Git Tutorial Using Command Line

Jesus Cornejo

February 2024

1 Prerequisites

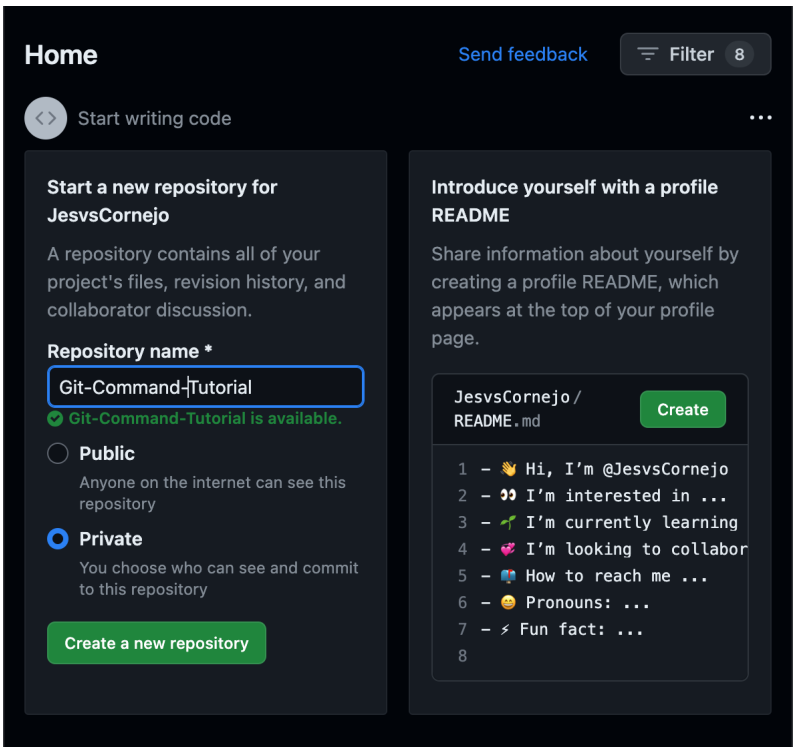
To follow along with this tutorial it is recommended that you have: - A basic understanding of the command line - Git installed and GitHub account set up

2 What is Git?

Git is a powerful tool used in software development that allows you to keep track of the changes you make to your files, and lets you share them with other developers who work on the same project. Git is a distributed system, meaning that each developer has a copy of the entire code history on their own computer.

3 Getting Started:

To start we will create a new repository on GitHub. A repository is where you store and manage your files using Git. A repository tracks the history and changes of your files, and allows you to work with other developers on the same project. It should look like this:



Once you've created a repository we can clone it to your computer using the command line with the command: **git clone [URL]**
You can find and copy the link to the URL of your repository on GitHub by left-clicking on the tab **Code**.

4 Creating a File:

In order to access and work within your repository use the 'cd' command: **cd *repository name***
At this point there are no files in the repository. Let's start by making a python file using the 'mkdir' command **mkdir *your file name*** I named my file heyGit.py
Once you successfully make the file you can use the 'ls -la' command to see the file in the repository.
It should look something like this:

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Desktop % cd Git-Command-Tutorial
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial % mkdir heyGit
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial % ls -la
total 0
drwxr-xr-x@  4 jesuscornejo  staff   128 Feb  9 19:09 .
drwx------@ 68 jesuscornejo  staff  2176 Feb  9 19:07 ..
drwxr-xr-x@  9 jesuscornejo  staff   288 Feb  9 19:07 .git
drwxr-xr-x@  2 jesuscornejo  staff    64 Feb  9 19:09 heyGit.py
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial %
```

5 Editing Files:

Now that we have a python file in our repository, we can learn how to edit the file and then we'll be able to commit our first changes in the repository.
To edit the file we use the 'vim' command: **vim *file name*** A new window will open, you must hit the 'i' key first to insert and write code. Then hit the 'escape' key and type '**:wq**' (write and quit) at the bottom to save your changes.

For my heyGit.py file I added a simple print statement.

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial % ls
heyGit.py
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial % vim heyGit
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial %
```

```
print("Hey Git users!!!")
~
~
~
```

6 Committing:

Now let's use the '**git status**' command to show us the changes we made to the file

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial % git status
On branch main

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial %
```

Here we see that I made a change, my untracked file: **heyGit**
Our next step is to commit the changes, but first we need to move our files to the staging area. The staging area, also known as index, is where you can selectively choose which changes to include in the next commit. By using the command:'**git add -A**' we add all files to the staging area.
Once we've added the file to the staging area we can commit to the repository using: '**git commit -m "Commit message"**'
The commit message should describe the changes you are committing in some way. For example I used the message: 'git commit -m "initial commit". Once you've successfully committed it should look like this:

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial % git add -A
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial % git commit -m
[main (root-commit) 2c32fc9] initial commit
 Committer: Jesus Cornejo <jesuscornejo@Jesuss-MacBook-Air-3.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 heyGit
```

7 Branches:

Now that we have a repository and have committed a file, lets take a look at branches in Git. Branches allow you to develop features, fix bugs, and safely experiment with new ideas in a contained area of your repository. The master or "**main**" branch remains stable while you or other developers work on different branches simultaneously.
To show the use of branching lets start by cloning this repository into a remote one. First we make a new repository just like before. I created the repository: **Git-Command-Remote** and have it cloned locally to my computer. Before we clone the old repository, we need to start by moving into our new remote repository using the 'cd' command: **cd Git-Command-Remote**.

If we clone our old repository into this remote one now we may run into an error since there is a .git file already in our repository (you can use the **ls -a** command to see the file).
We can get around this error using the command: **rm -rf .git**. This will remove the first .git files that are causing the error.
After deletion we can now clone the old repository, but in this case we use the git clone command like this: 'git clone **../Git-Command-Tutorial .**' instead of using the URL link.

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 ~ % cd Desktop
(base) jesuscornejo@Jesuss-MacBook-Air-3 Desktop % cd Git-Command-Tutorial
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial % cd Desktop
cd: no such file or directory: Desktop
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial % cd -
~/Desktop
(base) jesuscornejo@Jesuss-MacBook-Air-3 Desktop % git clone https://github.com/JesvsCornejo/Git-Command-Remote
Cloning into 'Git-Command-Remote'...
warning: You appear to have cloned an empty repository.
(base) jesuscornejo@Jesuss-MacBook-Air-3 Desktop % cd Git-Command-Remote
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % ls -a
.       ..      .git
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % rm -rf .git
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git clone ../Git-Command-Tutorial .
Cloning into '.'...
done.
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % ls -la
total 8
drwxr-xr-x@  4 jesuscornejo  staff   128 Feb 11 21:18 .
drwx------@ 74 jesuscornejo  staff  2368 Feb 11 21:15 ..
drwxr-xr-x@ 12 jesuscornejo  staff   384 Feb 11 21:18 .git
-rw-r--r--   1 jesuscornejo  staff    26 Feb 11 21:18 heyGit
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote %
```

We see that our remote repository now has the heyGit.py file. Lets edit this file and see how we can push changes using branches. Using the **vim** command again I simply added another print statement:

```
print("Hey Git users!!!")

print("I like to branch out")


~
~
~
```

Now before we push our changes to our target repository we need to create a temporary branch in the old repository. To do this, we must navigate our way to the old repository using the 'cd' commands
Then use the command: **git checkout -b** [**temporary branch name**]. Now that we're on a new branch we can head back to the remote repository and use the command: **git push**

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 ~ % cd Desktop
(base) jesuscornejo@Jesuss-MacBook-Air-3 Desktop % cd Git-Command-Tutorial
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial % git checkout -b
temp
Switched to a new branch 'temp'
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Tutorial %
```

Up until now we have been working on our main branch, but now we'll create a new branch for us to work in. Use the command: **git branch *name of new branch*** to create a new branch in the repository.
We can use the **git branch** command without the name to see all of the branches in the repository. Here you can see my new branch which I named **heyGitBranch** then I used the command: **git checkout -name of branch-** to move into the new branch. My new branch is highlighted:

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git branch heyGitBranch
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git branch
  heyGitBranch
* main
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git checkout heyGitBranch
M       heyGit
Switched to branch 'heyGitBranch'
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git branch
* heyGitBranch
  main
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote %
```

8 Merging:

Once you've finished doing work on a branch it is sometimes beneficial for developers to merge branches back to the main branch. We will start by making some changes to our python file on our new branch. Once again, I added a simple print statement and now we will commit just like we did before.

```
print("Hey Git users!!!")

print("I like to branch out")

print("See you later!")


~
~
~
```

After committing let's push this branch to our original repository, to do that use the command: **git push -u origin *branch name***

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git push -u origin heyGitBranch
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 334 bytes | 334.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To /Users/jesuscornejo/Desktop/Git-Command-Remote/../Git-Command-Tutorial
 * [new branch]      heyGitBranch -> heyGitBranch
branch 'heyGitBranch' set up to track 'origin/heyGitBranch'.
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote %
```

Now that we've made the changes on one branch, lets see how we can merge our two branches.
First move back into our main branch using our checkout command: **git checkout main**. If we use the command: **git branch –merged** we will only see our main branch since we haven't merged our new one

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git branch --merged
* main
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote %
```

So let's merge our branches! Use the command: **git merge *name of branch***

Then push to the original repository using the command: **git push origin main**

8.1 Deleting Branches:

Now that we've merged into main we can delete our branch. Use the command: **git branch -d [name of branch]**
Once the branch is deleted locally we need to delete it from the original repository. Use the command: **git push origin –delete [name of branch]**

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git merge heyGitBranch
Updating 2c32fc9..a2add34
Fast-forward
 heyGit | 6 ++++++
 1 file changed, 6 insertions(+)
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To /Users/jesuscornejo/Desktop/Git-Command-Remote/../Git-Command-Tutorial
   2c32fc9..a2add34  main -> main
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote %
```

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git branch -d heyGitBranch
Deleted branch heyGitBranch (was a2add34).
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git push origin --delete heyGitBranch
To /Users/jesuscornejo/Desktop/Git-Command-Remote/../Git-Command-Tutorial
 - [deleted]         heyGitBranch
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote %
```

9 Stashes:

After looking at how branches and commits work, we should also learn how to save our work when its not ready to commit.
Stashes are really useful in this case because they temporarily shelve changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on. To show this, let's make a change on our python file using 'vim' command. Once the file has changes made we can use the **git status** command to see the change:

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % vim heyGit
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   heyGit

no changes added to commit (use "git add" and/or "git commit -a")
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote %
```

Here we see that the file has changed but is not staged or committed. Let's say that now we want to change branches and work on something else. We can use the command '**git stash**' to save our work so we can reapply it later.

Here we see that, after stashing, our working tree is clean and our changes are saved.

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git stash
Saved working directory and index state WIP on main: a2add34 added print statements
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote %
```

9.1 Saved Stashes:

To bring back our work we can use the command: **git stash apply**

```
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote % git stash apply
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   heyGit

no changes added to commit (use "git add" and/or "git commit -a")
(base) jesuscornejo@Jesuss-MacBook-Air-3 Git-Command-Remote %
```

Here we see that we are back to our original state with our changes ready to be staged and committed.