What is version control?

How version control helps high performing development and DevOps teams prosper

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time. As development environments have accelerated, version control systems help software teams work faster and smarter. They are especially useful for DevOps teams since they help them to reduce development time and increase successful deployments.

Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

Benefits of version control systems

Using version control software is a best practice for high performing software and DevOps teams. Version control also helps developers move faster and allows software teams to preserve efficiency and agility as the team scales to include more developers.

Version Control Systems (VCS) have seen great improvements over the past few decades and some are better than others. VCS are sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System). One of the most popular VCS tools in use today is called Git. Git is a *Distributed* VCS, a category known as DVCS, more on that later. Like many of the most popular VCS systems available today, Git is free and open source.

Git is a popular DevOps tool used for source code management. It is one of the most popular version control systems today that is widely used to handle all sizes of projects effectively and efficiently. It helps in tracking changes in the source code, allowing different people to work on different parts of the same program. Git is a recognized approach to contribute to a project collaboratively.

What is Git?

Git is a version control system for monitoring the changes in computer files. It is used to collaborate with several people on a project and track progress throughout the project. Whenever a developer wishes to start working on something, a new branch is created, to ensure that the master branch always has a production-quality code.

What is GitHub?

GitHub is a website based service that is used by developers all over the world to store and share their code with other developers. Git repository hosting service provides a web-based graphical interface, unlike Git. GitHub helps all the team members to work together on the project from anywhere. The team members can access files and easily merge changes with the master branch of the project.

GitHub is one place where project managers and developers coordinate, monitor, and update their work, so there is transparency in the project, and it stays on schedule. The packages can be published privately, or within the team or publicly for the open-source community.

Different Git Commands

There are several commands used in Git like:

1. Git config
2. Git init
3. Git add
4. Git diff
5. Git commit
6. Git reset
7. Git status
8. Git merge
9. Git push
10. Git pull

What are GitHub's Features?

1. Easy Project Management

GitHub is a place where project managers and developers come together to coordinate, track, and update their work so that projects are transparent and stay on schedule.

2. Increased Safety With Packages

Packages can be published privately, within the team, or publicly to the open-source community. The packages can be used or reused by downloading them from GitHub.

3. Effective Team Management

GitHub helps all the team members stay on the same page and organized. Moderation tools like Issue and Pull Request Locking help the team to focus on the code.

4. Improved Code Writing

Pull requests help the organizations to review, develop, and propose new code. Team members can discuss any implementations and proposals through these before changing the source code.

5. Increased Code Safety

GitHub uses dedicated tools to identify and analyze vulnerabilities to the code that other tools tend to miss. Development teams everywhere work together to secure the software supply chain, from start to finish.

6. Easy Code Hosting

All the code and documentation are in one place. There are millions of repositories on GitHub, and each repository has its own tools to help you host and release code.

How Do You Use Git and GitHub?

Here's a very broad overview of the steps you need to use both Git and GitHub. You can find more details regarding the specific commands and syntax here on opensource.com.

1. Create your GitHub account, which you should have already done, thanks to the previous section!
2. Create a repository or "repo" for short. This is where you store your code.
3. Build a file.
4. Make a commit. Whenever you create a file or change it, you create a Git commit to store the new version.
5. Connect your repo with your computer system.

GitHub's Competitors

The market provides many alternatives and competitors to GitHub. As of the end of 2020, the top ten competitors are:

1. Bitbucket
2. Google Cloud Source Repositories
3. Phabricator
4. GitLab
5. Gogs
6. Gitea
7. SourceForge
8. Apache Allura
9. Launchpad
10. AWS CodeCommit

The git init usage

Using git init is the simplest way of setting up version-controlled system projects, as there is no need to generate a repository, input files etc.

- In order to get a working Git repository, you only need to cd into your project subdirectory and run git init command into your terminal.
- **git init**

- Transform the directory into your Git repository, to record the project changes. Create a new Git repository in a particular directory to generate a new .git subdirectory.
  git init <directory>

The git clone usage

First of all, the git clone command is used to target an existing repository and clone or copy it in a new directory.

Cloning to a certain folder
You should make a clone of the repository at **<repo>** into the folder called **<directory>** on the local machine.
git clone <repo> <directory>

Cloning a certain tag
Clone the repository at **<repo>** and clone only the ref for **<tag>**.
git clone --branch <tag> <repo>

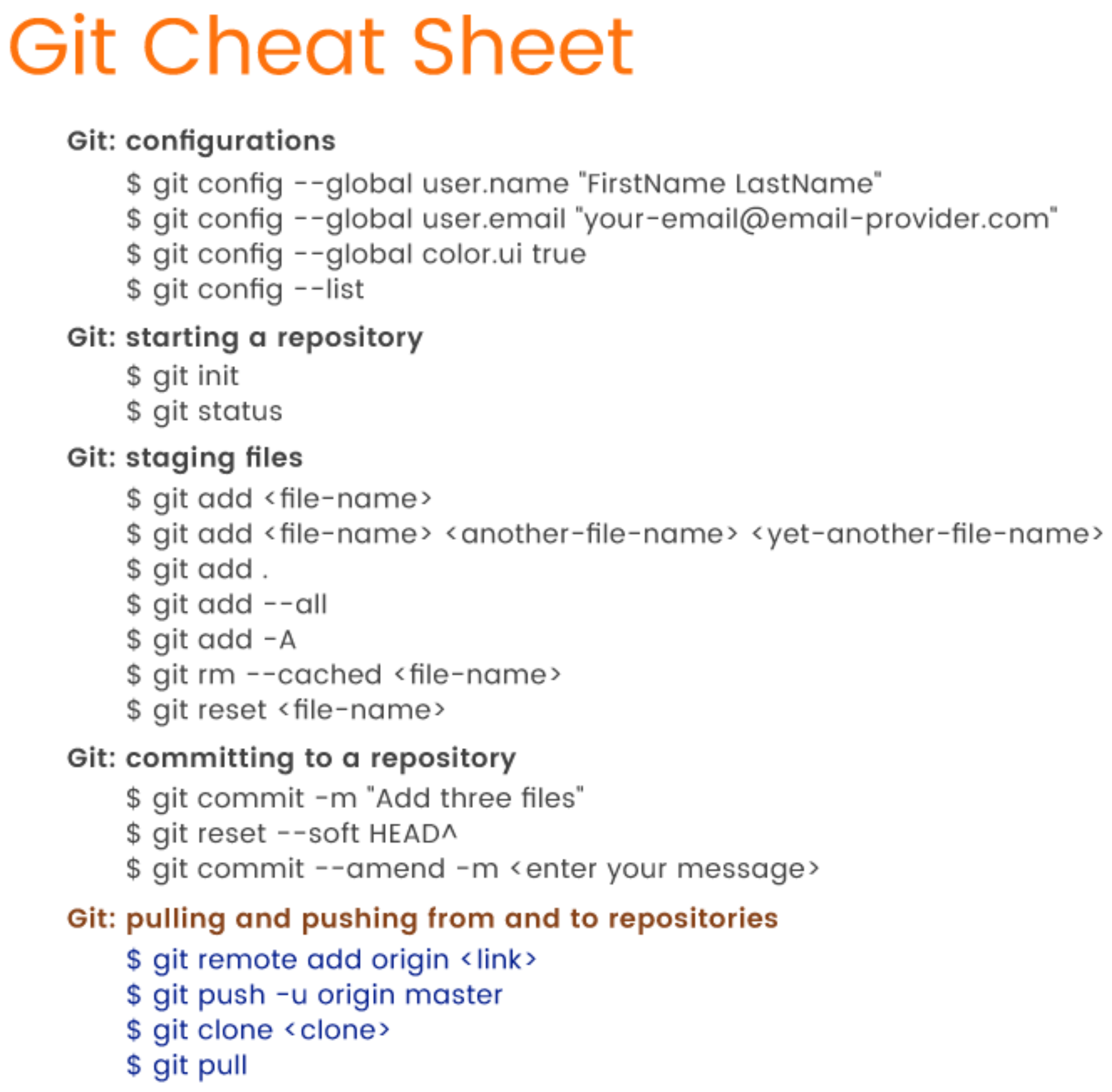
Difference Between git init and git clone
The git init and git clone are usually confused with each other. Here it's important to note that git clone is dependant on the git init and creates a copy of a repository that already exists. In other words, for generating a git clone, we need a repository created with git init. Only after that, we run a git clone to copy the data that is included in our repository mentioned above.

SSH
Secure Shell (SSH) is a network protocol, which helps to login from one computer to another securely. In most cases, SSH access to servers is configured by default. It's necessary to establish credentials with the hosting server before connecting.

git clone ssh://user@server/project.git

HTTPS
HTTPS stands for HyperText Transfer Protocol. This protocol is mostly used to transmit HTML data above the Internet. Git is configured to share information with HTTPS.

git clone http://example.com/gitproject.git



Git Cheat Sheet

Git: configurations

```
$ git config --global user.name "FirstName LastName"
$ git config --global user.email "your-email@email-provider.com"
$ git config --global color.ui true
$ git config --list
```

Git: starting a repository

```
$ git init
$ git status
```

Git: staging files

```
$ git add <file-name>
$ git add <file-name> <another-file-name> <yet-another-file-name>
$ git add .
$ git add --all
$ git add -A
$ git rm --cached <file-name>
$ git reset <file-name>
```

Git: committing to a repository

```
$ git commit -m "Add three files"
$ git reset --soft HEAD^
$ git commit --amend -m <enter your message>
```

Git: pulling and pushing from and to repositories

```
$ git remote add origin <link>
$ git push -u origin master
$ git clone <clone>
$ git pull
```

While working on Git, we actively use two repositories.

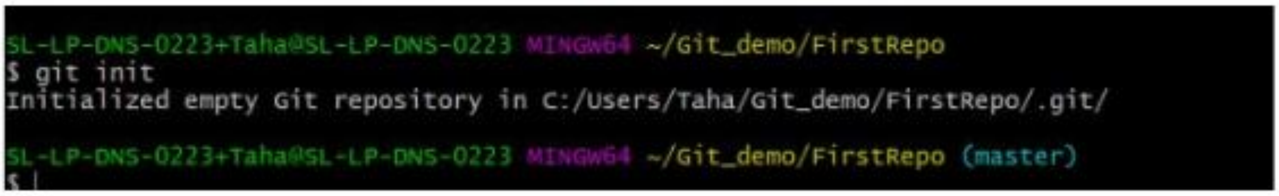
- Local repository: The local repository is present on our computer and consists of all the files and folders. This Repository is used to make changes locally, review history, and commit when offline.
- Remote repository: The remote repository refers to the server repository that may be present anywhere.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

Both repositories have their own set of commands

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
```

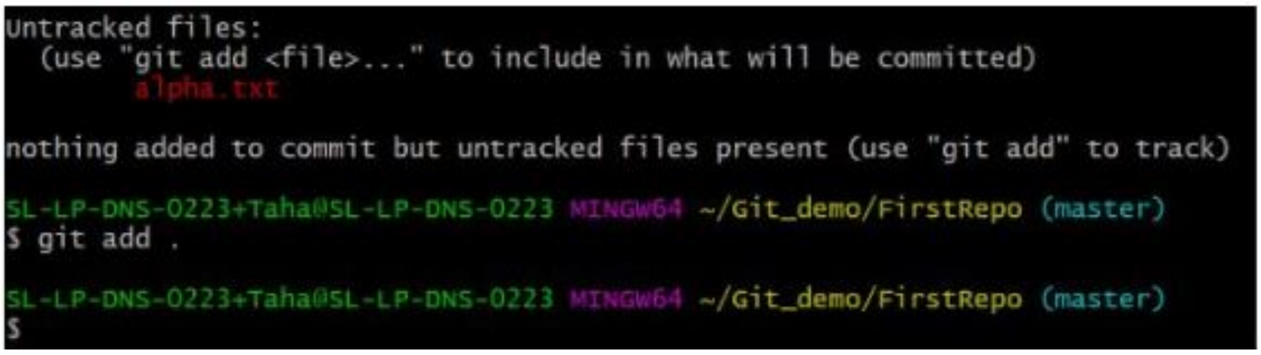


```
SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo
$ git init
Initialized empty Git repository in C:/Users/Taha/Git_demo/FirstRepo/.git/

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 .
```

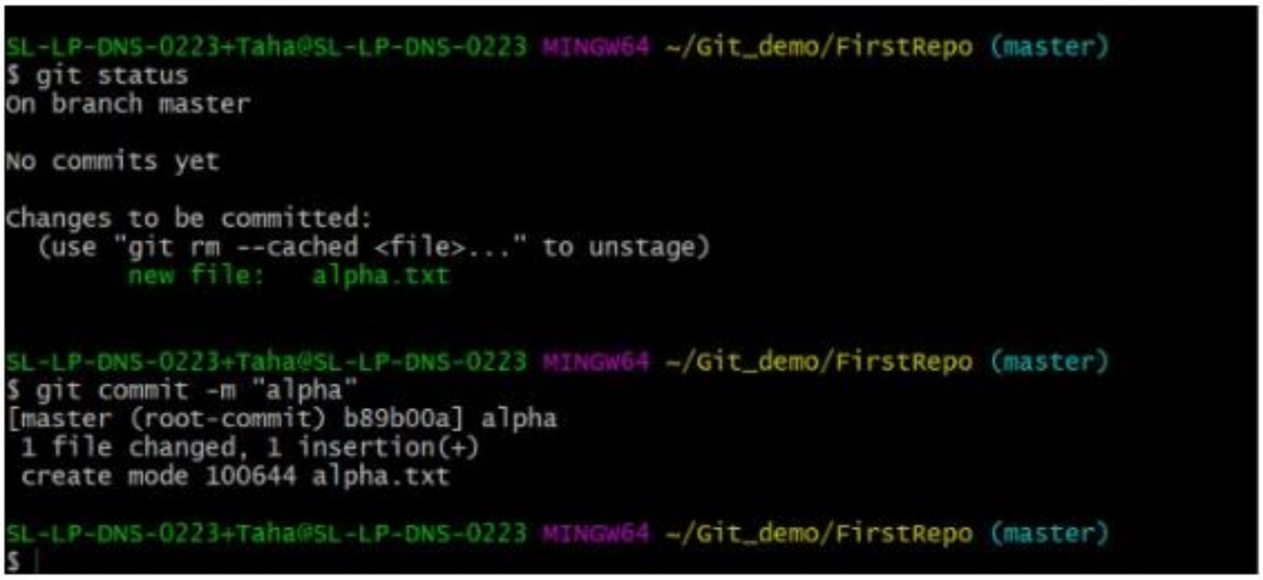


```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        alpha.txt

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”
```

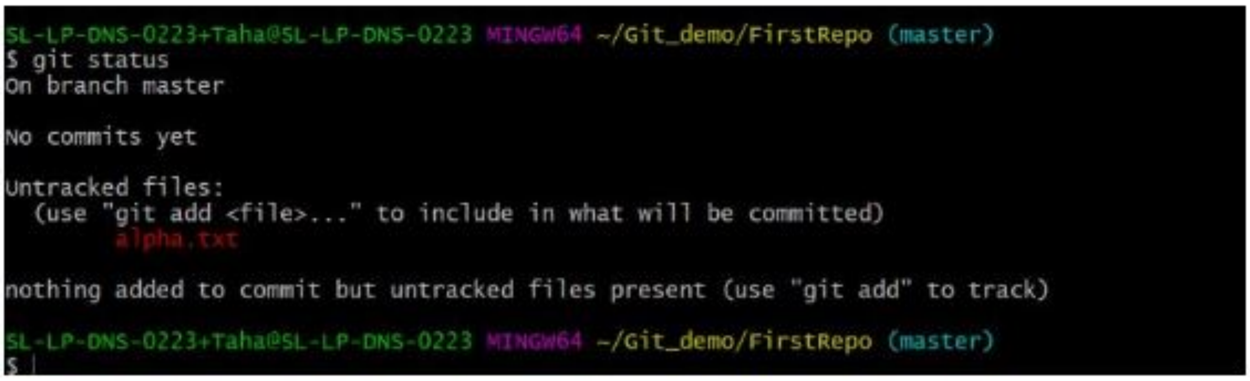


```
SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ git status
On branch master

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)
$ |
```

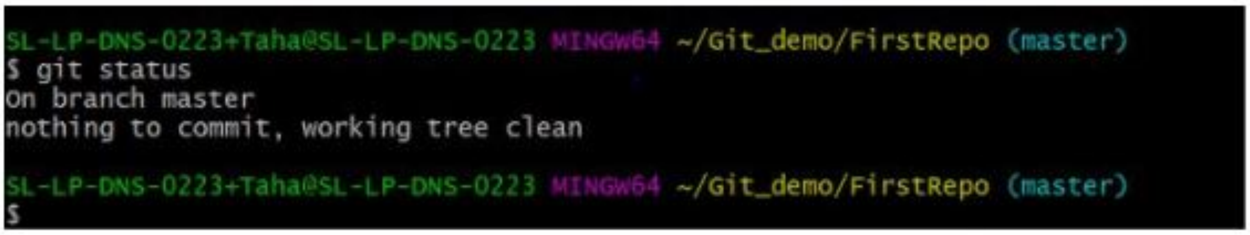
- **git status**

- 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)
$ |
```

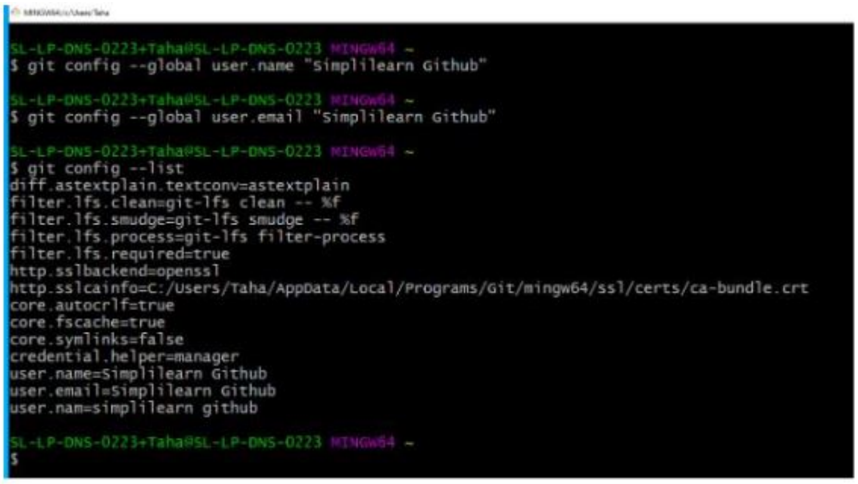


```
SL-LP-DNS-0223+Taha@SL-LP-DNS-0223 MINGW64 ~/Git_demo/FirstRepo (master)
$ git status
On branch master
nothing to commit, working tree clean

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**

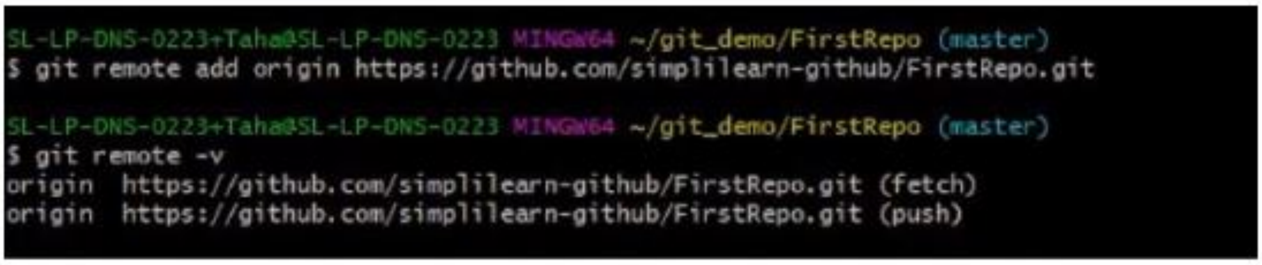
- The git merge command is used to integrate the branches together. The command combines the changes from one branch to another branch.
- It is used to merge the changes in the staging branch to the stable branch.

```
git merge <branch_name>
```

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.

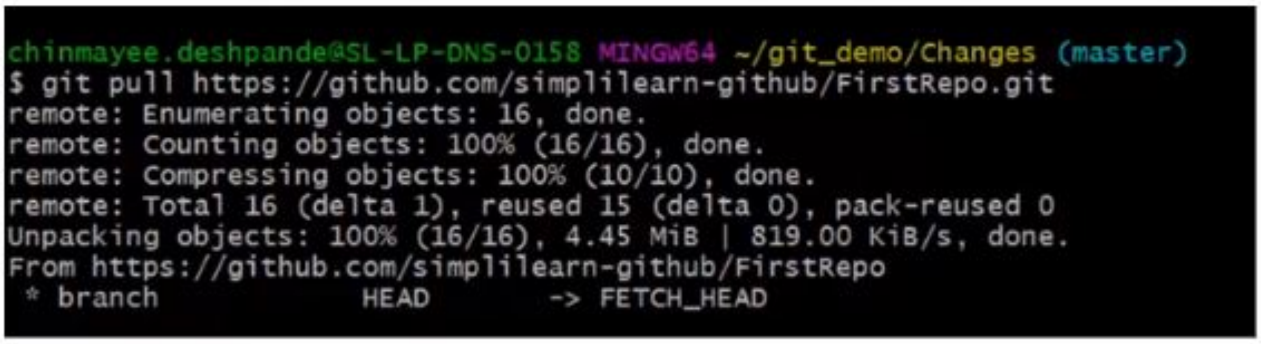
```
git clone <remote_URL>
```

- **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.

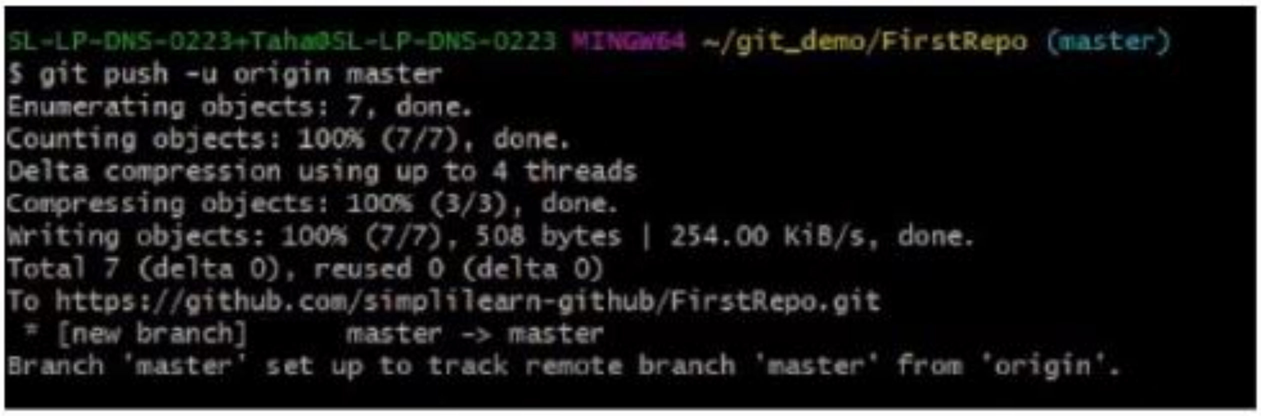
```
git pull <remote URL>
```



```
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

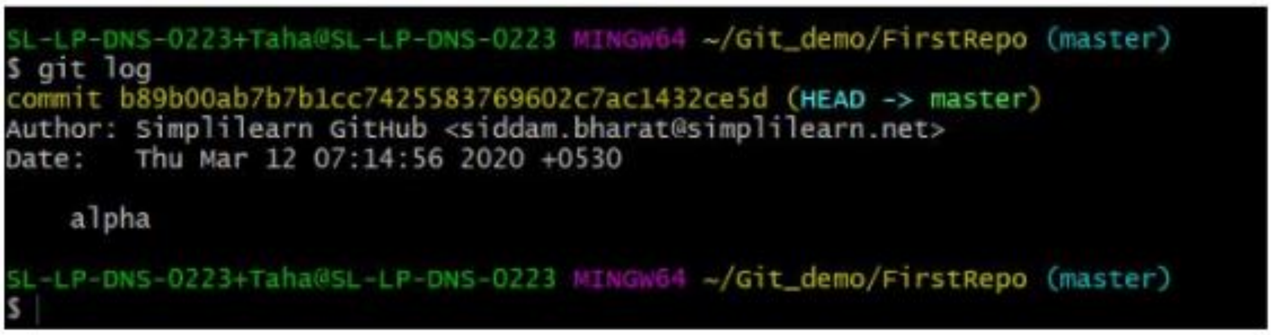
- **git stash**
- The git stash command takes your modified tracked files and saves it on a pile of incomplete changes that you can reapply at any time. To go back to work, you can use the stash pop.
- The git stash command will help a developer switch branches to work on something else without committing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