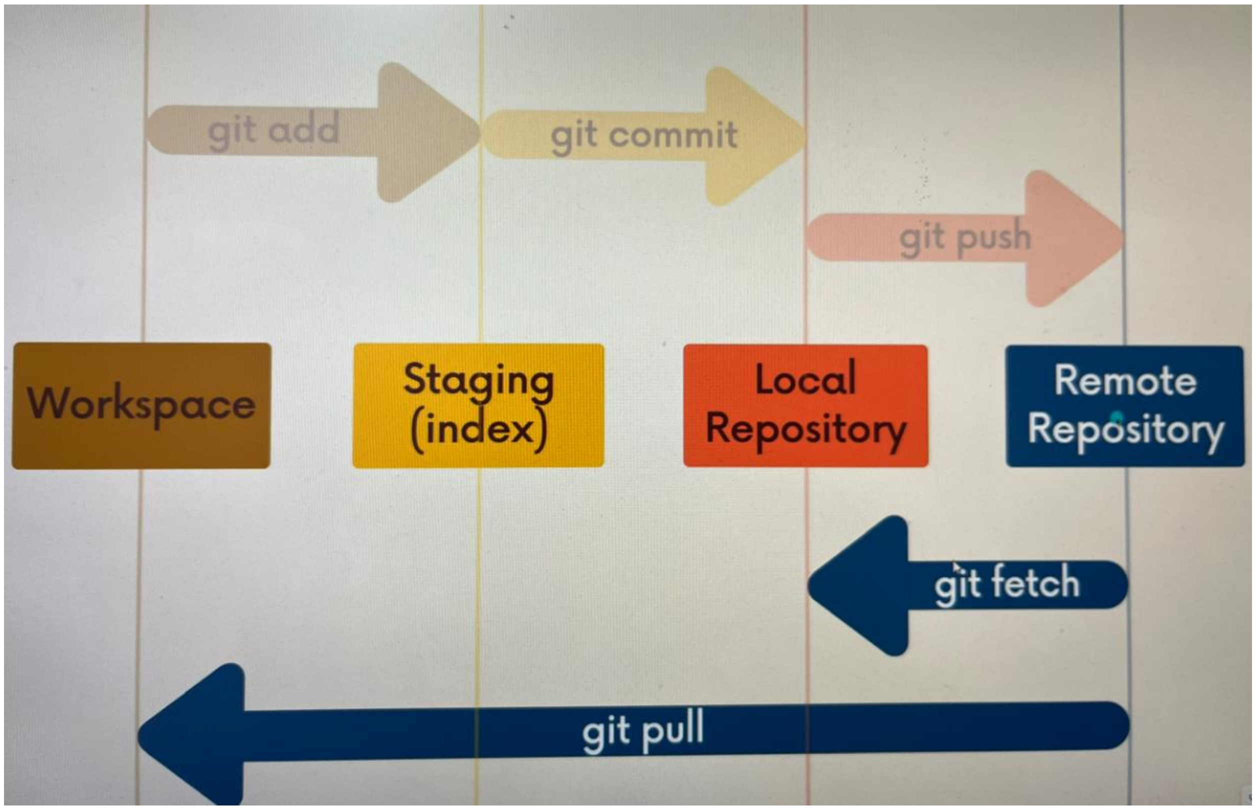


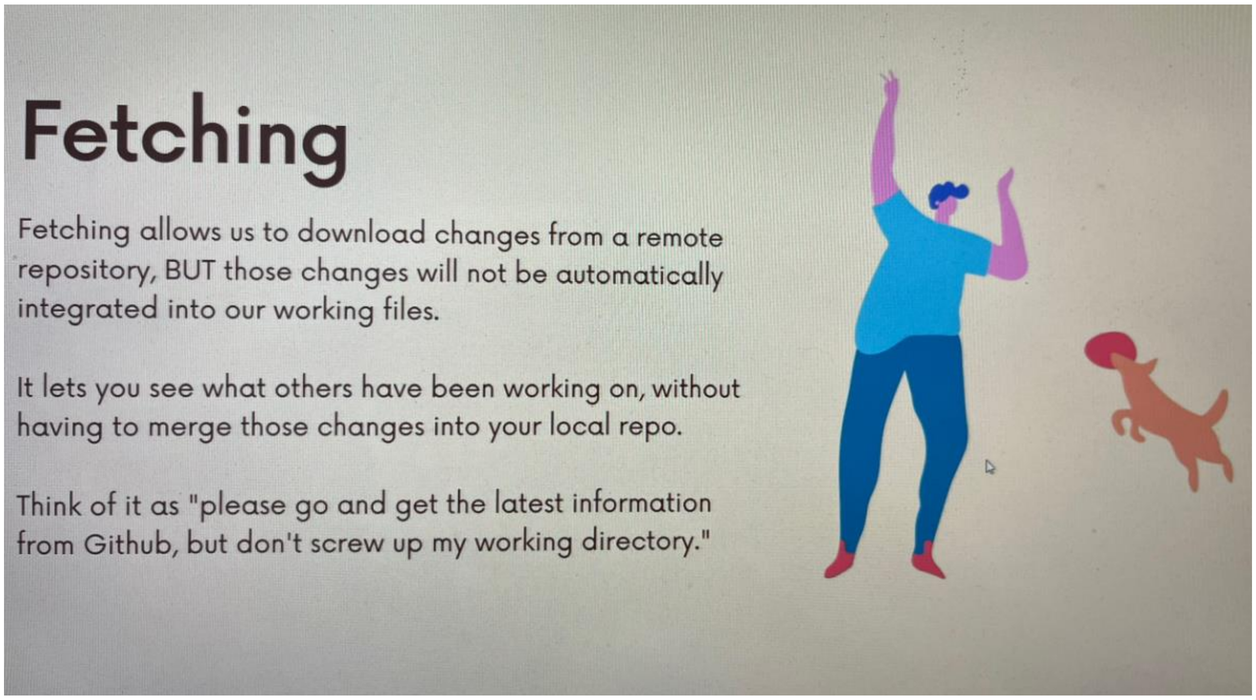


Uh oh! The remote repo has changed! A teammate has pushed up changes to the master branch, but my local repo doesn't know!

How do I get those changes???


git add
git commit
git push
Workspace
Staging (index)
Local Repository
Remote Repository
git fetch
git pull


git add
git commit
git push
Workspace
Staging (index)
Local Repository
Remote Repository
git fetch
git pull

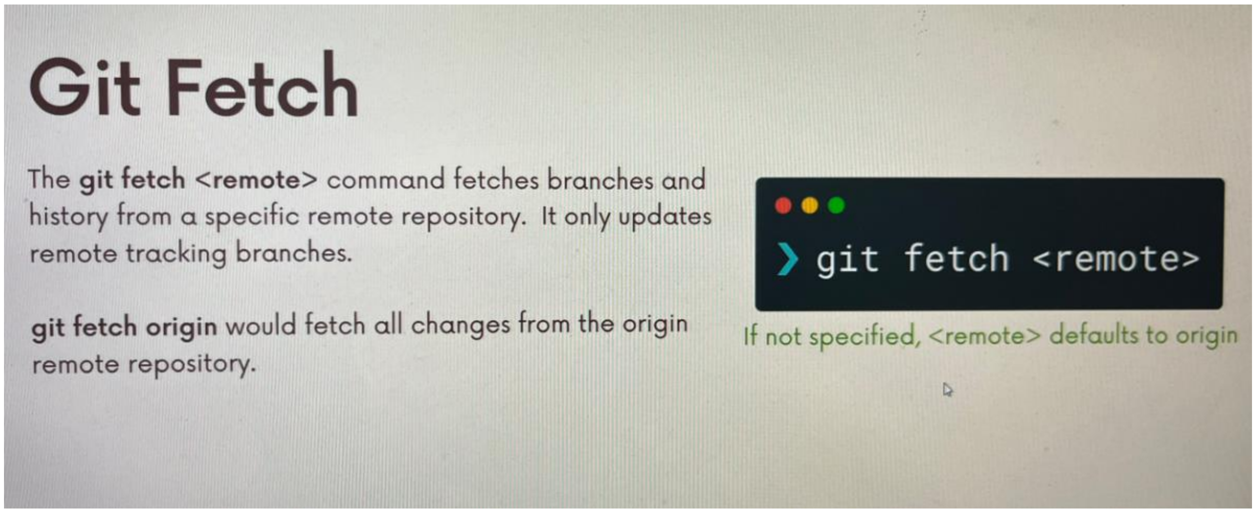


Fetching

Fetching allows us to download changes from a remote repository, BUT those changes will not be automatically integrated into our working files.

It lets you see what others have been working on, without having to merge those changes into your local repo.

Think of it as "please go and get the latest information from Github, but don't screw up my working directory."



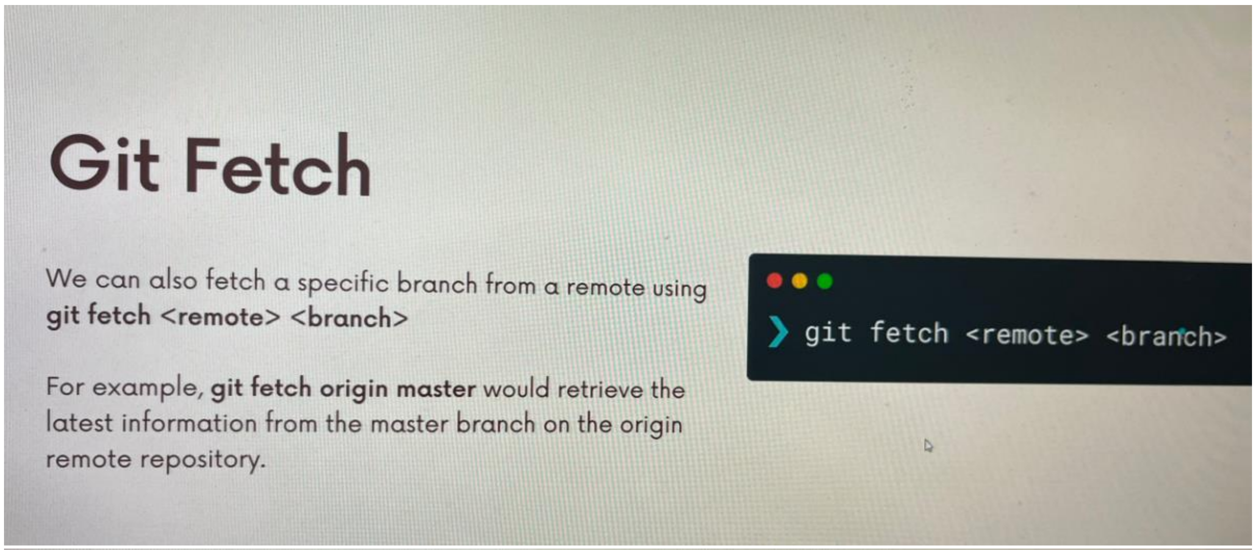
Git Fetch

The **git fetch <remote>** command fetches branches and history from a specific remote repository. It only updates remote tracking branches.

git fetch origin would fetch all changes from the origin remote repository.

```
❯ git fetch <remote>
```

If not specified, <remote> defaults to origin

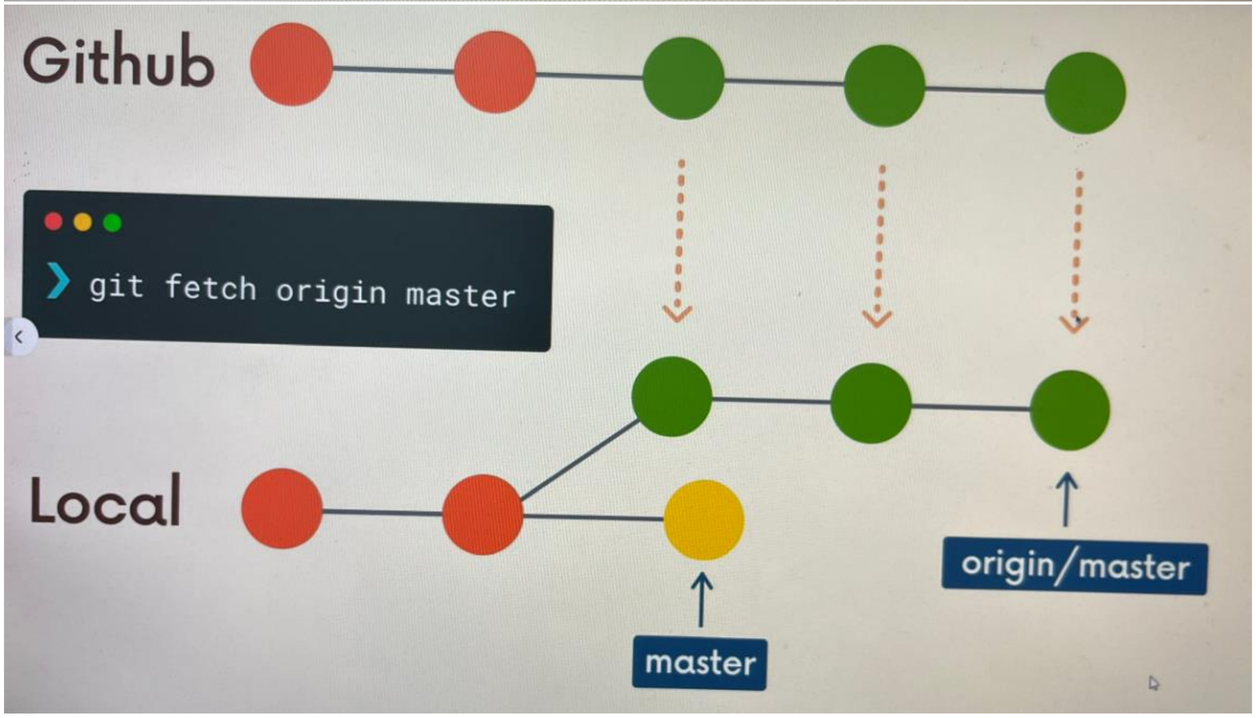


Git Fetch

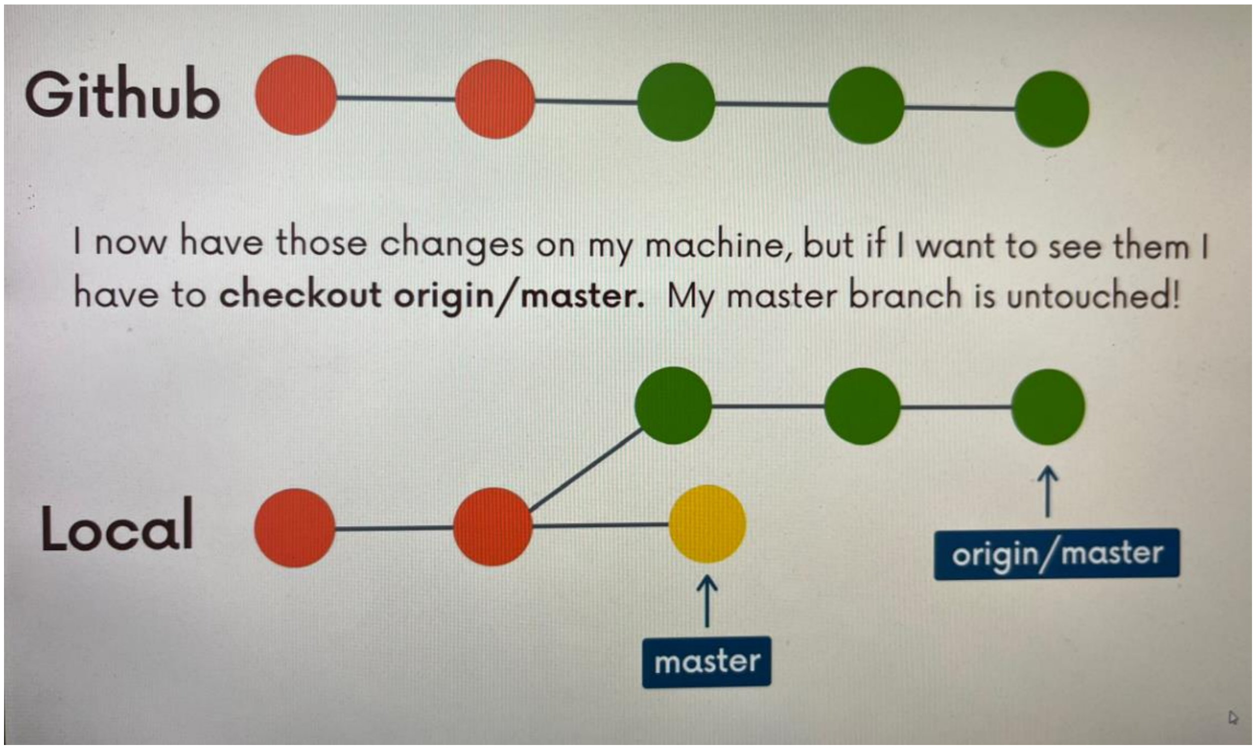
We can also fetch a specific branch from a remote using **git fetch <remote> <branch>**

For example, **git fetch origin master** would retrieve the latest information from the master branch on the origin remote repository.

```
❯ git fetch <remote> <branch>
```



```
❯ git fetch origin master
```



I now have those changes on my machine, but if I want to see them I have to **checkout origin/master.** My master branch is untouched!

-Git Pull: The basics

Pulling

Git pull is another command we can use to retrieve changes from a remote repository. Unlike fetch, pull actually updates our HEAD branch with whatever changes are retrieved from the remote.

“Go and download data from Github AND immediately update my local repo with those changes”

Git pull = git fetch + git merge

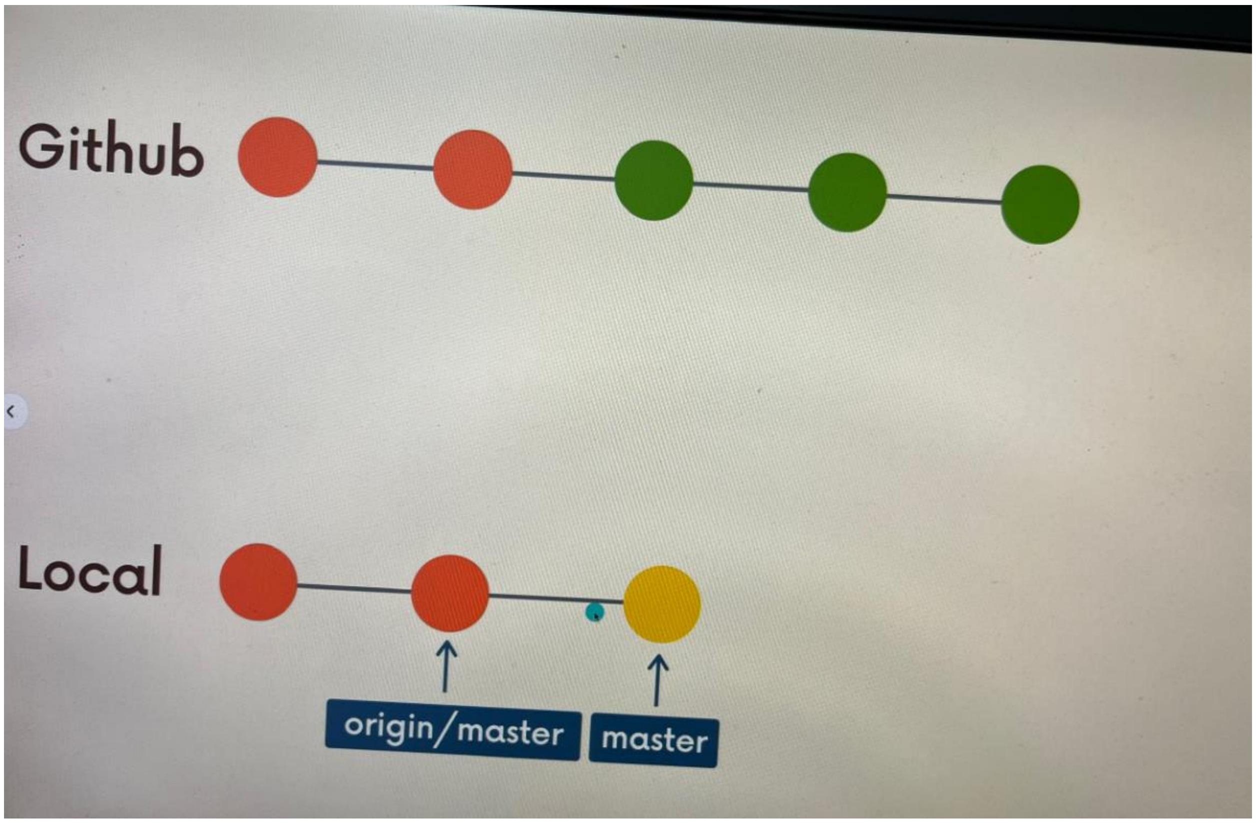
= update the remote tracking branch with latest changes from the remote repository + update my current branch with whatever changes are on the remote tracking branch

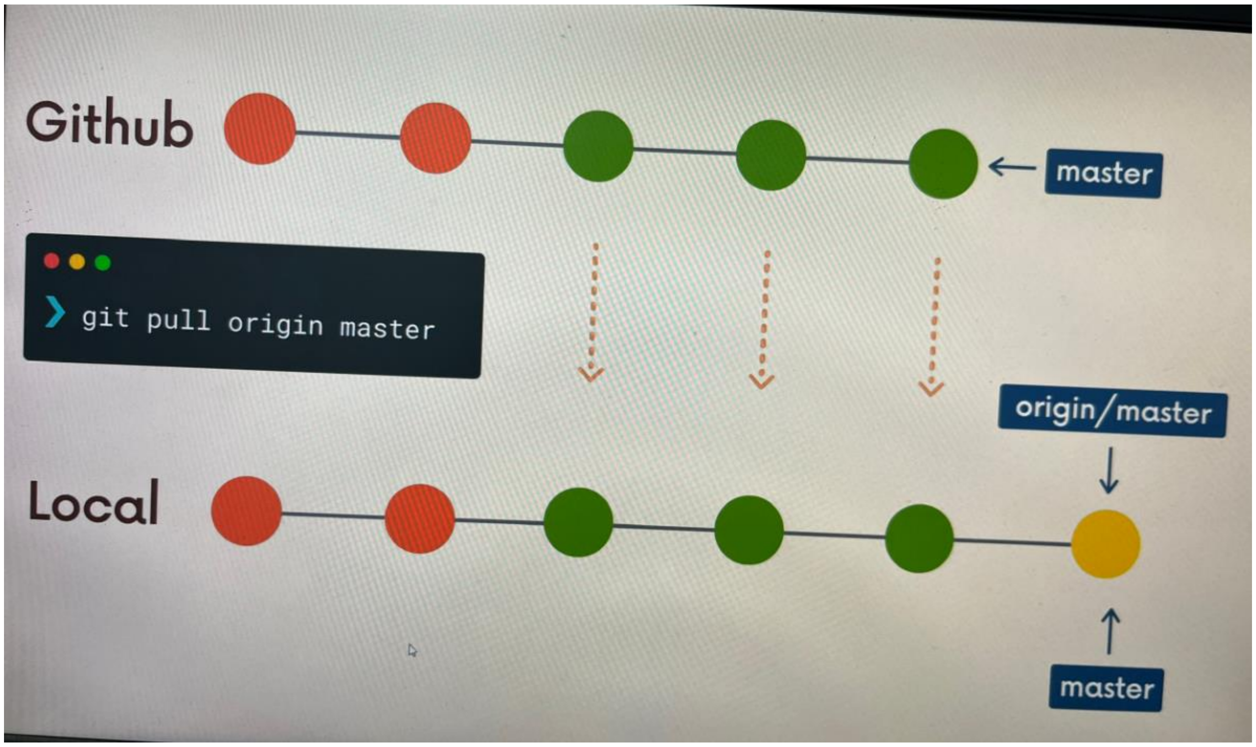
To pull, we specify the particular remote and branch we want to pull using git pull <remote> <branch>. Just like with git merge, it matters WHERE we run this command from. Whatever branch we run it from is where the changes will be merged into.

Git pull origin master would fetch the latest information from the origin’s master branch and merge those changes into our current branch.

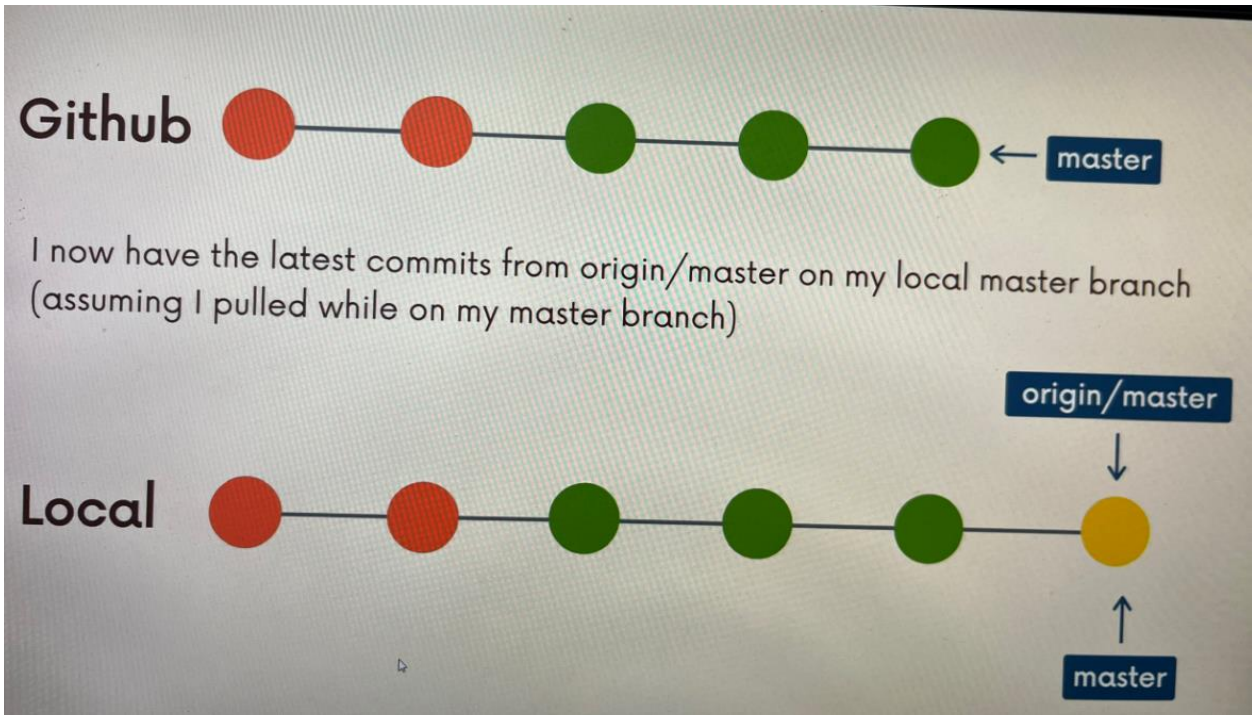
Command Syntax: git pull <remote> <branch>

Note: git pull can cause merge conflict

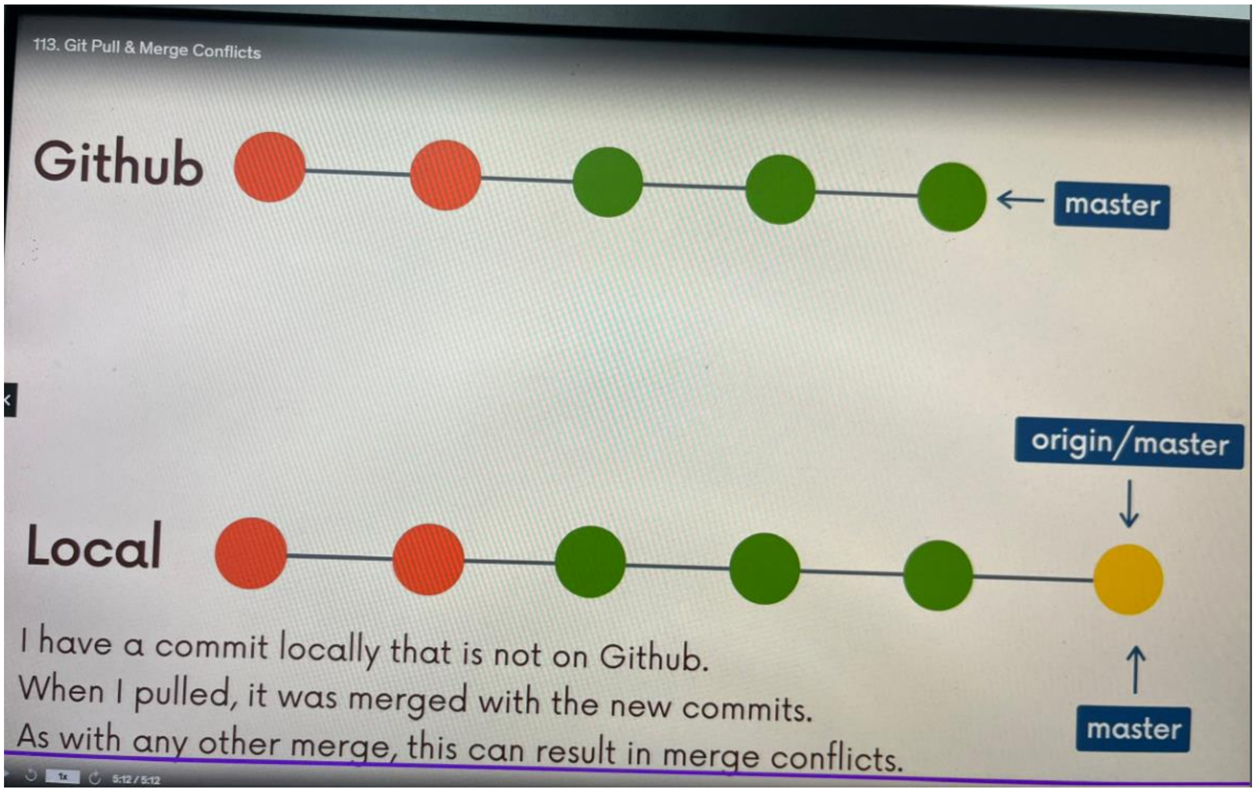




```
❯ git pull origin master
```



I now have the latest commits from origin/master on my local master branch (assuming I pulled while on my master branch)



I have a commit locally that is not on Github.
When I pulled, it was merged with the new commits.
As with any other merge, this can result in merge conflicts.

An even easier syntax!

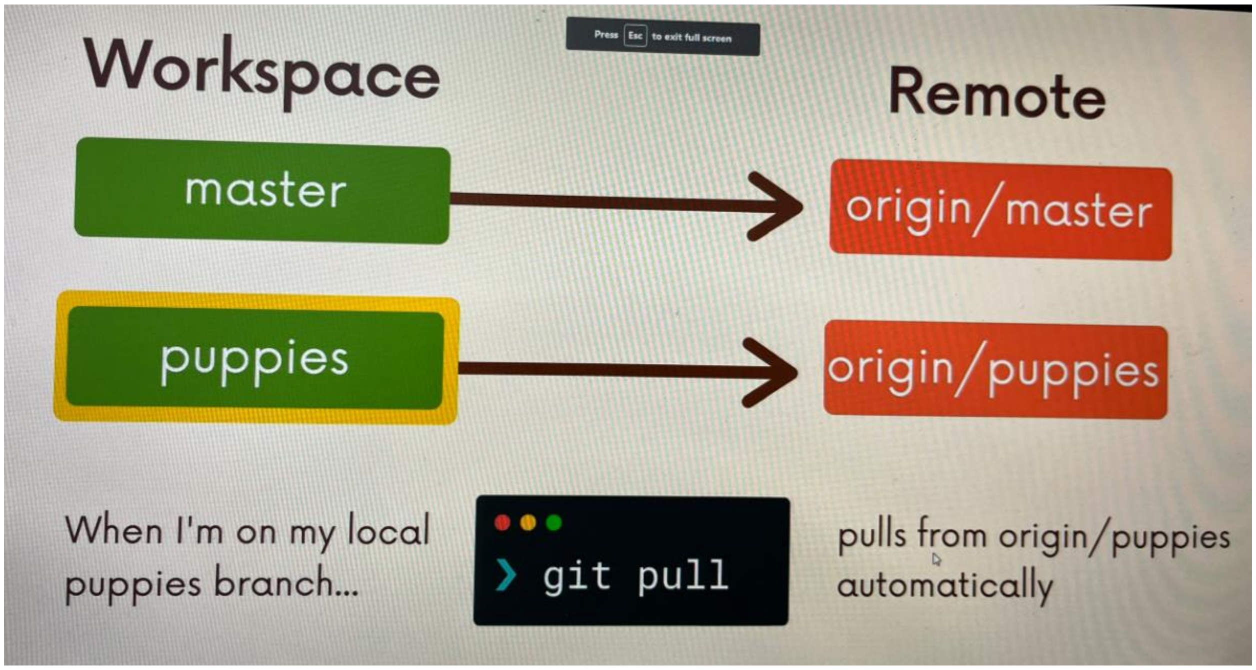
If we run git pull without specifying a particular remote or branch to pull from, git assumes the following:

+ remote will default to origin

+ branch will default to whatever tracking connection is configured for your current branch.

Note: this behavior can be configured, and tracking connections can be changed manually. Most people do not mess with that stuff.

Command syntax: git pull


Press Esc to exit full screen
Workspace
Remote
master
origin/master
puppies
origin/puppies
When I'm on my local puppies branch...
pulls from origin/puppies automatically

```
❯ git pull
```

Git fetch:

+ gets changes from remote branch(es)

+ updates the remote-tracking branches with the new changes

+ Does not merge changes onto your current HEAD branch

+ Safe to do at anytime

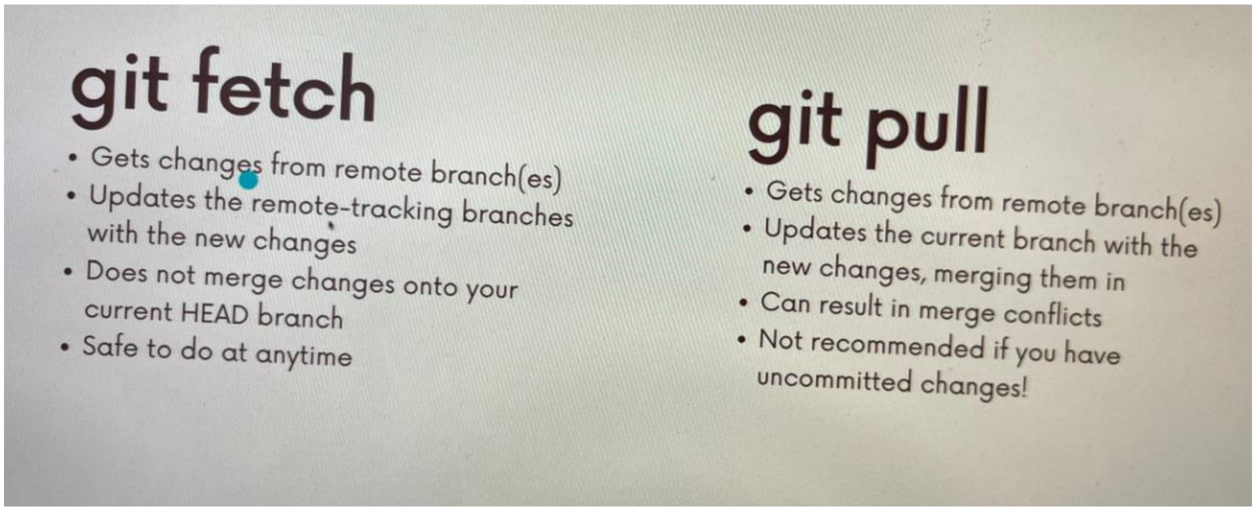
Git pull:

+ Gets changes from remote branch(es)

+ Updates the current branch with the new changes, merging them in

+ Can result in merge conflicts

+ Not recommend if you have uncommitted changes!



git fetch

- Gets changes from remote branch(es)
- Updates the remote-tracking branches with the new changes
- Does not merge changes onto your current HEAD branch
- Safe to do at anytime

git pull

- Gets changes from remote branch(es)
- Updates the current branch with the new changes, merging them in
- Can result in merge conflicts
- Not recommended if you have uncommitted changes!

9/ Github Grab Bag: Odds and Ends

- Github Repo Visibility: Public vs Private

+ Public repos are accessible to everyone on the internet. Anyone can see the repo on Github.

That still does NOT mean people can change the content of the repository. They can NOT push to the repository. The have to have collaborators privileges or permission to do that.

+ Private repos are only accessible to the owner and people who have been granted access.

Adding Collaborators

Working with collaborators , mean allow other users to push changes to the repository.

- READMEs:

A README file is used to communicate important information about a repository including:

+ What the project does

+ How to run the project

+ Why it's noteworthy

+ Who maintains the project

If you put a README in the root of your project, Github will regconize it and automatically display it on the repo's home page.

READMED.md

READMEs are markdown files, ending with the .md extension. Markdown is a convenient syntax to generate formatted text. It's easy to pick up!

- Github Gists

Github Gists are a simple way to share code snippets and useful fragments with others. Gists are much easier to create, but offer far fewer features than a typical Github repository.

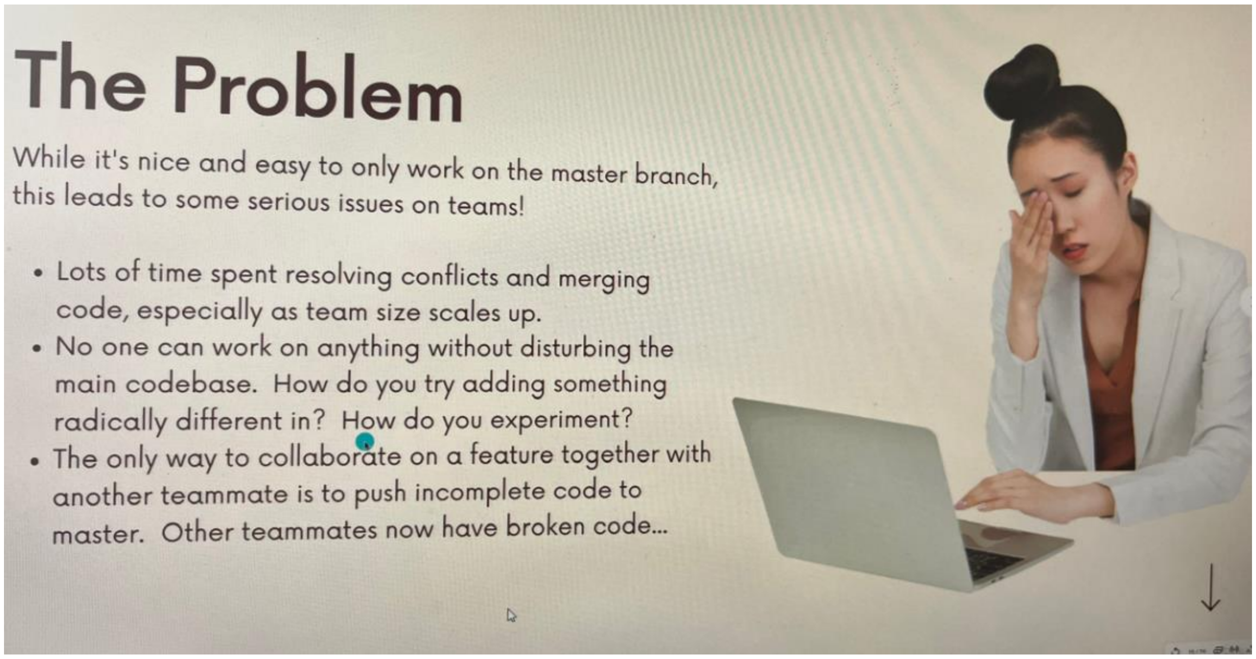
gh pages

Github Pages are public webpages that are hosted and published via Github. They allow you to create a website simply by pushing your code to Github.

Github Pages is a hosting service for static webpages, so it does not support server-side code like Python, Ruby, or Node. Just HTML/CSS/JS!.

10/ Git Collaborations Workflow

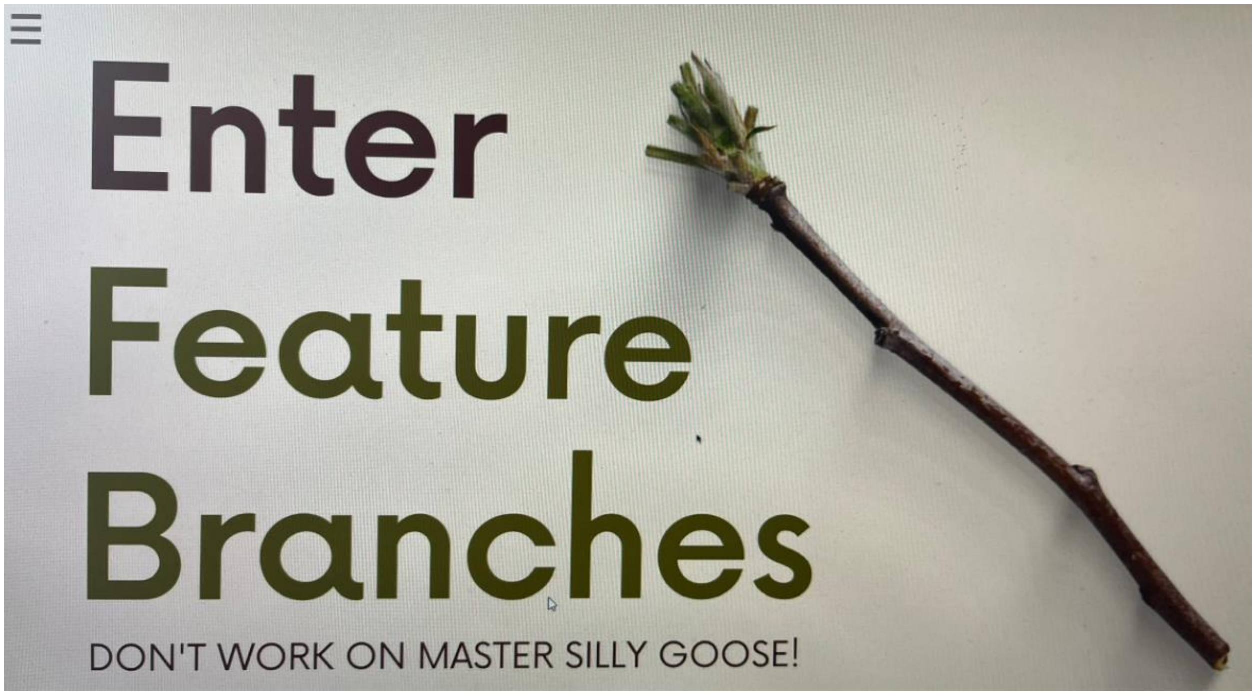
Collaboration Workflow



The Problem

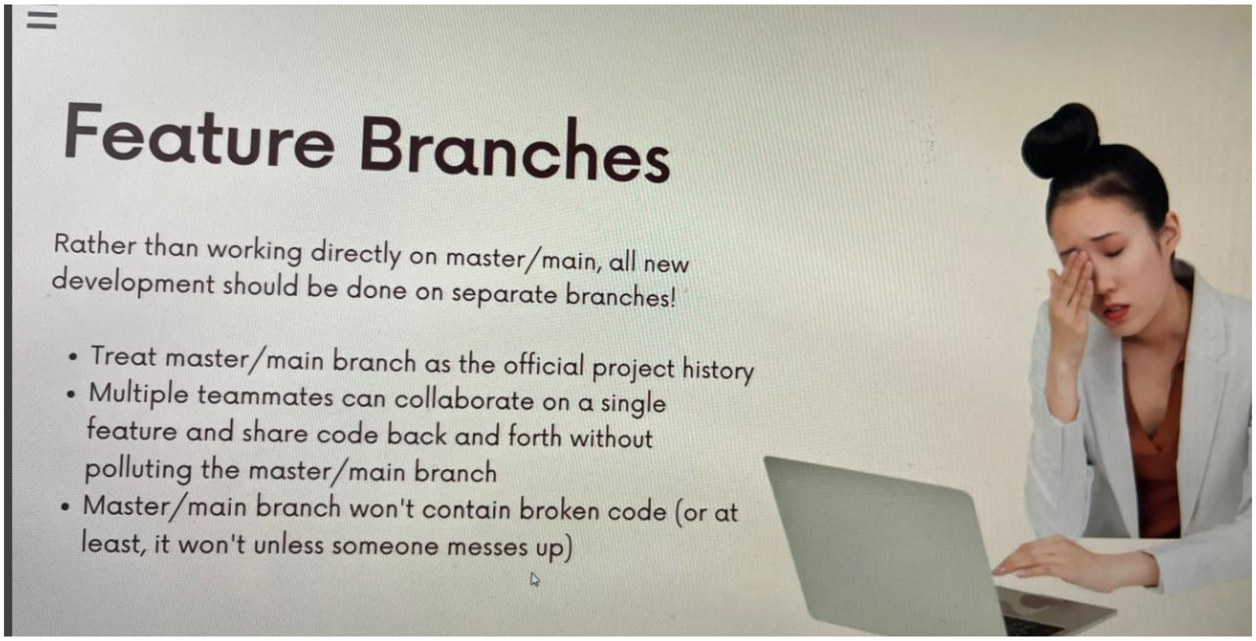
While it's nice and easy to only work on the master branch, this leads to some serious issues on teams!

- Lots of time spent resolving conflicts and merging code, especially as team size scales up.
- No one can work on anything without disturbing the main codebase. How do you try adding something radically different in? How do you experiment?
- The only way to collaborate on a feature together with another teammate is to push incomplete code to master. Other teammates now have broken code...



Enter Feature Branches

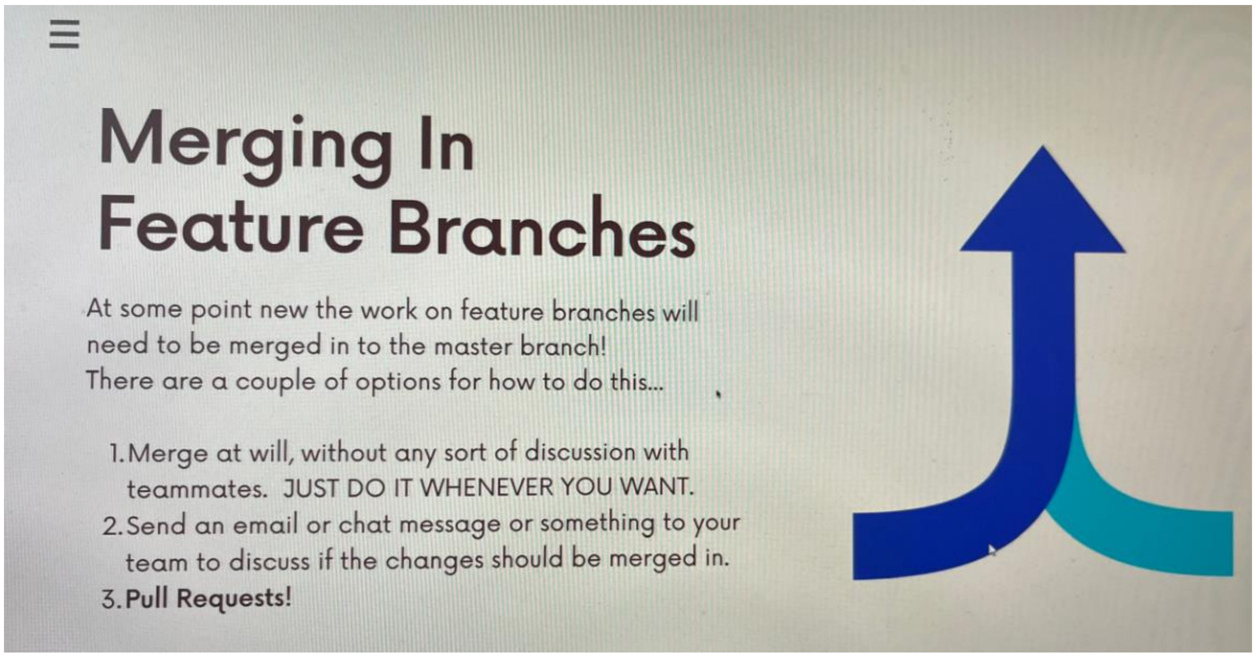
DON'T WORK ON MASTER SILLY GOOSE!



Feature Branches

Rather than working directly on master/main, all new development should be done on separate branches!

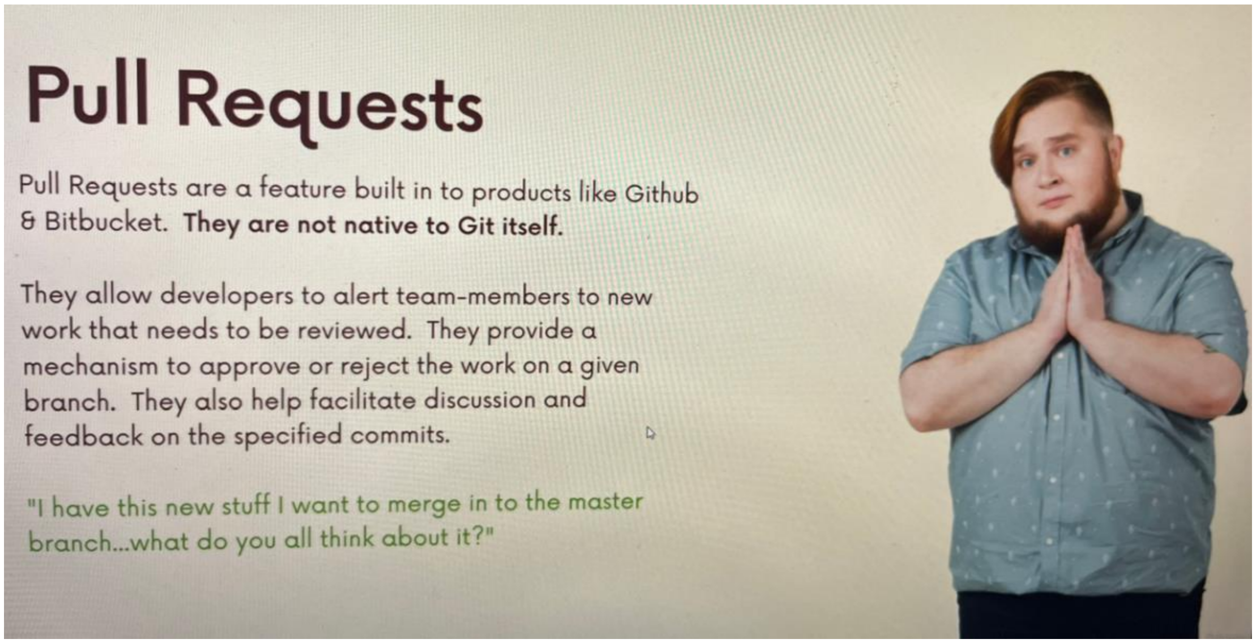
- Treat master/main branch as the official project history
- Multiple teammates can collaborate on a single feature and share code back and forth without polluting the master/main branch
- Master/main branch won't contain broken code (or at least, it won't unless someone messes up)



Merging In Feature Branches

At some point new the work on feature branches will need to be merged in to the master branch!
There are a couple of options for how to do this...

1. Merge at will, without any sort of discussion with teammates. JUST DO IT WHENEVER YOU WANT.
2. Send an email or chat message or something to your team to discuss if the changes should be merged in.
3. **Pull Requests!**

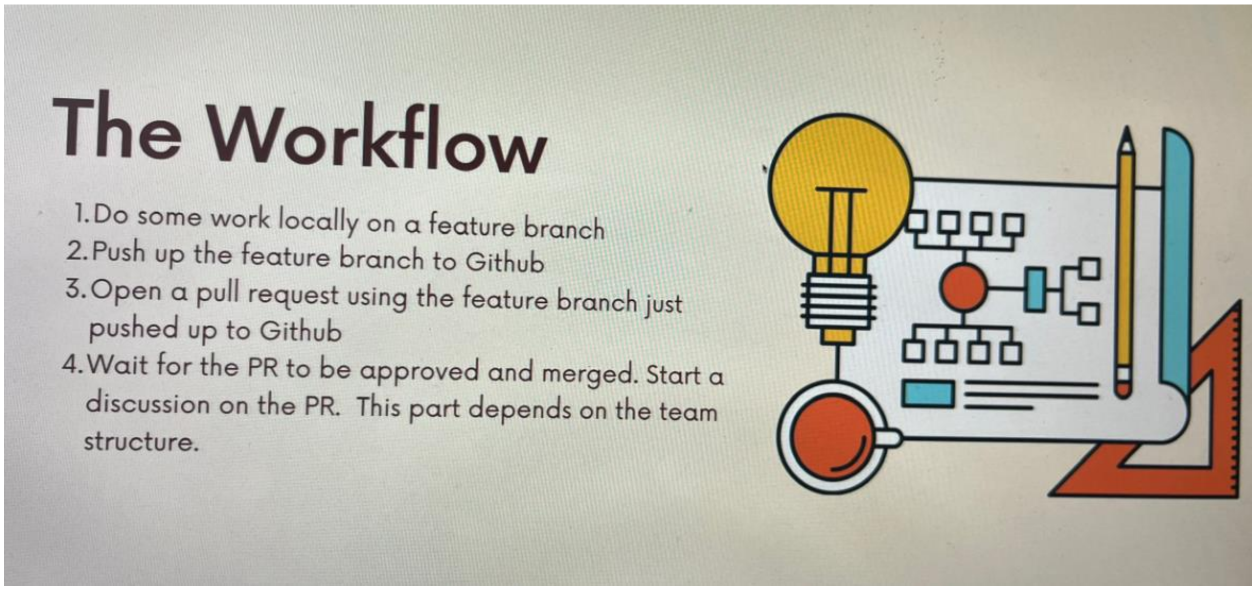


Pull Requests

Pull Requests are a feature built in to products like Github & Bitbucket. **They are not native to Git itself.**

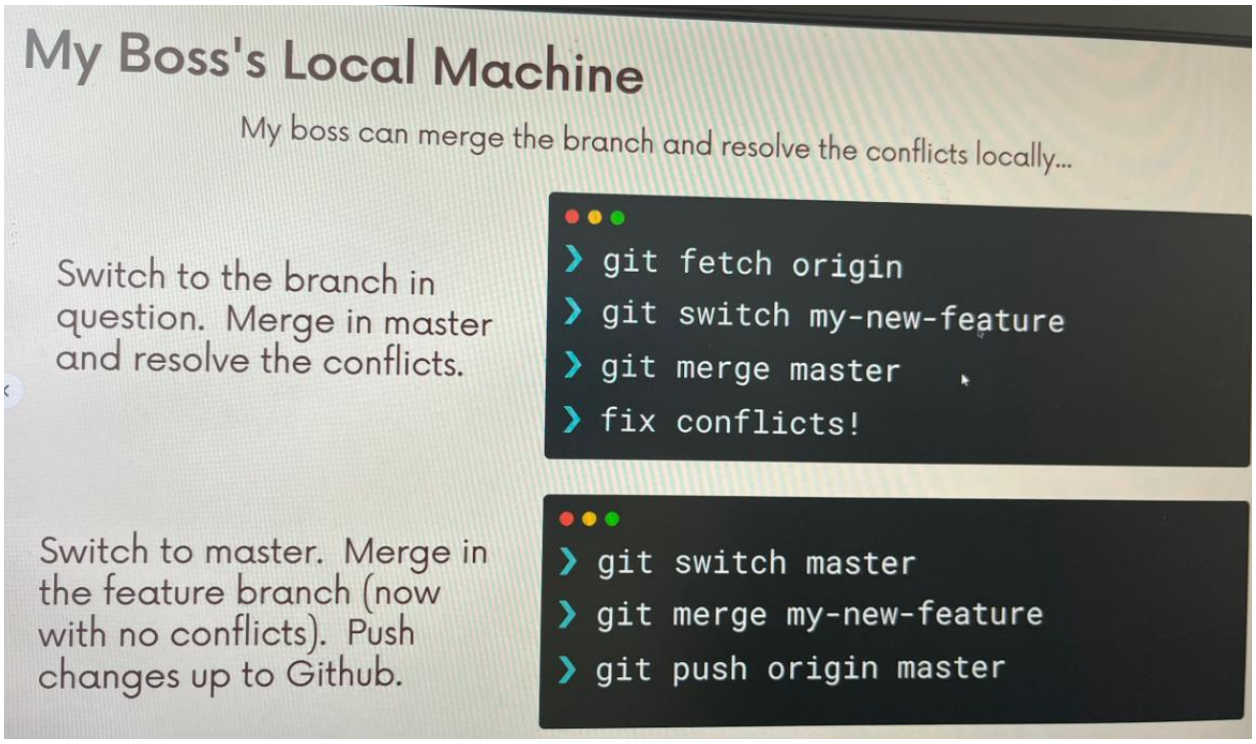
They allow developers to alert team-members to new work that needs to be reviewed. They provide a mechanism to approve or reject the work on a given branch. They also help facilitate discussion and feedback on the specified commits.

"I have this new stuff I want to merge in to the master branch...what do you all think about it?"



The Workflow

1. Do some work locally on a feature branch
2. Push up the feature branch to Github
3. Open a pull request using the feature branch just pushed up to Github
4. Wait for the PR to be approved and merged. Start a discussion on the PR. This part depends on the team structure.



My Boss's Local Machine

My boss can merge the branch and resolve the conflicts locally...

Switch to the branch in question. Merge in master and resolve the conflicts.

```
❯ git fetch origin
❯ git switch my-new-feature
❯ git merge master
❯ fix conflicts!
```

Switch to master. Merge in the feature branch (now with no conflicts). Push changes up to Github.

```
❯ git switch master
❯ git merge my-new-feature
❯ git push origin master
```

Fork & Clone: Another workflow:

The “fork & clone” workflow is different from anything we’ve seen so far. Instead of just one centralized Github repository, every developer has their own Github repository in addition to the “main repo”. Developers make changes and push to their own forks before making pull requests.

It’s very commonly used on large open-source projects where they may be thousands of contributors with only a couple maintainers.

Forking

Github (and similar tools) allow us to create personal copies of other people’s repositories. We call those copies a “fork” of the original.

When we fork a repo, we’re basically asking Github “Make me my own copy of this repo please”

As with pull requests, forking is not Git feature. The ability to fork is implemented by Github.

Now what?

Now that I ‘ve forked, I have my very own copy of the repo where I can do whatever I want!

I can clone my fork and make changes, add features, and break things without fear of disturbing the original repository.

If I do want to share my working, I can make a pull request from my fork to the original repo.