



Git & Github





Git

Git is a free and open source distributed version control system. Git's purpose is to keep track of projects and files as they change over time with manipulations happening from different users. Git stores information about the project's progress on a repository. A repository has commits to the project or a set of references to the commits called heads. All this information is stored in the same folder as the project in a sub-folder called `.git` and will mostly be hidden by default in most systems.



git



So basically. Git keeps track of the changes a couple of people make on a single project and then merges the code where people have worked on different parts into one project. This way, when someone introduces a bug, you can track down the code that introduced the bug by going through the commits. Commits must be made to a project to tell git that you're satisfied with the changes you've made and want to commit the changes into the main branch called master by default.

You can then upload the code to GitHub or BitBucket where authorised users can either view, pull the code or push changes.



GitHub

Humans have always thought of ways to connect and to work together as a team, over the years we have created new ways to connect virtually allowing us to work together without having to worry about the distance. Applications like Facebook and Twitter are now indispensable for us. Almost all video games allow you to play online with friends or strangers from all over the world. GitHub is another one of these pieces of software that allow people to connect and work together in a more efficient, fast, and dynamic way. In this blog, I will explain how GitHub works and the different things that we can use it for.



1. Initialize Git

To create a new repo, you'll use the `git init` command. `git init` is a one-time command you use during the initial setup of a new repo. Executing this command will create a new `.git` subdirectory in your current working directory.

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top left corner. The text `git init` is displayed in a light blue monospaced font.

```
git init
```

Once, the repository is initialized git tracks the changes in the files and folders of the project.



2. Add file to the staging area

The `git add` command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, `git add` doesn't really affect the repository in any significant way—changes are not actually recorded until you run `git commit`.

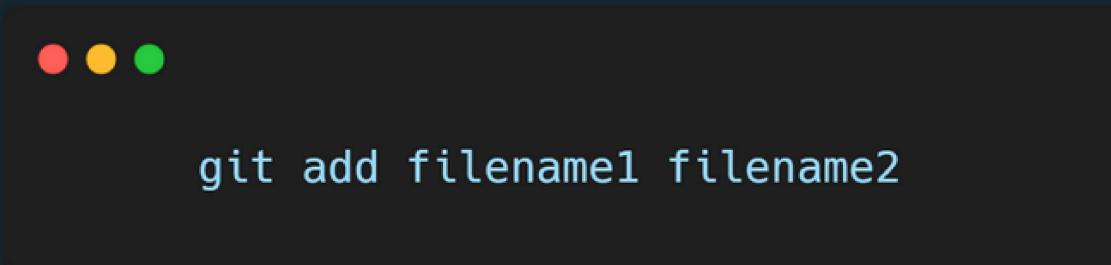
To add a single file, we use the `git add` command followed by a file name



```
git add filename
```



To add multiple files using their file names using the command `git add` followed by file names



```
git add filename1 filename2
```

Sometimes, we may make a lot of changes, and adding files one by one is tiring and not product. Therefore, we can use a short and product way. The `git add` command followed by a dot allows adding files and folders at once to the stage area. Remember, there is a space between the `add` and the dot.

To add all files and folders at once



```
git add .
```

3. Unstage a file

The easiest way to unstage files on Git is to use the “git reset” command and specify the file you want to unstage. By default, the commit parameter is optional : if you don’t specify it, it will be referring to HEAD.

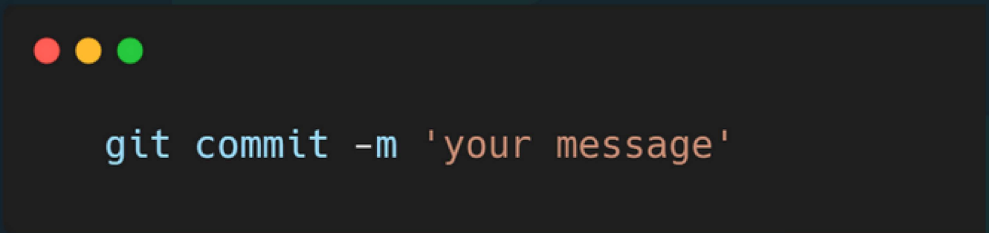


```
git reset HEAD filename
```




4. Commit the changes

The git commit command is one of the core primary functions of Git. Prior use of the git add command is required to select the changes that will be staged for the next commit. Then git commit is used to create a snapshot of the staged changes along a timeline of a Git projects history.

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. The text 'git commit -m 'your message'' is displayed in a light blue monospace font.

```
git commit -m 'your message'
```

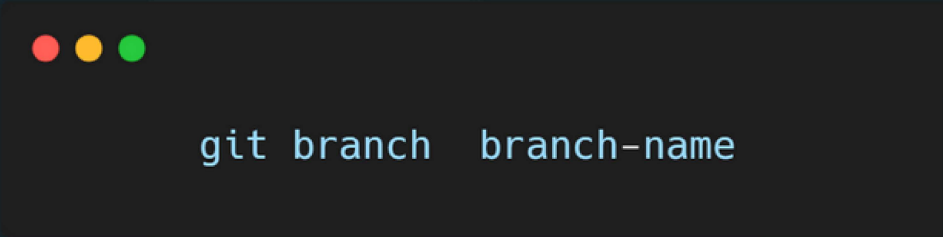
Your commit message has to be associated with the changes or modifications you make.



5. Creating a branch

The git branch command can be used to create a new branch. When you want to start a new feature, you create a new branch off the main using git branch new_branch. Once created you can then use git checkout new_branch to switch to that branch.

Only to create a branch



```
git branch branch-name
```

To create and checkout to the branch at the same time:



```
git checkout -b branch-name
```

To switch between branches:



```
git checkout main  
git checkout branch-name
```

To list down all the branches:



```
git branch
```



6. Create account on GitHub

Go to <https://github.com/join> .

- Type a user name, your email address, and a password.
- Choose Sign up for GitHub, and then follow the instructions.

7. Create Repository on GitHub

To put your project up on GitHub, you will need to create a repository for it. Create a repository

8. Connecting git with remote repository

In this step, you will connect your local git repository with your remote GitHub repository



```
git remote add origin remote_repository_url
```

The word origin could be any word. It is a means to assign the repository URL. If this step is passed without error, you are ready to push it to your remote GitHub repository. Push means actually uploading what you have on your local to remote repository. How to connect to a remote Git repository?

9. Push

git push updates the remote branch with local commits. It is one of the four commands in Git that prompts interaction with the remote repository.



You can also think of git push as update or publish. Before you push(upload), please commits any changes and if it is ready push your files to your remote GitHub repository using the following command.

```
git push -u origin master
```

10. Pull

Git pull updates your current local working branch, and all of the remote tracking branches. It's a good idea to run git pull regularly on the branches you are working on locally.



Without git pull, (or the effect of it,) your local branch wouldn't have any of the updates that are present on the remote.

```
git pull 'remote_name' 'branch_name'.
```

11. Git clone

git clone is primarily used to point to an existing repo and make a clone or copy of that repo at in a new directory, at another location. The original repository can be located on the local filesystem or on remote machine accessible supported protocols. The git clone command copies an existing Git repository.



```
Shreya@DESKTOP-VSQ7BAB MINGW64 ~ (master)
```

```
$ cd Desktop/
```

```
Shreya@DESKTOP-VSQ7BAB MINGW64 ~/Desktop (master)
```

```
$ git clone https://github.com/geeksterin/WebCurriculum
```

12. The .gitignore file

The .gitignore file is a text file that tells Git which files or folders to ignore in a project. A local .gitignore file is usually placed in the root directory of a project. You can also create a global .gitignore file and any entries in that file will be ignored in all of your Git repositories. The .ignore file



Here's an example of a .gitignore file that you can use in a Git repository to specify files and directories that should be ignored:

```
● ● ●  
# Ignore compiled object files  
*.o  
*.ko  
*.obj  
*.elf
```

Git Status

The git status command is used to show the status of the git repository. This command displays the state of the local directory and the staging area.



```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ touch demofile

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/NewDirectory (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        demofile

nothing added to commit but untracked files present (use "git add" to track)
```

git diff

git diff lists out the changes between your current working directory and your staging area.

```
$ git diff
diff --git a/dog.txt b/dog.txt
index 85a6632..2a90585 100644
--- a/dog.txt
+++ b/dog.txt
@@ -1,1 @@
-my name is puppy
\ No newline at end of file
+my name is pup.
\ No newline at end of file
```



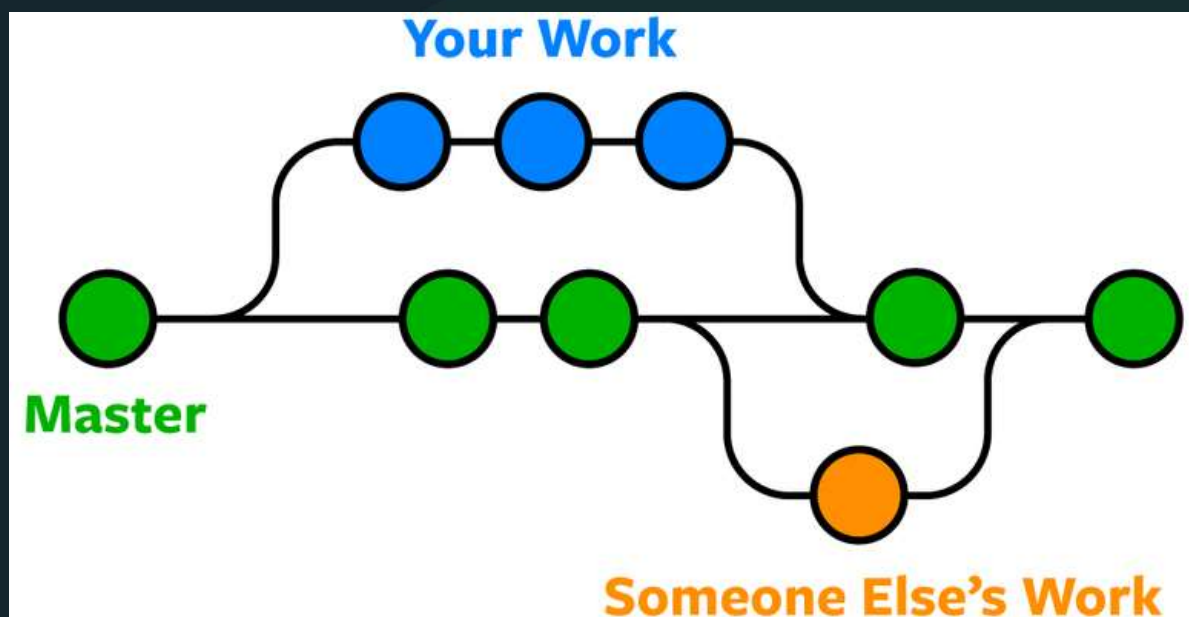
Git Branch

Imagine you are working on a group project with your friends, and you all have different tasks to complete. Now, let's say you're working on a particular task and you want to make sure your work doesn't interfere with what others are doing.





In this scenario, a "git branch" is like having a separate copy of the project where you can work independently on your task without affecting the main project or what others are doing. It's like having your own sandbox to experiment and make changes without worrying about breaking anything.





So, when you create a git branch, you're essentially creating a parallel version of the project that allows you to make changes and try out new things without affecting the main project. Once you're done with your task and are happy with the changes you made, you can merge your branch back into the main project so that everyone can benefit from your work.

Operations on Branches

We can perform various operations on Git branches. The git branch command allows you to create, list, rename and delete branches. Many operations on branches are applied by git checkout and git merge command. So, the git branch is tightly integrated with the git checkout and git merge commands.



Create Branch

You can create a new branch with the help of the git branch command. This command will be used as:



```
$ git branch <branch name>
```

List Branch

You can List all of the available branches in your repository by using the following command.

Either we can use git branch – list or git branch command to list the available branches in the repository.



```
$ git branch --list  
or  
$ git branch
```

Here, both commands are listing the available branches in the repository.

Delete Branch

You can delete the specified branch. It is a safe operation. In this command, Git prevents you from deleting the branch if it has unmerged changes. Below is the command to do this.



```
$ git branch -d<branch name>
```



Delete a Remote Branch

You can delete a remote branch from Git desktop application. Below command is used to delete a remote branch:

```
$ git push origin -delete <branch name>
```

Switch Branch

Git allows you to switch between the branches without making a commit. You can switch between two branches with the git checkout command. To switch between the branches, below command is used:



```
$ git checkout<branch name>
```

Rename Branch

We can rename the branch with the help of the git branch command. To rename a branch, use the below command:



```
$ git branch -m <old branch name><new branch name>
```

Merge Branch



Git allows you to merge the other branch with the currently active branch. You can merge two branches with the help of git merge command. Below command is used to merge the branches:



```
$ git merge <branch name>
```