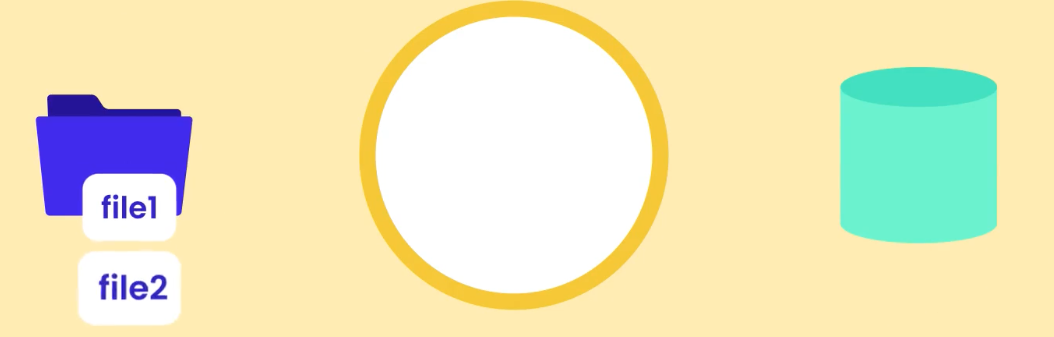
First time we want to add files to a git repository, we have to initialize the repository. The following command will initialize an empty repository in current path in the subdirectory .**git.** By default, this directory is hidden.

* git init

**Git Workflow:**

1. First, we make some changes in our files



1. we take a snap-shot of our project and send them to staging area.

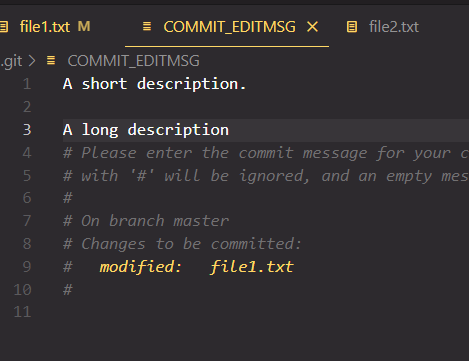


1. Here we review our changes and if everything is good, we commit to the git to permanently store the snapshot.



1. One important note, is that once we commit the staging area doesn’t become empty. So, in case we make some changes in the file1 for example, we have to use **git add file1.** And in case we delete let’s say file2, we have to use **git add file2** to make the changes as well in the staging area. *In short, once we add a file to staging area and if we make changes to file after that, we have to add them to stage area again*.

* **Staging area:** To add files to stage area, we use command **git add.** We can use multiple files separated by a comma, or pattern like **\*.txt** (all text files) or **.** (Adds entire directory recursively, careful using this one).
  + We can check current files in the staging area by using command **git ls-files**
  + If files are red in untracked file area, it is not in staging area, just in the working directory. But if its green it’s in the staging area.
* **Commit:** Once we add the files to staging area, and review the files we can commit them to git. Here we use command **git commit -m “commit description”.** 
  + **Note:** In case the short description message isn’t enough and we need long descriptions we just use **git commit** and it will open our editor. Here on first line, we can use the short description after that with a line break, we can use the long description. After that we save the changes and close the window.



* + **Skipping the staging area:** In case we don’t want to go through staging area, we can directly commit all files using command **git commit -a -m “short commit message”.**
* **Deleting files:** After adding a file to staging area, if we delete the file, it will still be present in the staging area. To make the changes we can use **git add filename.txt** and then **git commit -m “…”** But as this is a very common situation, git gives us a shortcut **git rm filename.txt.** This command removes the files from both working directory and staging area.
* **Renaming/Moving files or Directory:** To make this change we can use the command **git mv source.js destination.js.** Here the first file is the original name and second one is the changed name. Note: the file has to be in the staging area for this to work.
* **Ignoring files:** We can add the list if files in **.gitignore** file (which must be in root directory).
  + One issue is that if we already add a file in git, and then add the file in the .gitignore file, it wouldn’t work. To solve the issue, we can use **git rm –cached -r bin/** Here -r is for adding files recursively.
  + We can find base template for .gitignore files for different languages in the GitHub repo **GitHub/gitignore**
* **Viewing changes:** the command to see the staging area is **git diff –staged.** This is not used though using gitbash, we will use diff tool. There are different tools like **Kdiff3**, **P4Merge**, **WinMerge**. We can use VSCode as well for this.
  + Modern IDE cover this feature and this command isn’t very useful to use
* **History:** Some commands are,
  + **git log**
  + **git log –oneline** This shows short summary of history.
  + **git log –oneline –reverse** shows the history on reverse.
* **View Commit:** we can use command **git show <logNameOfCommit (**we can also use HEAD or HEAD~1**)**
* **Un-staging files:** older versions used the command **reset,** but in new version **restore** command is used for un-staging files. With this we can easily restore files from both working directory and staging area. **git restore**