Overview:

This is a walkthrough of basic Git and GitHub concepts for beginners.

Table of contents:

What is Version Control System (VCS)?

Version Control Systems are the software tools for tracking/managing all the changes made to the source code during the project development. It keeps a record of every single change made to the code. It also allows us to turn back to the previous version of the code if any mistake is made in the current version. Without a VCS in place, it would not be possible to monitor the development of the project.

Types of VCS

The three types of VCS are:

1. Local Version Control System
2. Centralized Version Control System
3. Distributed Version Control System

Local Version Control System

Local Version Control System is located in your local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost. If anything happens to a single version, all the versions made after that will be lost.

Also, with the Local Version Control System, it is not possible to collaborate with other collaborators.

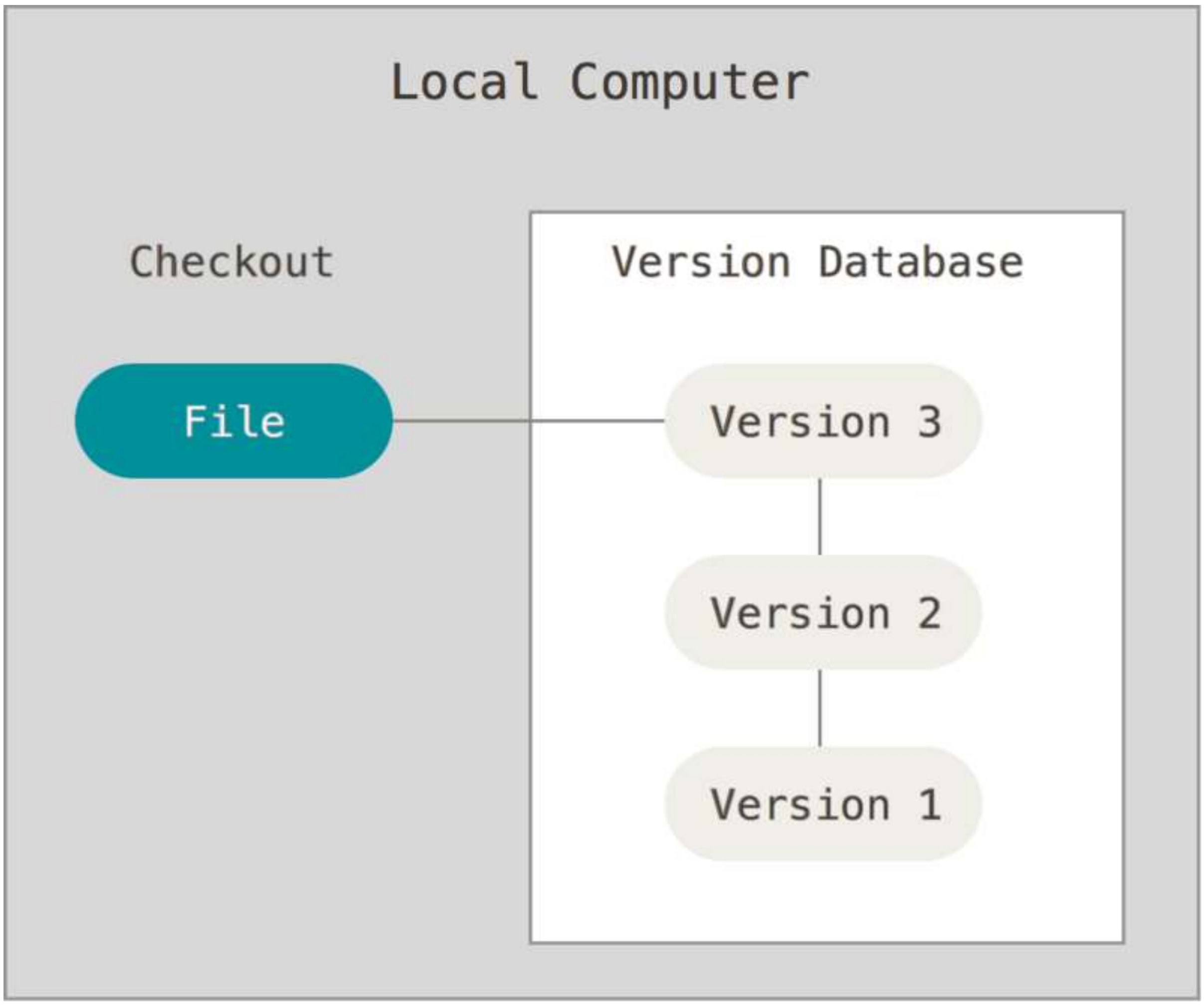


Image 2

To collaborate with other developers on other systems, Centralized Version Control Systems are developed.

Centralized Version Control System

In the Centralized Version Control Systems, there will be a single central server that contains all the files related to the project, and many collaborators checkout files from this single server (you will only have a working copy). The problem with the Centralized Version Control Systems is if the central server crashes, almost everything related to the project will be lost.

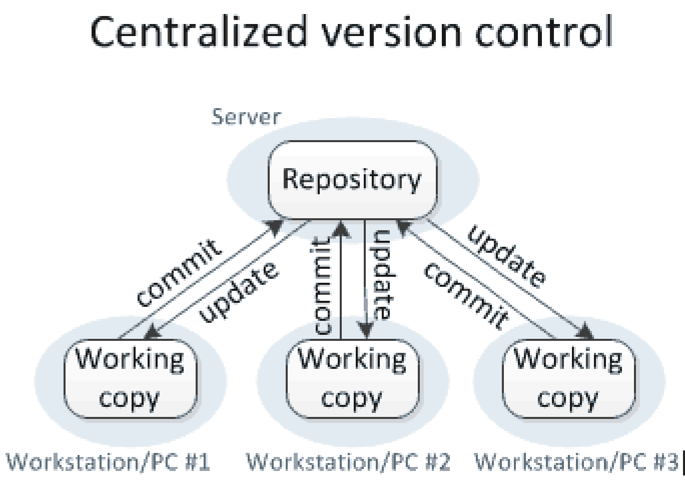


Image 3

To overcome all the above problems, Distributed Version Control Systems are developed.

Distributed Version Control System

In a distributed version control system, there will be one or more servers and many collaborators similar to the centralized system. But the difference is, not only do they check out the latest version, but each collaborator will have an exact copy (mirroring) of the main repository(including its entire history) on their local machines.

Each user has their own repository and a working copy. This is very useful because even if the server crashes we would not lose everything as several copies are residing in several other computers.

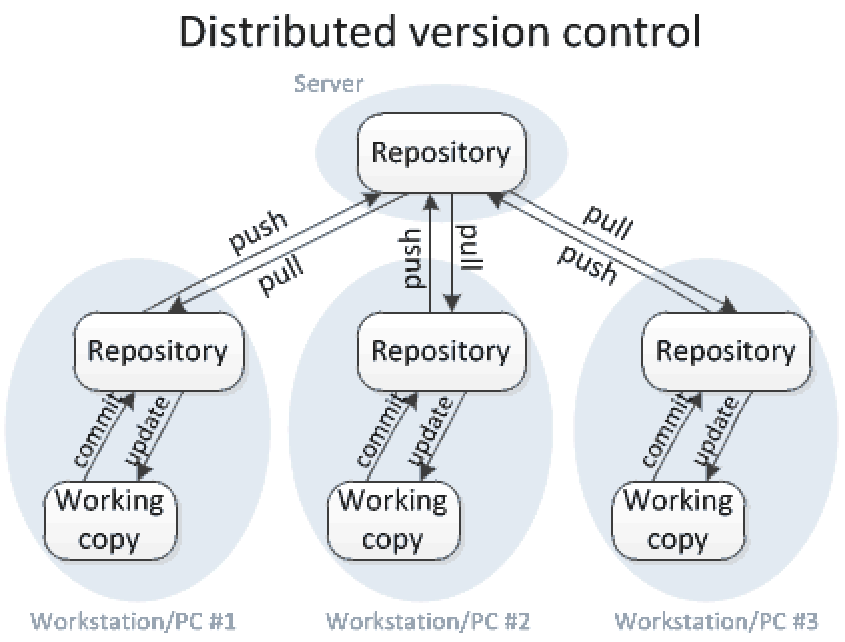


Image 4

Difference between Git and GitHub

Git is a version control tool (software) to track the changes in the source code.

GitHub is a web-based cloud service to host your source code (Git repositories). It is a centralized system.

Git doesn't require GitHub but GitHub requires Git.

Installation of Git

There are two ways of installing Git.

1. **Install Git for using WSL** (Windows Subsystem for Linux)

Install Ubuntu – https://www.youtube.com/watch?v=X-DHaQLrBi8

Install Git in Ubuntu – https://www.digitalocean.com/community/tutorials/how-to-install-git-on-ubuntu-20-04

2. **Install Git software for windows –** https://git-scm.com/download/win

Go ahead with whichever method is comfortable for you. Continue further once you are done with the installation.

5 Git operations and commands

Before deep-diving into Git operations and commands, create an account for yourself on GitHub if you don't have it already.

Create repositories

Create a remote central repository on GitHub.

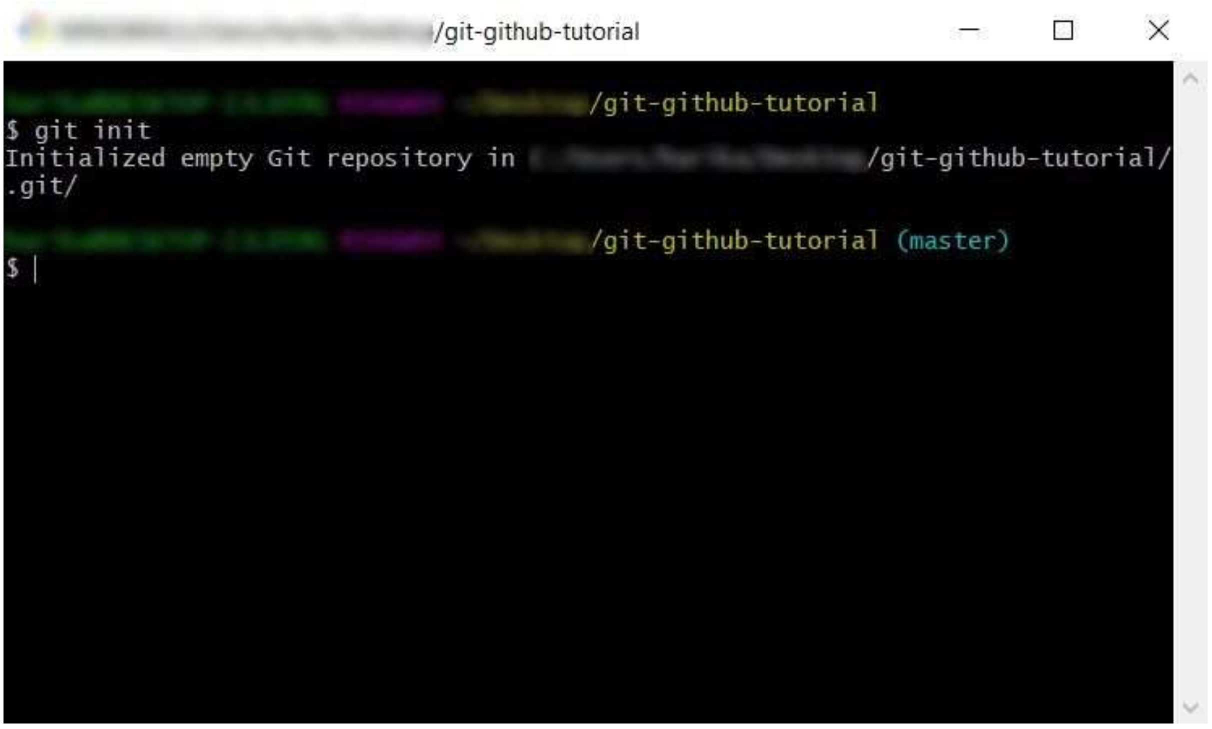
https://docs.github.com/en/get-started/quickstart/create-a-repo

create a local repository using git (I am using Git software on Windows 10)

Open your file explorer, navigate to the working directory, right-click and select "Git Bash Here". This opens the Git terminal. To create a new local repository use the command **git init** and it creates a folder **.git**.

git init to create a new Git repository

```
$ git init
```



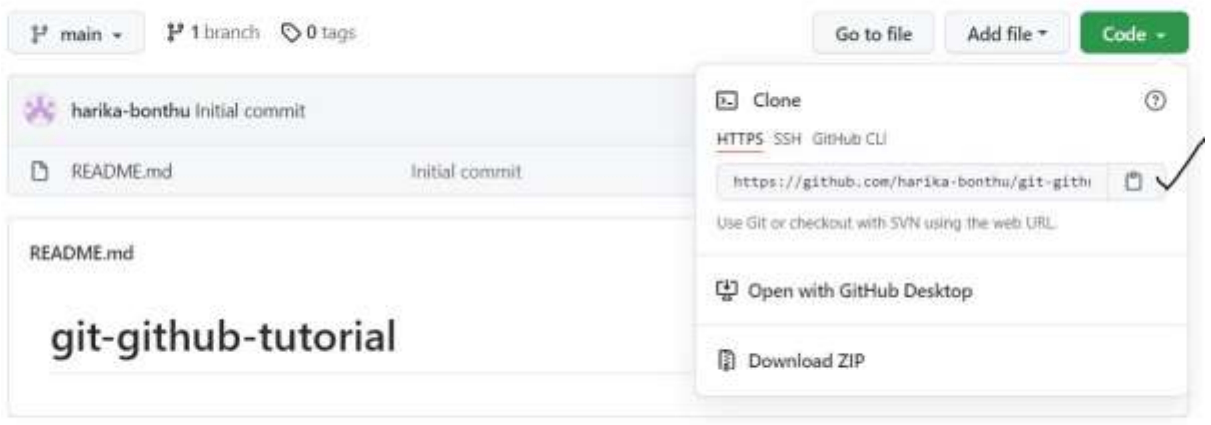
```
[redacted]/git-github-tutorial

[redacted]/git-github-tutorial
$ git init
Initialized empty Git repository in [redacted]/git-github-tutorial/
.git/

[redacted]/git-github-tutorial (master)
$ |
```

(master) is the default branch of the local repository.

Next, we need to sync the local and the central repositories.

git remote add to add a new remote repository.

To get the URL of the central repo, open your repository in GitHub and copy the link.



```
https://github.com/harika-bonthu/git-githu
```

Use Git or checkout with SVN using the web URL.

Run the below command,

```
$ git remote add origin "https://github.com/harika-bonthu/git-github-tutorial.git"
```

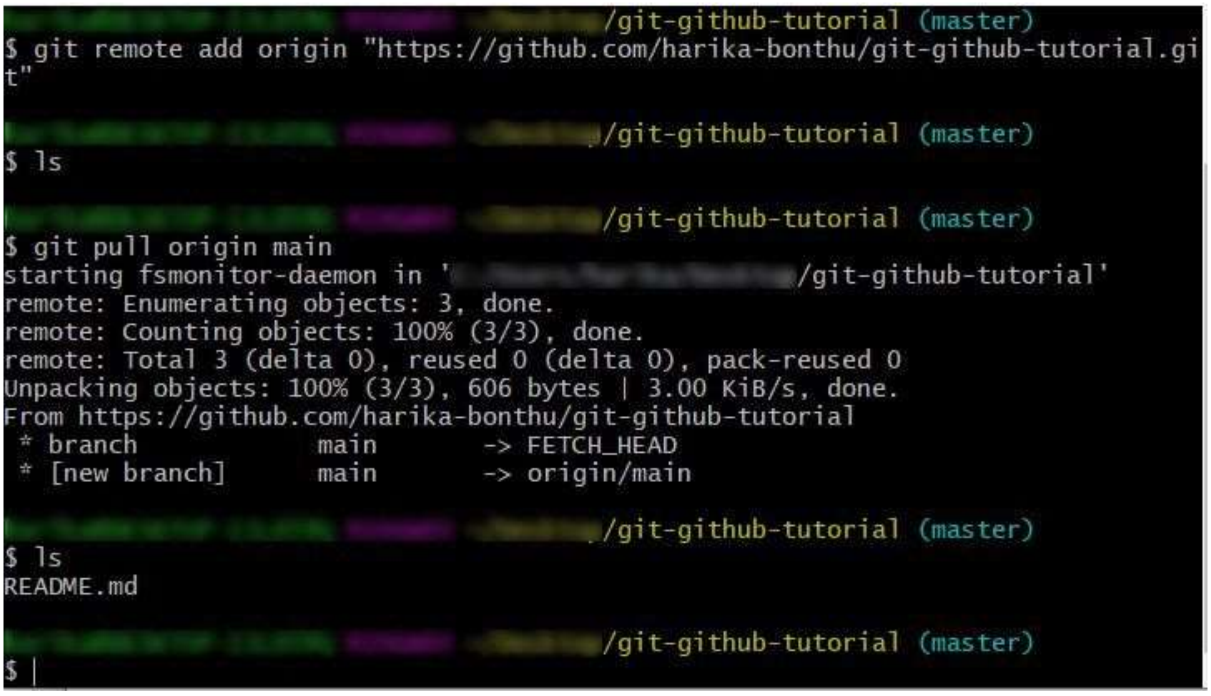
Generally, Origin is the shorthand name of the remote repository that we are cloning.

After adding, we need to pull the files from the remote repo.

git pull to download all the content from the remote repo

```
$ git pull origin main
```

(main is the branch in our central/remote repository. Kindly check the branch name before pull request)



```
[redacted]/git-github-tutorial (master)
$ git remote add origin "https://github.com/harika-bonthu/git-github-tutorial.gi
t"

[redacted]/git-github-tutorial (master)
$ ls

[redacted]/git-github-tutorial (master)
$ git pull origin main
starting fsmonitor-daemon in '[redacted]/git-github-tutorial'
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 606 bytes | 3.00 KiB/s, done.
From https://github.com/harika-bonthu/git-github-tutorial
 * branch            main       -> FETCH_HEAD
 * [new branch]      main       -> origin/main

[redacted]/git-github-tutorial (master)
$ ls
README.md

[redacted]/git-github-tutorial (master)
$ |
```

With just adding the origin, we do not have any files. After pulling from the main branch, we now have a README.md file in the local repository.

Now, if you again try to pull, it says “Already up to date.”

```
$ git pull origin main
```

```
From https://github.com/harika-bonthu/git-github-tutorial
 * branch            main       -> FETCH_HEAD
Already up to date.
```

Next, if you want to check if any files are modified or to be committed, use the below command.

git status to check the status of the working directory and the staging area.

Working directory – It is the place where we make changes to the existing files or create new files.

Staging area – It is the place where the files are ready to be committed.

```
$ git status
On branch master
nothing to commit, working tree clean
```

Since the last pull, we haven’t made any changes in the working directory. So it says “nothing to commit, working tree clean)

Now the question is, how do we add files to the staging area.

git add to add files to the index or the staging area.

To demonstrate it with an example, I am modifying the README.md file and creating two more text files “file1.txt”, “file2.txt”

If you wish to use the command line for creating or modifying files, please refer to the video: https://www.youtube.com/watch?v=UeF4ZhnPzZQ

After making changes in the working directory, once again check the status using the command **git status.**

$ git status
On branch master

Changes not staged for commit:

(use “git add …” to update what will be committed)

(use “git restore …” to discard changes in working directory)

modified: README.md

Untracked files:

(use “git add …” to include in what will be committed)

file1.txt

file2.txt

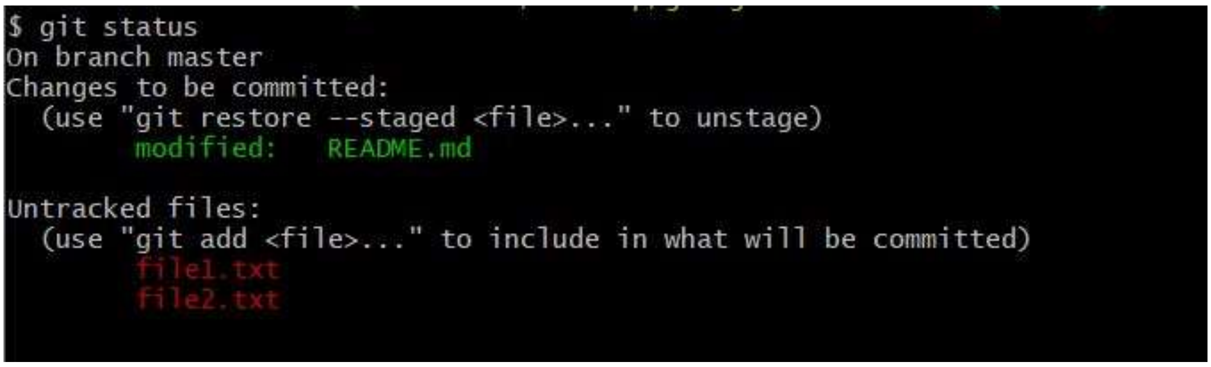
no changes added to commit (use “git add” and/or “git commit -a”)

It shows that the files **file1.txt**, **file2.txt** are untracked and **README.md** is modified.

Next, we will see how to add the README.md file to the staging area.

```
$ git add README.md
```

Below is the status after adding it to the staging area.

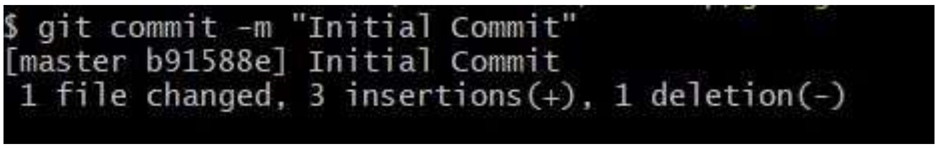


```
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt
```

The next step is to commit these changes to the local repository.

git commit to save the changes to the local repository.

```
$ git commit -m "Initial commit"
```



```
$ git commit -m "Initial Commit"
[master b91588e] Initial Commit
 1 file changed, 3 insertions(+), 1 deletion(-)
```

-m in the above command stands for the message. The message lets other developers know what changes have been made.

Don’t forget we still have two files in the working directory that are to be committed.

Now, I am going to modify file1.txt, file2.txt files using the “nano” command.

To add multiple files to the staging area, we can simply use **-A** flag in the git add command.

```
$ git add -A
```

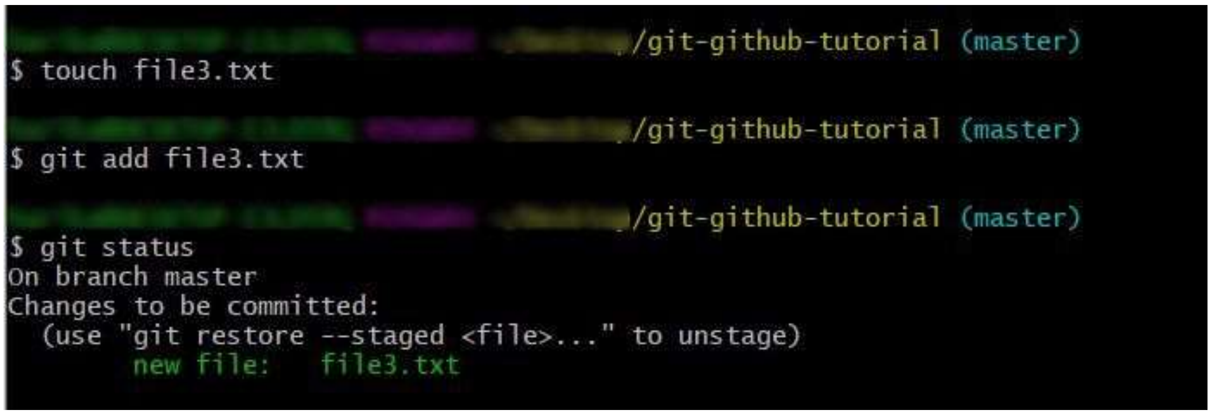
Then check the status and commit them.

```
$ git status
$  git commit -m "Committed txt files"
```

Now, what if you want to undo staging? Let’s see how it is done.

For that, I am creating another file named “file3.txt” and add it to the staging area and check the status.

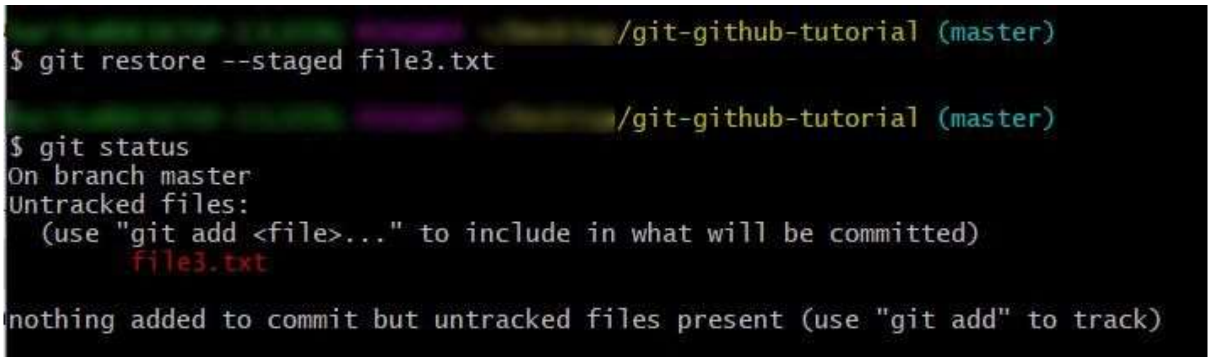
```
$ touch file3.txt
$ git add file3.txt
$ git status
```



```
[redacted]/git-github-tutorial (master)
$ touch file3.txt

[redacted]/git-github-tutorial (master)
$ git add file3.txt

[redacted]/git-github-tutorial (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file3.txt
```

To undo it, use the below command.

```
git restore --staged file3.txt
```



```
[redacted]/git-github-tutorial (master)
$ git restore --staged file3.txt

[redacted]/git-github-tutorial (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file3.txt

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

To see all the commits that are made till now, check the log.

git log to see all the commits

```
$ git log
```

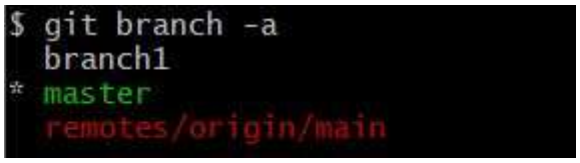
Once you get familiar with the concepts that are discussed now, we will move to the topic **branches**.

A branch in Git is an independent line of work (a pointer to a specific commit). It allows users to create a branch from the original code (master branch) and isolate their work.

git branch to create a new branch

$ git branch branch1

To see all the branches used **git branch -a**

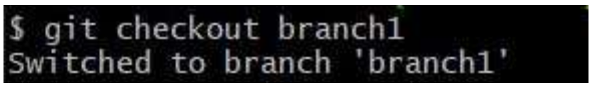


```
$ git branch -a
  branch1
* master
  remotes/origin/main
```

master is highlighted as we are currently working in the master branch. To switch to another branch we need to checkout.

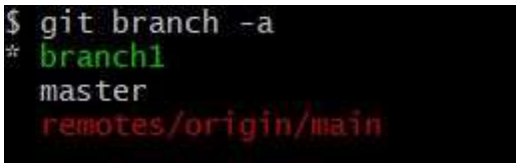
git checkout to switch to another branch

$ git checkout branch1



```
$ git checkout branch1
Switched to branch 'branch1'
```

$ git branch -a



```
$ git branch -a
* branch1
  master
  remotes/origin/main
```

branch1 will have all the files of the master branch as it is originated from the master.

```
$ ls
README.md  file1.txt  file2.txt  file3.txt
```

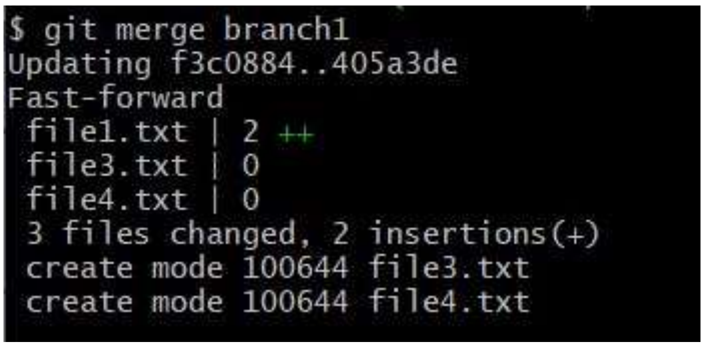
In branch1, I would like to make changes to file1.txt and create another text file names file4.txt

Now add these files to the staging area and commit. If you now check the master branch, these changes are not yet made there.

To make these changes to the master branch, we need to merge branch1 with master.

$ git checkout master

$ git merge branch1



```
$ git merge branch1
Updating f3c0884..405a3de
Fast-forward
 file1.txt | 2 ++
 file3.txt | 0
 file4.txt | 0
 3 files changed, 2 insertions(+)
 create mode 100644 file3.txt
 create mode 100644 file4.txt
```

To revert to a particular commit, we can use the first 8 digits of the hexadecimal code of a respective commit

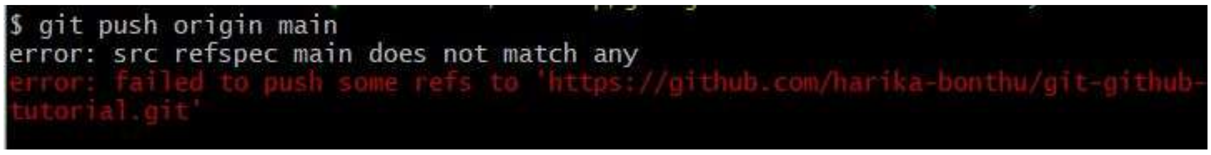
git checkout **8digitcode** file1.txt

```
git checkout f3c0884b file1.txt

Updated 1 path from 32610ca
```

Once we are done working, we need to push all these code files to the central/remote repository.

git push to send all files to the remote repository.

```
$ git push origin main
```



```
$ git push origin main
error: src refspec main does not match any
error: failed to push some refs to 'https://github.com/harika-bonthu/git-github-tutorial.git'
```

If you encounter such a problem, use the below command.

```
$ git push origin HEAD:main
```

Now go to your GitHub and TADA your files are hosted on the central repository.