Getting Started with Git

You can imagine that web design is like a building and Git is one of the many important pillars that web design is built on.  
So, our primary objective right now is to:

* Understand what Git is.
* Learn how to use the basic Git commands.
* Discover how to configure Git.
* Understand what Github is and how to work with remote repository.

# What is VCS?

A version control system is a software tool that helps developer teams track and manage source code changes over time.

As you edit and add to your code, you tell the version control system to take a snapshot of your files or save a checkpoint of your progress.  
The version control system saves that snapshot permanently so you can recall it later on if you need it.

Without a VCS, you’re tempted to keep multiple copies of code on your computer. This is dangerous because it’s easy for a file to get corrupted or deleted and you’re susceptible to losing months or even years worth of work.

Version control systems solves that problem by presenting you with all versions of your code with a clear history of the changes made. This allows you to go back to older versions of the code and try a different approach.

# What is Git?

Git is a free and open source distributed VCS designed to handle everything from small to very large projects with speed and efficiency.

In plain English, Git is a tool that allows developers to track versions of their code over time. It does this by creating "snapshots" of the current state of the code base whenever you tell it to.

Git is essential when collaborating with other developers to ensure that there are no code conflicts between them and that previous "snapshots" of the code can be revisited if necessary.

For example, if you are coding and you accidentally break or crash the app, you’ve just lost all your progress and you’re forced to start from scratch. However, it’s easier and safer if you're using Git where you can simply roll back to a previous version of the code.



Installing Git

Git isn’t usually set up by default on your computer, so you need to install and configure it before you can start using it to manage your code.  
It’s important to keep Git up to date, just like all the other software on your machine. Updates protect you from security vulnerabilities, fix bugs, and give you access to new features.

The recommended method of installing and maintaining Git is provided for three major platforms below :

**Windows**  
Download and install Git for [Windows](https://git-scm.com/download/win). Once installed, you’ll be able to use Git from the command prompt or PowerShell. We recommend that you stick with the defaults selected during the installation unless you have a good reason to change them.  
**Linux**  
On the terminal, just run sudo apt install git-all.  
**Mac**  
The best thing to do is to install [Homebrew](https://brew.sh/), and then from the terminal run the command brew install git.

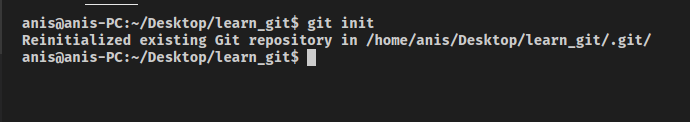
Start using Git

Once we have Git installed, we need to "initialize" a repository before we can start using it.  
It’s very easy to do, just follow my lead:

1. Access the wanted folder using the terminal (prompt cmd).
2. To initialize a repository, we only have to run:

git init

The output would be :



***P.S****: This command just created a hidden folder called .git, where all the magic happens.*

# Adding and committing Files

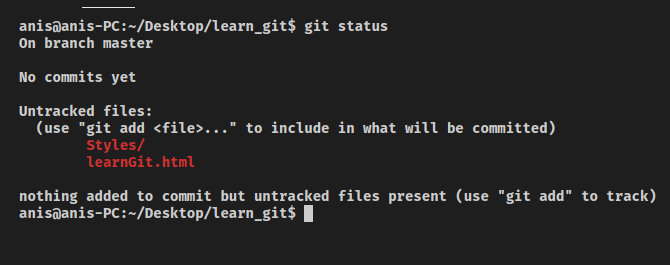
Remember that folder we’ve created? It’s high time we used it. We’ll be creating a file and placing it in there.  
Feel free to add whatever you want!

## **Check the Status of Your Repository**

Now that we have a few files in our repository, let's see how Git processes them.  
To check the current status of your repository, we use the git status command.

git status

The output would be :

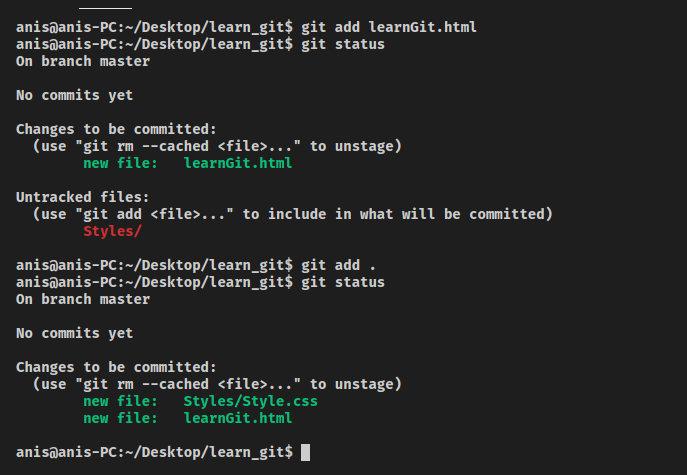


## **Adding Files for Git to Track**

At this point, we do not have any files for Git to track.  
We need to add files specifically to Git in order to tell Git to track them. We add files using the “add” command.  
After running git add . Git will add all the repository files to to an intermediate area called the **staging area**. We can also add what we want simply by running git add myFileName

git add .

The output would be :

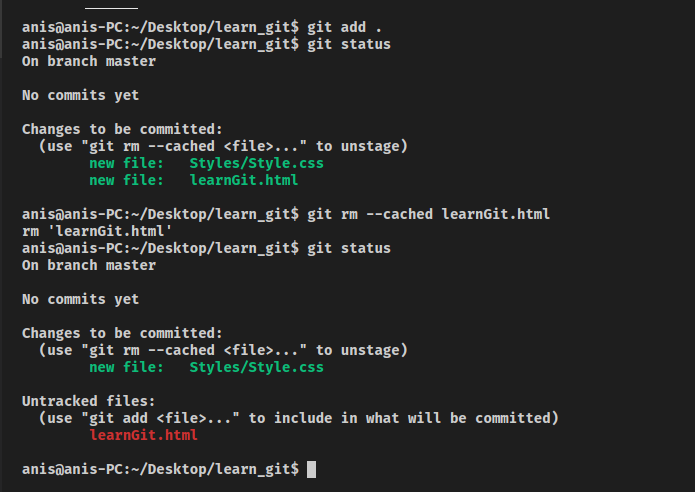


## **Removing Files**

Let’s say that you have added files to Git and you do not want it to track any of them.  
In a situation like this, you tell Git to stop tracking them.  
However, running a simple git rm command will not only remove it from Git, but will also remove it from your local file system as well! To tell Git to stop tracking a file while still keep it in your local system, run the following command:

git rm --cached [file\_name]

The output would be :

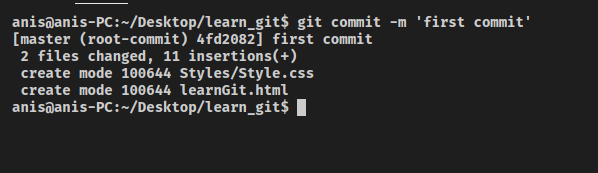


## **Committing Changes**

Once you have staged your files, you can commit them into Git.  
Imagine a commit command as a snapshot in a certain point and time where you can return back to access your repository at that stage. You assign a commit message to every commit, which you can pass with the -m prefix.

git commit -m 'first commit'

The output would be :



# Files status

Git sees every file in your working copy as one of three things:

**Tracked**  
A file which has been previously staged or committed.  
**Untracked**  
A file which has not been staged or committed.  
**Ignored**  
A file which Git has been explicitly told to ignore.

Ignored files are usually built artifacts and machine generated files that can be derived from your repository source or should otherwise not be committed.

**PS**: If we have a file that we do not want to track, we just put it inside .gitignore file.

# Configuration settings

The .gitconfig file contains a list of configurations that affects the behaviour of git commands, so in order to manipulate it, we use the command git config <configuration>. Therefore, to change the name and email used by Git to identify the user, we run the following command (PS. change YOUR.NAME and YOUR.EMAIL with your own values.)

git config --global user.name "YOUR NAME"

git config --global user.email "YOUR EMAIL"

# Git aliases

Oftentimes, you’ll be finding yourself typing git commands over and over again. For example, git add, git init and git status are commands you will be repeatedly using.  
The smart thing thing to do then is to use a shortcut and that’s an alias. Aliases help make your Git experience simpler, easier and more familiar.  
To create a temporary alias, which will last as long as your terminal session is open, you can type:

git config alias.KEYBOARD\_SHORTCUT COMMAND

So if we wanted to type git st and have it function like git status, we would type git config alias.st status. Now we can type git st and have the same output as if we typed git status.

If you would like your alias to be a part of your global configuration, add the --global command after git config. For example, to alias git i to git init globally, you would type git config --global alias.i init.

# What is GitHub?

GitHub is a web-based Git repository hosting service.  
Simply put, it is a tool that enables collaboration by hosting shared Git repositories that teams of developers can all contribute to.  
While GitHub uses Git, the functionality it provides is **entirely different** from Git. So try make it stick in your mind that Git and GitHub are not the same thing.

In short, Git is a Version Control System while GitHub is an online platform for hosting and sharing code, text files and more complex file formats.



Why use GitHub?

GitHub provides a great way for you to store your code in a safe and remote location (in case something happens to your local machine). It's a fantastic way to collaborate with other developers both privately and publicly.

Many large open source projects are hosted on GitHub, which makes it very easy to examine the code both on GitHub and on local machines.

In the next couple of chapters, we will learn how to move code from our local repository to a remote repository on GitHub using the push command, as well as retrieve code from a remote repository on GitHub using the pull command.

We'll also learn about GitHub specific concepts like forking and pull requests.

Want to get a sense for how popular GitHub is in the development community? Here are some projects you may have heard of that are hosted there:

* [Angular](https://github.com/angular/angular)
* [React](https://github.com/facebook/react)
* [Ruby on Rails](https://github.com/rails/rails)
* [Twitter Bootstrap](https://github.com/twbs/bootstrap)
* [Node.js](https://github.com/nodejs/node)
* [JQuery](https://github.com/jquery/jquery)
* [Homebrew](https://github.com/Homebrew/brew)

# Getting started with GitHub

If you don't have an account on GitHub yet, head to [github.com](https://github.com/) and create an account. Signing up is easy!

Be sure to use whatever email address is in your .gitconfig when you sign up on GitHub.

If you'd rather sign up with a different email address, change your .gitconfig accordingly. You'll run into some minor annoyances if there's a mismatch between the email address in your GitHub profile and the email address in your .gitconfig.

# Creating a remote repository

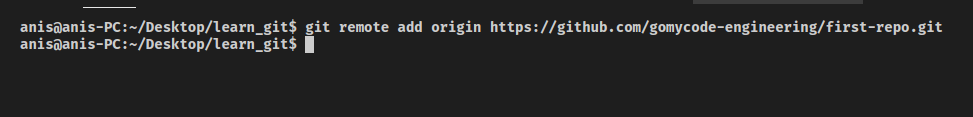
Once you have a GitHub account, head over to https://github.com/new and create a repository. In this example, we will be creating a repository called first\_repo. Do not worry about the description or checking the box to initialize the repository with **README**.

After you've created the repository, GitHub will give you a few instructions to get started. The instructions should look something like this (the url towards the end will depend on your GitHub username):

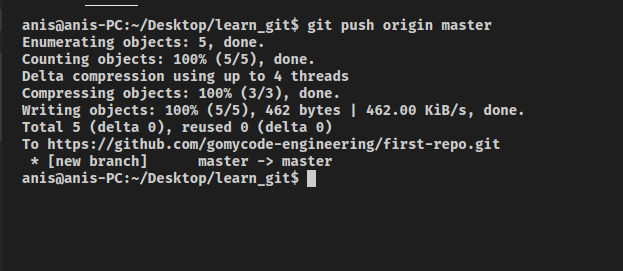


# Pushing code to GitHub

Now after creating a repository in GitHub, we need to add our local code (after commits) to the remote repo. It’s very simple. First, we need to attach the address of the GitHub repo to our local project, simply by running git remote add origin YOUR\_REPO\_URL.



Now, after providing the remote URL to the local repository to take the local files to the remote host, we have to run this command git push origin master.



And voila! Our files are now in GitHub.



# ork

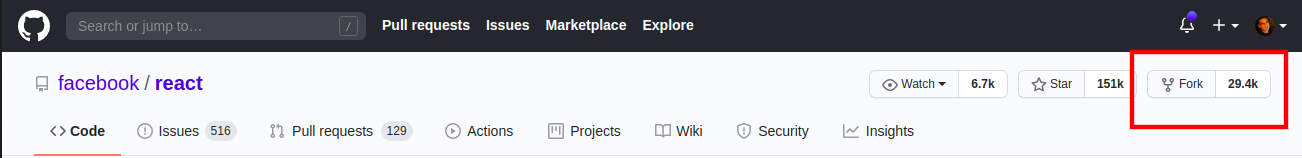
Now that we know how to push code to GitHub, let's explore one of GitHub’s important features: forking.

When collaborating with others, you’re unable to push directly to the original repository (Imagine if you could play with one one of the world’s largest open source projects and repositories… that would be insane).

So the logical thing to do is make a copy of someone else's remote repository and make sure it is under our username so that we can push code to it. This ensures that our experimenting with the repository doesn’t affect the original version.

To practice forking, head over to any repo on [github](https://github.com/) and on the top right you will see a button with the text Fork. Click on this button and you will have a copy of the repository under your name!

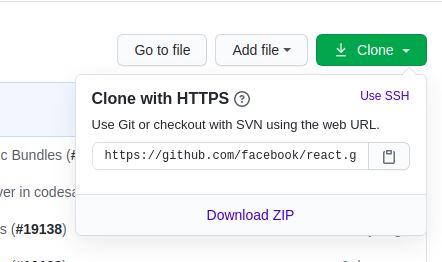
Remember, "forking" is strictly a **GitHub** feature and it is not by any means related to Git. It is simply a way to make your own copy of a repository on your account where you have permission to push your code to GitHub.



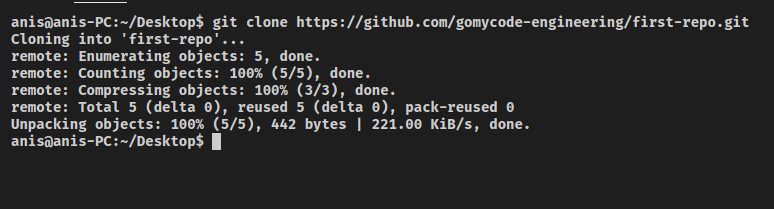
# Clone

Once you have applied “forking” on the repository, you need to select it (the remote one you just made) and download the code on your local computer (i.e. make a local repository). Instead of making a folder and going through the whole git init process and adding a remote, you can conveniently use the git clone command, which accepts a link to the repository and downloads it into a folder (with everything already set up!).

In order to do that, first click on the button clone and that will provide us with a remote URL. Finally, just copy the URL.  
In the terminal, just run git clone THE\_COPYED\_ADDRESS.



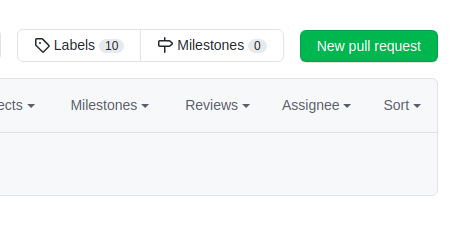
git clone https://github.com/facebook/react.git



# Pull Request

Now let's say you are collaborating with an organization on GitHub (where you forked the repository from) and you would like to merge your changes with the original repo that you forked (remember you can't just push to it, because you do not have permission to do so). You can issue a pull request and the one who can grant permission can either merge or reject it.

To do this, click on the "New pull request button" and then click on the "Create pull request". You should then be able to go to the original repository and see your pull request or "PR".



# Kudos to you for making it this far!

We can fairly say we have mastered Git & GitHub!

But, there’s always more to learn.

We can now work with remote repositories and keep track of the differents versions of our code. We can also work and collaborate with teams in order to build large projects.