

Git and Github for Beginners

What is GIT and why should you care?

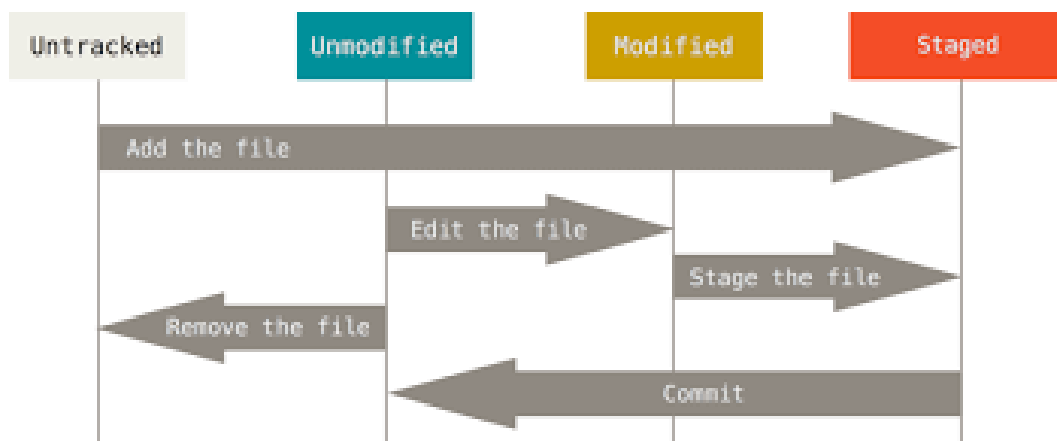
Git is the most widely used version control system on the planet. Professionals use it to develop effectively when working in teams as well as a safety net when working solo. To understand the GIT, one simply needs to think about a world without version control.

Consider a big project employing many developers. Individual developers work on several components of a system parallelly and completely independent of each other. These files need to then be organised and put together before release. Even after doing all this work, the codebase is filled with errors caused by friction between the components. It is difficult for any one developer to get access to the other's code and just one bad file can begin a long debugging session to rectify the mistake.

Git provides a distributed system, ie, a system where each individual developer can access the entire codebase leading to increased efficiency. It supports rollback to previous versions of a project in case of unfixable errors and enables parallel workflow by making use of branching. Commit messages state exactly what changes, and bad commits are quickly identified. All these factors, (along with many others) make Git a worthwhile technology to learn.

Basic Git workflow

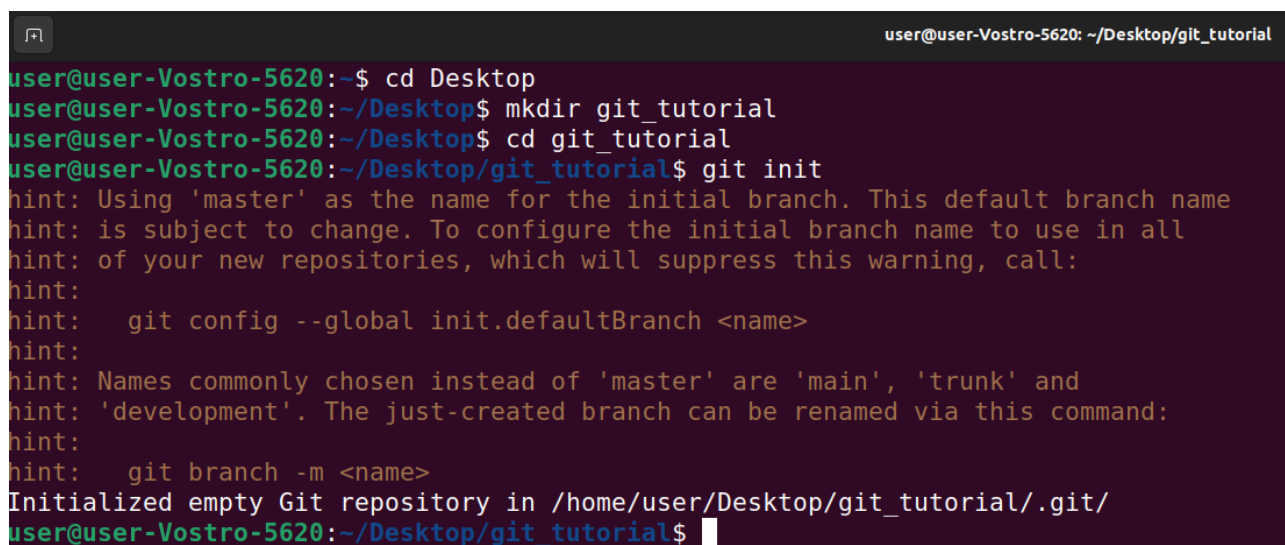
Alright, you want to learn Git. Great. First, it is important to have a clear understanding of how Git tracks and saves versions. See the below diagram.



Files are by default untracked. Git has no idea if it has changed or even what content it has. The files need to be staged for Git to know about them. Modified files after saving can be staged (ie, put into staging area). After staging commit the files to save this current version into the current branch. Be sure to give a commit message stating what has changed.

Trying it out

After completing installation of Git (which is completely free) open up the terminal of your choice and navigate to the target folder containing files to be tracked. Now, we have to initialise a git repository. A git repository is a folder where previous versions are stored. Use the 'git init' command to create a repository. See the example given below. Note that I am using a Linux machine and terminal commands (not git ones) can vary between operating systems.

A terminal window with a dark background and light green text. The title bar at the top reads 'user@user-Vostro-5620: ~/Desktop/git_tutorial'. The terminal shows the following commands and output:

```
user@user-Vostro-5620:~$ cd Desktop
user@user-Vostro-5620:~/Desktop$ mkdir git_tutorial
user@user-Vostro-5620:~/Desktop$ cd git_tutorial
user@user-Vostro-5620:~/Desktop/git_tutorial$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/user/Desktop/git_tutorial/.git/
user@user-Vostro-5620:~/Desktop/git_tutorial$
```

Now, create a file (preferably text file) and store some data in it. In this example I will be using a file 'test.txt' with content 'Hello world'. Type 'git status' to check the staging area as well as check for any unmodified files. Then, use 'git add <filename>' to add file to staging area. Check the status again. You are now ready to commit your first change and git begins tracking the file.

```
user@user-Vostro-5620: ~/Desktop/git_tutorial
user@user-Vostro-5620:~/Desktop/git_tutorial$ ls
test.txt
user@user-Vostro-5620:~/Desktop/git_tutorial$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
user@user-Vostro-5620:~/Desktop/git_tutorial$ git add test.txt
user@user-Vostro-5620:~/Desktop/git_tutorial$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
       new file:   test.txt

user@user-Vostro-5620:~/Desktop/git_tutorial$
```

To commit a change use ‘git commit – m “<message>”’. Commit history can be viewed using ‘git log’. Note that you are now committing to the main branch which is either ‘main’ or ‘master’. Now change the content of the file, save, add to staging area and commit your second change with another message. Check the commit history again to verify.

```
user@user-Vostro-5620: ~/Desktop/git_tutorial
user@user-Vostro-5620:~/Desktop/git_tutorial$ git commit -m "First commit"
[master (root-commit) 2b223ae] First commit
1 file changed, 1 insertion(+)
create mode 100644 test.txt
user@user-Vostro-5620:~/Desktop/git_tutorial$ git log
commit 2b223ae785746178d7f2ab40f17e789f8fab0552 (HEAD -> master)
Author: Nykaj A K <2nykajak@gmail.com>
Date:   Wed Dec 28 15:54:22 2022 +0530

    First commit

user@user-Vostro-5620:~/Desktop/git_tutorial$ nano test.txt
user@user-Vostro-5620:~/Desktop/git_tutorial$ git add test.txt
user@user-Vostro-5620:~/Desktop/git_tutorial$ git commit -m "Changed Hello world to Hello people"
[master b01643d] Changed Hello world to Hello people
1 file changed, 1 insertion(+), 1 deletion(-)
user@user-Vostro-5620:~/Desktop/git_tutorial$ git log
commit b01643d31799cda4bc42d05f77d10a05bd8dc03c (HEAD -> master)
Author: Nykaj A K <2nykajak@gmail.com>
Date:   Wed Dec 28 15:55:19 2022 +0530

    Changed Hello world to Hello people

commit 2b223ae785746178d7f2ab40f17e789f8fab0552
Author: Nykaj A K <2nykajak@gmail.com>
Date:   Wed Dec 28 15:54:22 2022 +0530

    First commit

user@user-Vostro-5620:~/Desktop/git_tutorial$
```

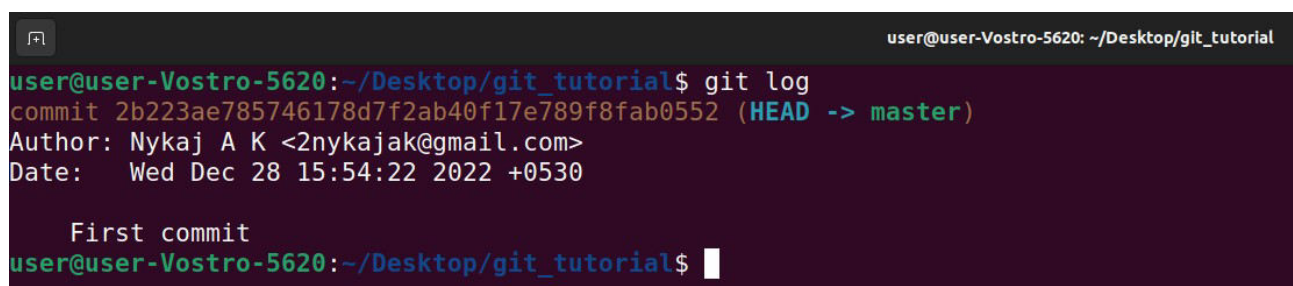
Congratulations! You have created two versions of your file and saved them in the repository. The git repository is hidden from users but can be accessed by 'ls -a'. By default it is saved as a .git folder.

Version control.

This is all well and good. But how do you revert to a previous version? What is this 'branch' thing and where is the promised flexibility? Let's walk through all that.

Reverting to previous versions.

In case of catastrophe, you need to load the backup. The previous versions which are stored in the .git folder can be accessed and loaded by their unique hash. Note that doing this using 'git reset <hash>' deletes all commits after the target making it dangerous to use. The preferred command to be used is 'git revert <hash>' which is safer but more difficult to use. This is the log after resetting to first commit. The file content will also be changed.

A terminal window with a dark background. The title bar shows 'user@user-Vostro-5620: ~/Desktop/git_tutorial'. The prompt is 'user@user-Vostro-5620:~/Desktop/git_tutorial\$'. The command 'git log' has been executed, showing the output: 'commit 2b223ae785746178d7f2ab40f17e789f8fab0552 (HEAD -> master)', 'Author: Nykaj A K <2nykajak@gmail.com>', and 'Date: Wed Dec 28 15:54:22 2022 +0530'. Below this, it says 'First commit'. The prompt is now 'user@user-Vostro-5620:~/Desktop/git_tutorial\$' with a cursor.

```
user@user-Vostro-5620:~/Desktop/git_tutorial$ git log
commit 2b223ae785746178d7f2ab40f17e789f8fab0552 (HEAD -> master)
Author: Nykaj A K <2nykajak@gmail.com>
Date: Wed Dec 28 15:54:22 2022 +0530

First commit
user@user-Vostro-5620:~/Desktop/git_tutorial$
```

Using branches and merging

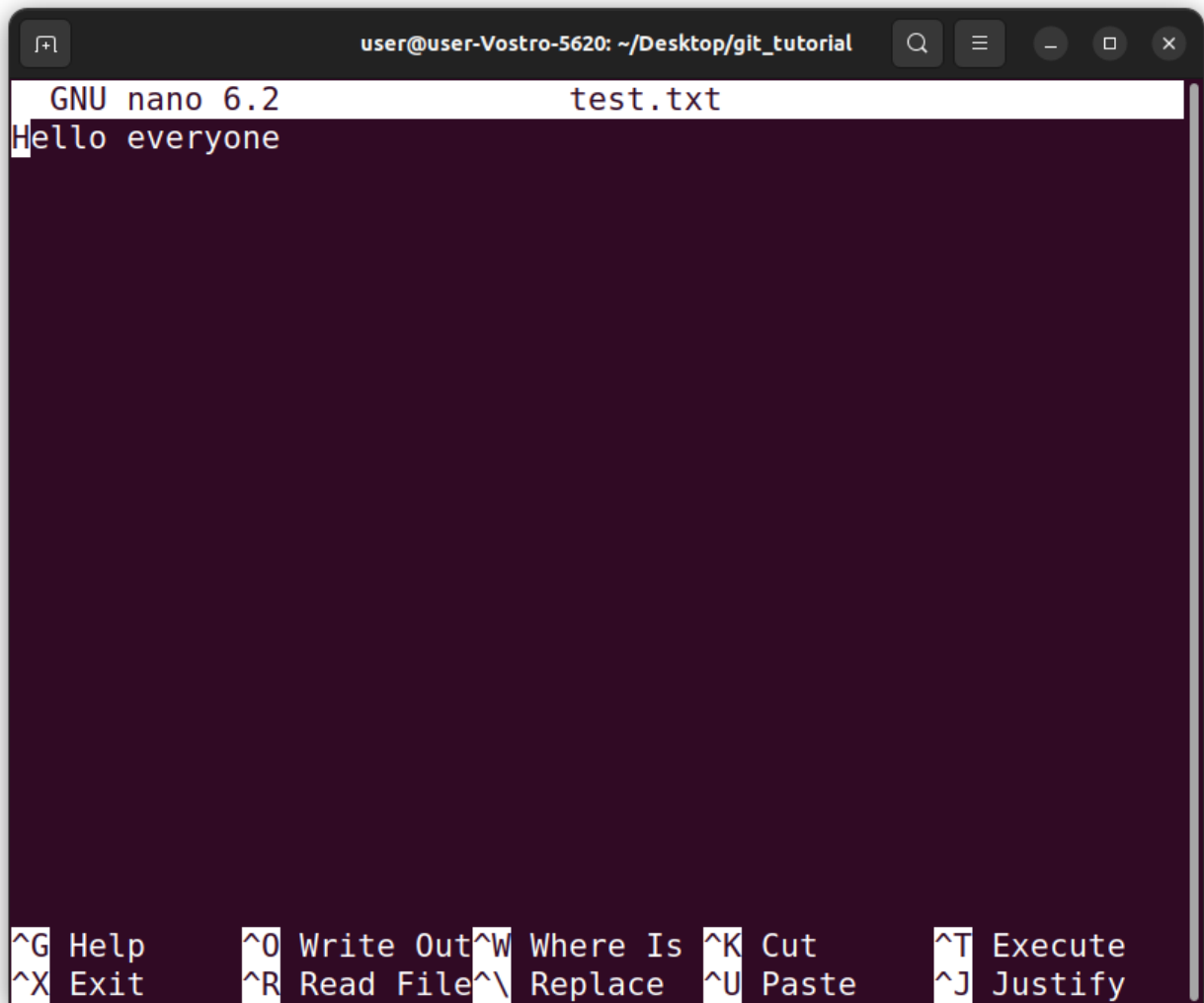
Git branches contain copies of the 'main' or 'master' branch which can be modified without modifying the main branch. After a new feature is made this branch is merged with the main one to update it. Multiple branches allow parallel workflow and a more efficient team.

Use 'git branch' to see all current branches, 'git branch <branchname>' to create a new branch and 'git switch <branchname>' to jump to the named branch. Branches can be merged after changes into another one. This is done by navigating to the branch to which the other is to be merged and executing 'git merge <targetbranch>'. See examples below.

```
user@user-Vostro-5620: ~/Desktop/git_tutorial
user@user-Vostro-5620:~/Desktop/git_tutorial$ git branch
* master
user@user-Vostro-5620:~/Desktop/git_tutorial$ git branch newbranch
user@user-Vostro-5620:~/Desktop/git_tutorial$ git branch
* master
  newbranch
user@user-Vostro-5620:~/Desktop/git_tutorial$ git switch newbranch
M      test.txt
Switched to branch 'newbranch'
user@user-Vostro-5620:~/Desktop/git_tutorial$ git branch
  master
* newbranch
user@user-Vostro-5620:~/Desktop/git_tutorial$
```

The above commands created a 'newbranch' branch and moves to it. I will change test.txt to contain 'Hello everyone' and merge 'newbranch' with master. See the below image.

```
user@user-Vostro-5620: ~/Desktop/git_tutorial
user@user-Vostro-5620:~/Desktop/git_tutorial$ nano test.txt
user@user-Vostro-5620:~/Desktop/git_tutorial$ git add test.txt
user@user-Vostro-5620:~/Desktop/git_tutorial$ git commit -m "Changed people to everyone"
[newbranch b6aa241] Changed people to everyone
1 file changed, 1 insertion(+), 1 deletion(-)
user@user-Vostro-5620:~/Desktop/git_tutorial$ git switch master
Switched to branch 'master'
user@user-Vostro-5620:~/Desktop/git_tutorial$ git merge newbranch
Updating 2b223ae..b6aa241
Fast-forward
 test.txt | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
user@user-Vostro-5620:~/Desktop/git_tutorial$ nano test.txt
user@user-Vostro-5620:~/Desktop/git_tutorial$
```



```
user@user-Vostro-5620: ~/Desktop/git_tutorial
GNU nano 6.2 test.txt
Hello everyone

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify
```

Github

It is an online service that provides various services, most notable among them being an online repository compatible with Git. Open source code is hosted on these repositories which can be improved upon by contributors. This website provides a unique opportunity to contribute to real world applications and learn from industry code. This provides a place for code to be hosted for easy access from anyplace and at any time.