All The Git Commands You Need to Know About

Git is one of the most sought after DevOps tools used to handle small and large projects efficiently. The tool enables collaboration among different people in different parts of the same project. Git is nothing without its commands, so here, all about Git commands that are needed to efficiently and effectively work on the tool.

What is Git?

Git is a widely used modern version control system for tracking changes in computer files. The term version control system suggests a system that records all the changes made to a file or set of data, so a specific version can be considered whenever needed. This feature makes the process of collaboration so feasible with all team members, making it considerably more comfortable to work over a big project.

Git makes it possible for several people involved in the project to work together and track each other's progress over time. In software development, the tool helps in Source Code Management. Git favors not only programmers but also non-technical users by keeping track of their project files.

While working on Git, we actively use two repositories.

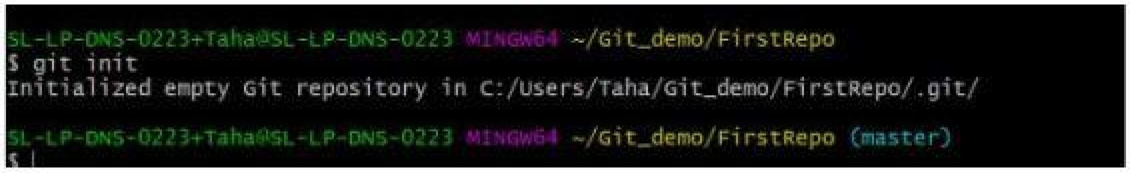
- Local repository: The local repository is present on our computer and consists of all the files and folders. This Repository is used to make changes locally, review history, and commit when offline.
- Remote repository: The remote repository refers to the server repository that may be present anywhere. This repository is used by all the team members to exchange the changes made.

Both repositories have their own set of commands. There are separate Git Commands that work on different types of repositories.

Git Commands: Working With Local Repositories

- git init
- The command git init is used to create an empty Git repository.
- After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.

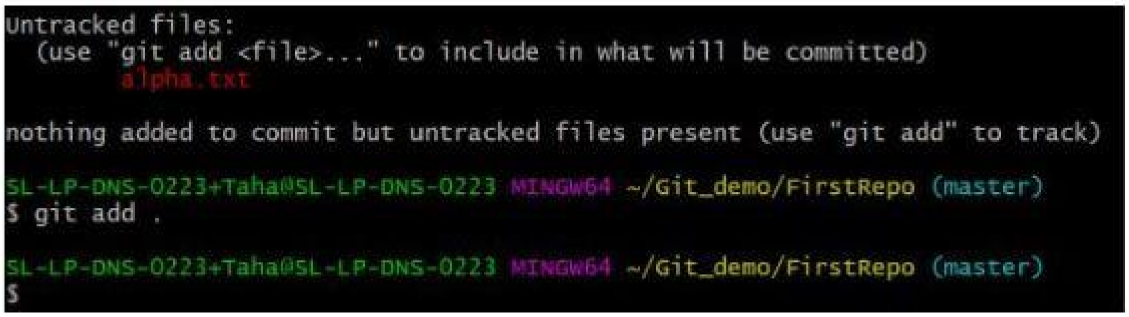
```
git init
```



```
SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo
$ git init
Initialized empty Git repository in C:/Users/Taha/Git_demo/FirstRepo/.git/

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ |
```

- git add
- Add command is used after checking the status of the files, to add those files to the staging area.
- Before running the commit command, "git add" is used to add any new or modified files.

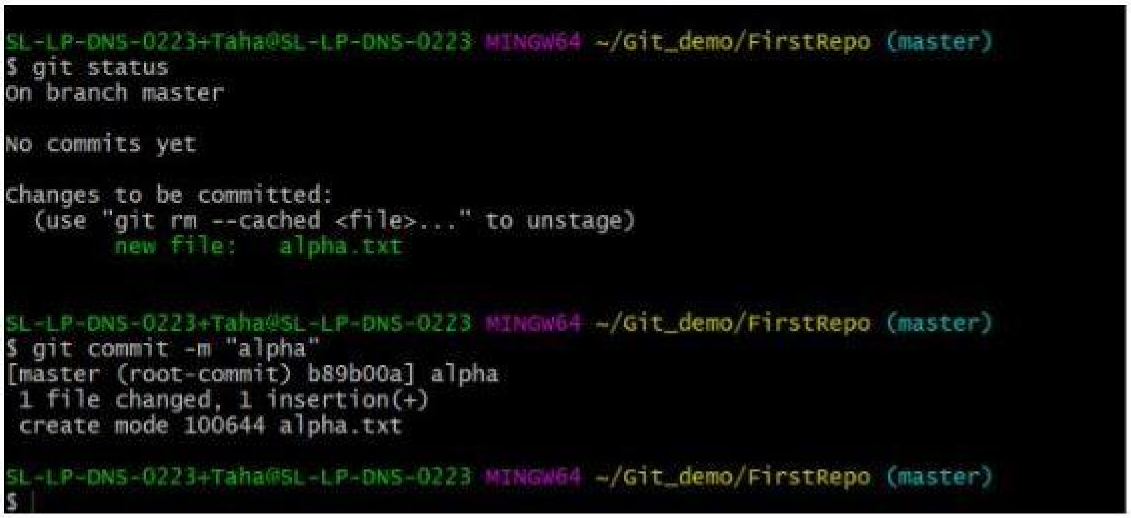
```
git add .
```



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

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

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ git add .

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$
```

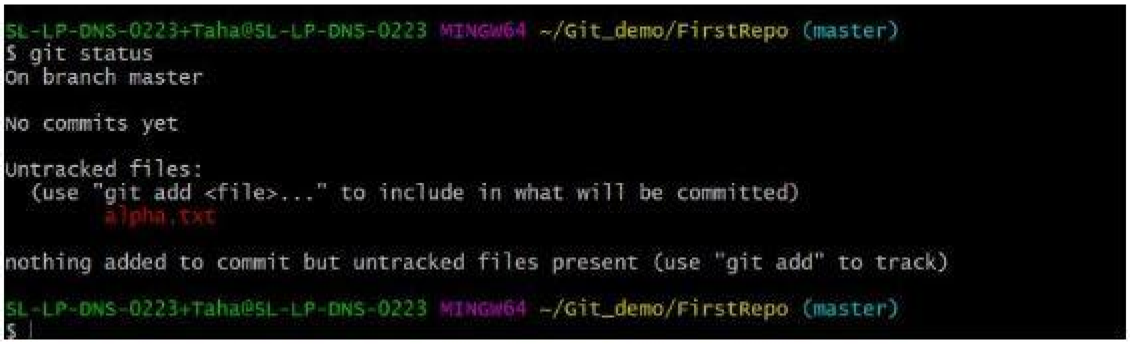
- git commit
- The commit command makes sure that the changes are saved to the local repository.
- The command "git commit –m <message>" allows you to describe everyone and help them understand what has happened.

```
git commit -m “commit message”
```

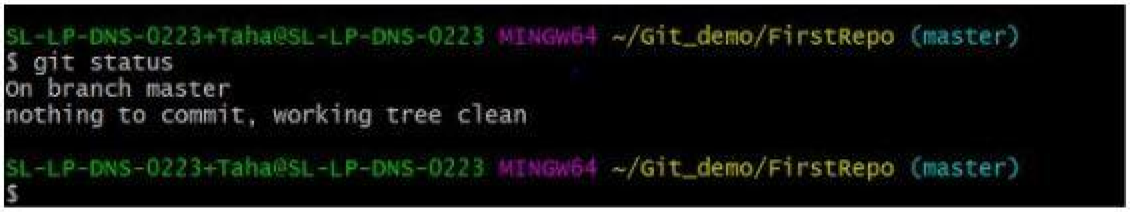

```
SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ git status
On branch master

No commits yet

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


SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ git commit -m "alpha"
[master (root-commit) b89b00a] alpha
 1 file changed, 1 insertion(+)
 create mode 100644 alpha.txt

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ 
```

- git status
- The git status command tells the current state of the repository.
- The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status. Also, if there are no changes, it will show the message no changes to commit, working directory clean.

```
git status
```



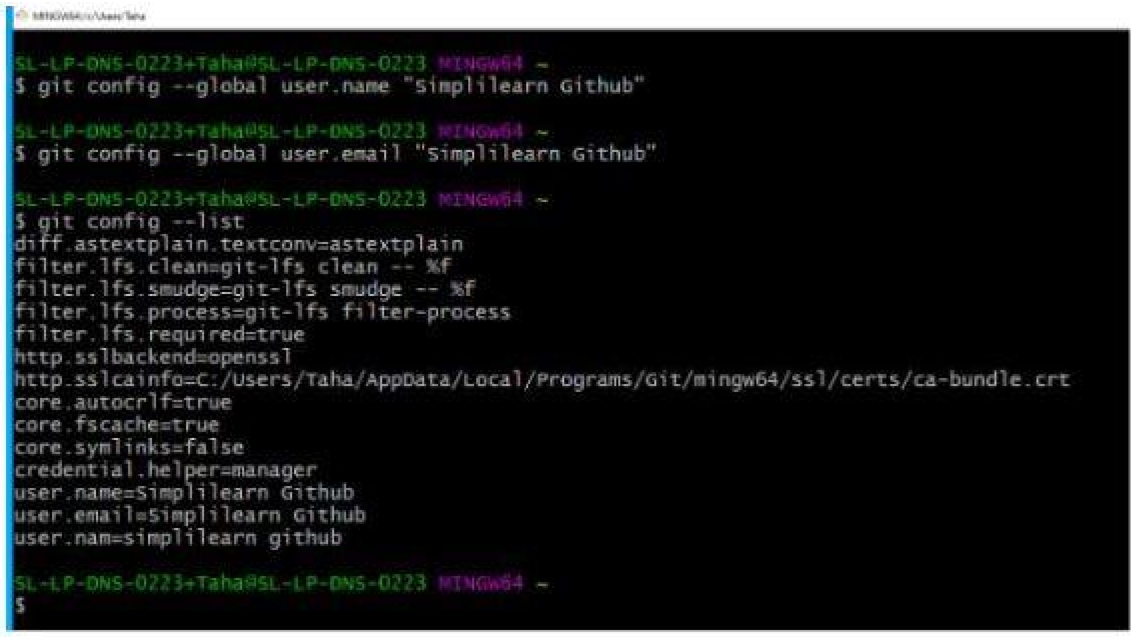
```
SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ git status
On branch master

No commits yet

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

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

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ |
```



```
SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ git status
On branch master
nothing to commit, working tree clean

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$
```

- git config
- The git config command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository.
- When git config is used with --global flag, it writes the settings to all repositories on the computer.

```
git config --global user.name “any user name”

git config --global user.email <email id>
```



```
MINGW64:/c/Users/Taha

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~
$ git config --global user.name "Simplilearn Github"

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~
$ git config --global user.email "Simplilearn Github"

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Users/Taha/AppData/Local/Programs/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
credential.helper=manager
user.name=Simplilearn Github
user.email=Simplilearn Github
user.nam=simplilearn github

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~
$
```

- git branch
- The git branch command is used to determine what branch the local repository is on.
- The command enables adding and deleting a branch.

Create a new branch git branch <branch_name>
List all remote or local branches git branch -a

```
# Delete a branch

  git branch -d <branch_name>
```

-
  git checkout
- The git checkout command is used to switch branches, whenever the work is to be started on a different branch.
- The command works on three separate entities: files, commits, and branches.

```
# Checkout an existing branch

  git checkout <branch_name>
```

```
# Checkout and create a new branch with that name

  git checkout -b <new_branch>
```

-
  git merge
- The git merge command is used to integrate the branches together. The command combines the changes from one branch to another branch.
- It is used to merge the changes in the staging branch to the stable branch.

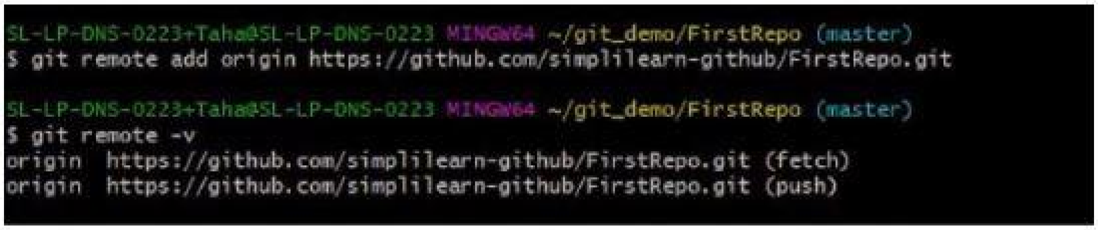
```
git merge <branch_name>
```

However, these are popular and basic git commands used by developers.

Git Commands: Working With Remote Repositories

- git remote
- The git remote command is used to create, view, and delete connections to other repositories.
- The connections here are not like direct links into other repositories, but as bookmarks that serve as convenient names to be used as a reference.

```
git remote add origin <address>
```



```
SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/git_demo/FirstRepo (master)
$ git remote add origin https://github.com/simplilearn-github/FirstRepo.git

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/git_demo/FirstRepo (master)
$ git remote -v
origin  https://github.com/simplilearn-github/FirstRepo.git (fetch)
origin  https://github.com/simplilearn-github/FirstRepo.git (push)
```

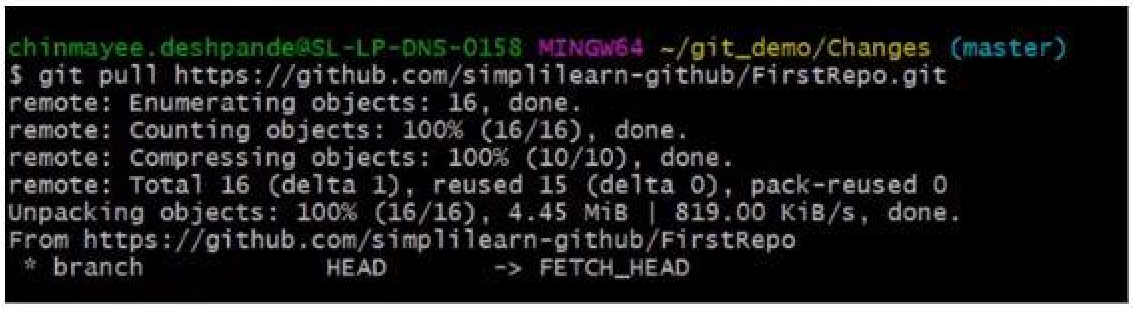
- git clone
- The git clone command is used to create a local working copy of an existing remote repository.
- The command downloads the remote repository to the computer. It is equivalent to the Git init command when working with a remote repository.

```
git clone <remote_URL>
```

-

  git pull
- The git pull command is used to fetch and merge changes from the remote repository to the local repository.
- The command "git pull origin master" copies all the files from the master branch of the remote repository to the local repository.

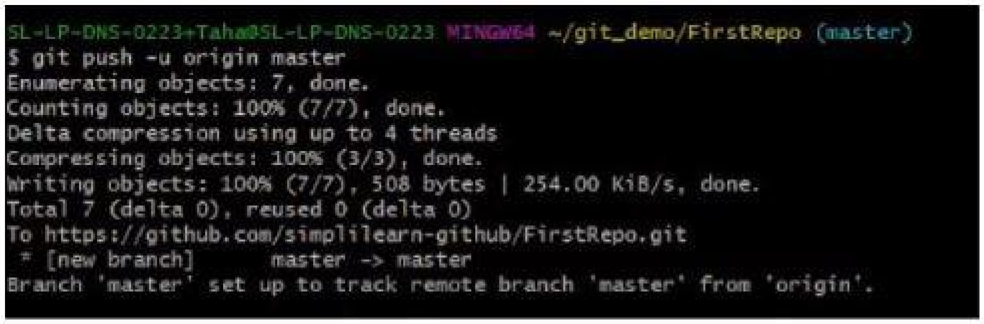
```
git pull <branch_name> <remote URL>
```



```
chinmayee.deshpande@SL-LP-DNS-0158 MINGW64 ~/git_demo/Changes (master)
$ git pull https://github.com/simplilearn-github/FirstRepo.git
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 16 (delta 1), reused 15 (delta 0), pack-reused 0
Unpacking objects: 100% (16/16), 4.45 MiB | 819.00 KiB/s, done.
From https://github.com/simplilearn-github/FirstRepo
 * branch            HEAD       -> FETCH_HEAD
```

- git push
- The command git push is used to transfer the commits or pushing the content from the local repository to the remote repository.
- The command is used after a local repository has been modified, and the modifications are to be shared with the remote team members.

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



```
SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/git_demo/FirstRepo (master)
$ git push -u origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (7/7), 508 bytes | 254.00 KiB/s, done.
Total 7 (delta 0), reused 0 (delta 0)
To https://github.com/simplilearn-github/FirstRepo.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Some Advanced Git Commands

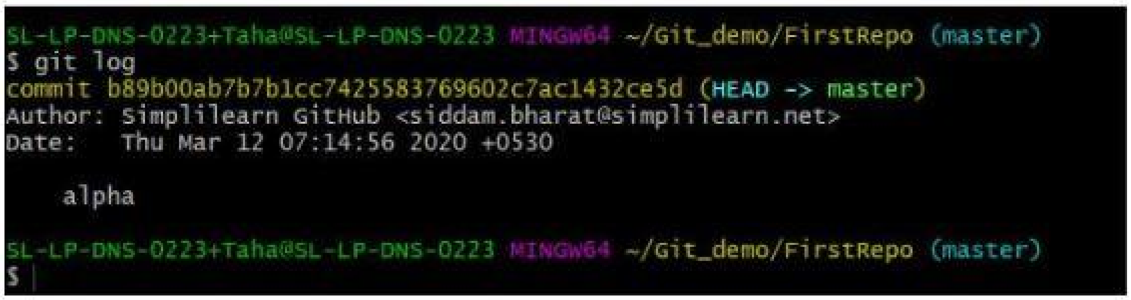
- git stash
- The git stash command takes your modified tracked files and saves it on a pile of incomplete changes that you can reapply at any time. To go back to work, you can use the stash pop.
- The git stash command will help a developer switch branches to work on something else without committing to incomplete work.

```
# Store current work with untracked files

 git stash -u
```

```
# Bring stashed work back to the working directory

  git stash pop
```

-
  ### git log
- The git log command shows the order of the commit history for a repository.
- The command helps in understanding the state of the current branch by showing the commits that lead to this state.

```
git log
```



```
SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ git log
commit b89b00ab7b7b1cc7425583769602c7ac1432ce5d (HEAD -> master)
Author: Simplilearn GitHub <siddam.bharat@simplilearn.net>
Date:   Thu Mar 12 07:14:56 2020 +0530

    alpha

SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ |
```

Conclusion

We hope this Git commands tutorial has helped you understand various useful commands in Git. You have learned the basics of Git and the different commands that are used, and also saw different basic Git commands and advanced too. according to the repository they are used in, followed by some advanced commands. We understood what each of the mentioned commands does and also came across the syntax of each command.

In order to gain expertise in the principles of continuous development and deployment, automation of configuration management, inter-team collaboration, and IT service agility, using DevOps tools like Git, Docker, Jenkins, Cucumber, Ansible, TeamCity, and Nagios, you can check out the DevOps Online Training Course.

Do you have any questions for us? Let us know in the comment section of the “All You Need to Know About Git Commands”. We will have our experts let it answer it for you.