Git + GitHub Tutorial

Provided by

Muhammad Farhan

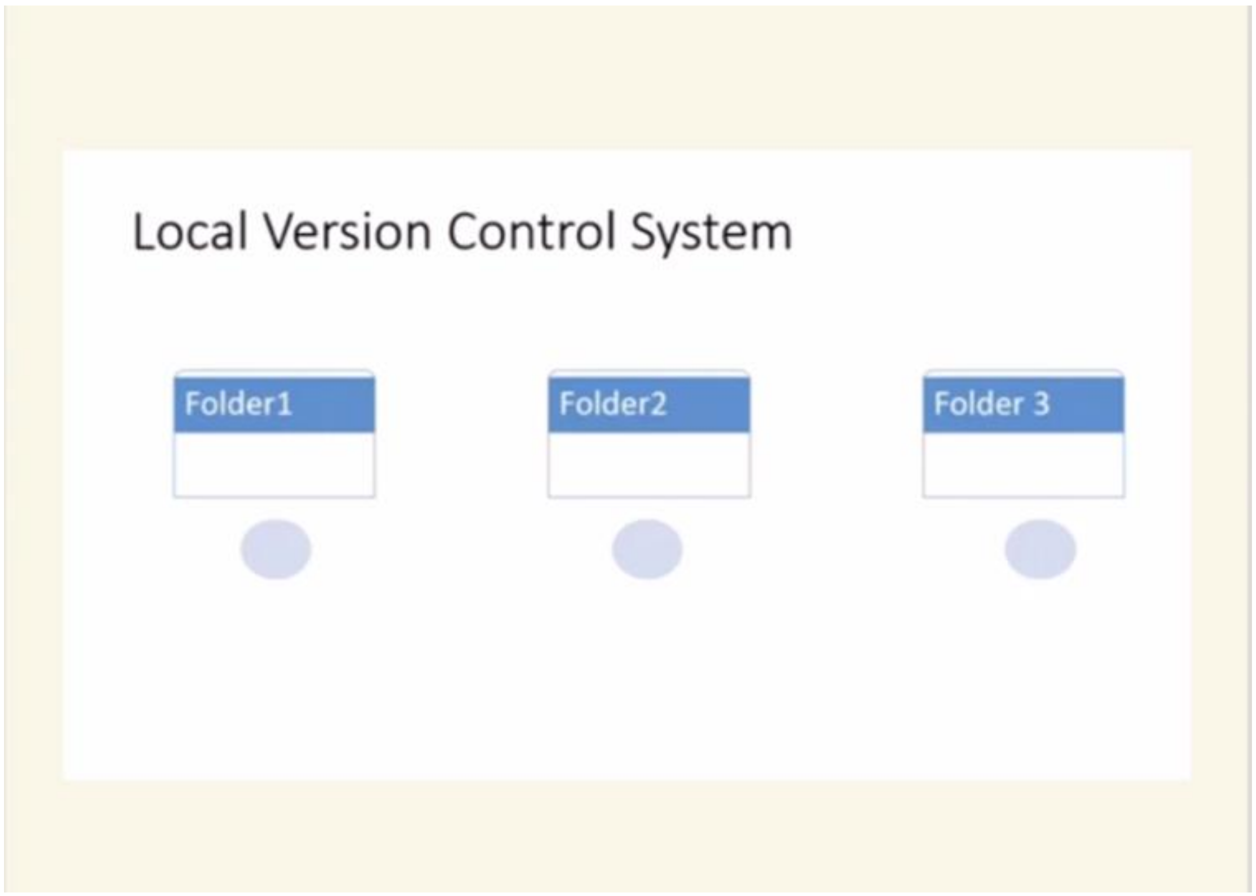
What is Git?

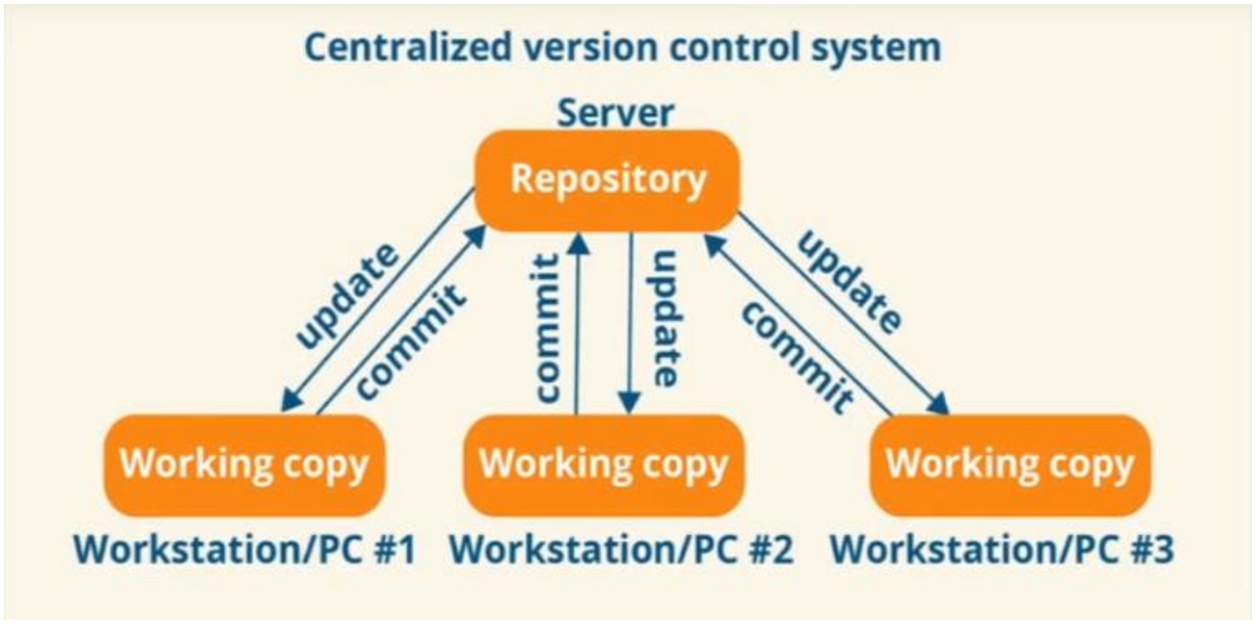
Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

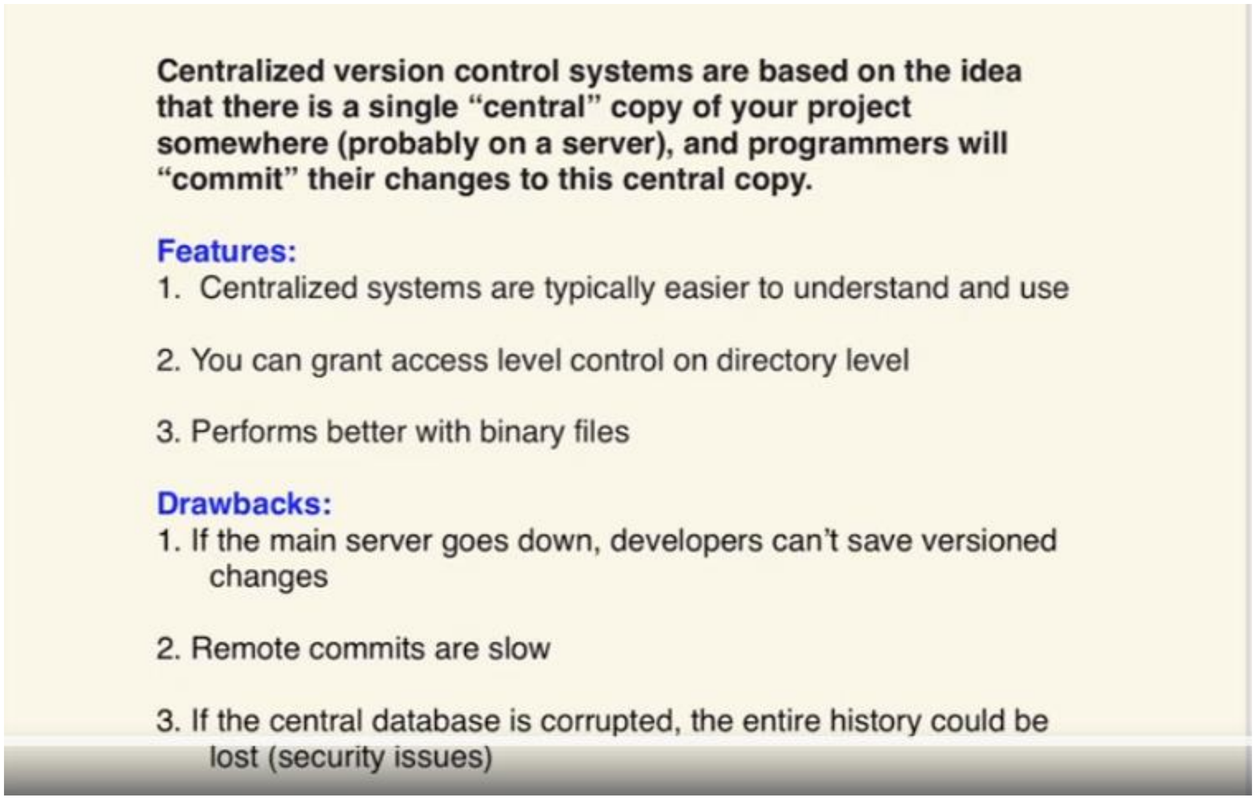
It is used for:

- Git is a version control system.
- Git helps you keep track of code changes.
- Git is used to collaborate on code.

Types of Git:







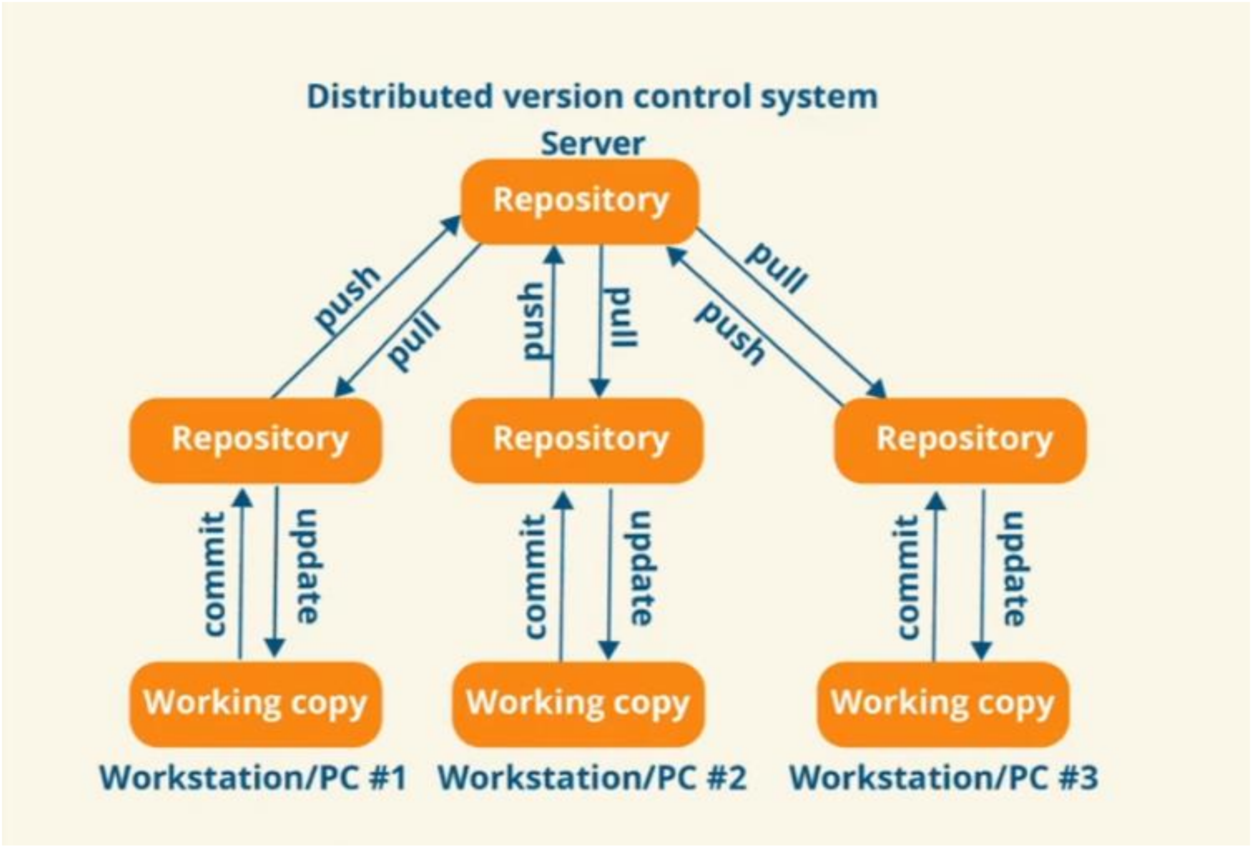
Centralized version control systems are based on the idea that there is a single "central" copy of your project somewhere (probably on a server), and programmers will "commit" their changes to this central copy.

Features:

1. Centralized systems are typically easier to understand and use
2. You can grant access level control on directory level
3. Performs better with binary files

Drawbacks:

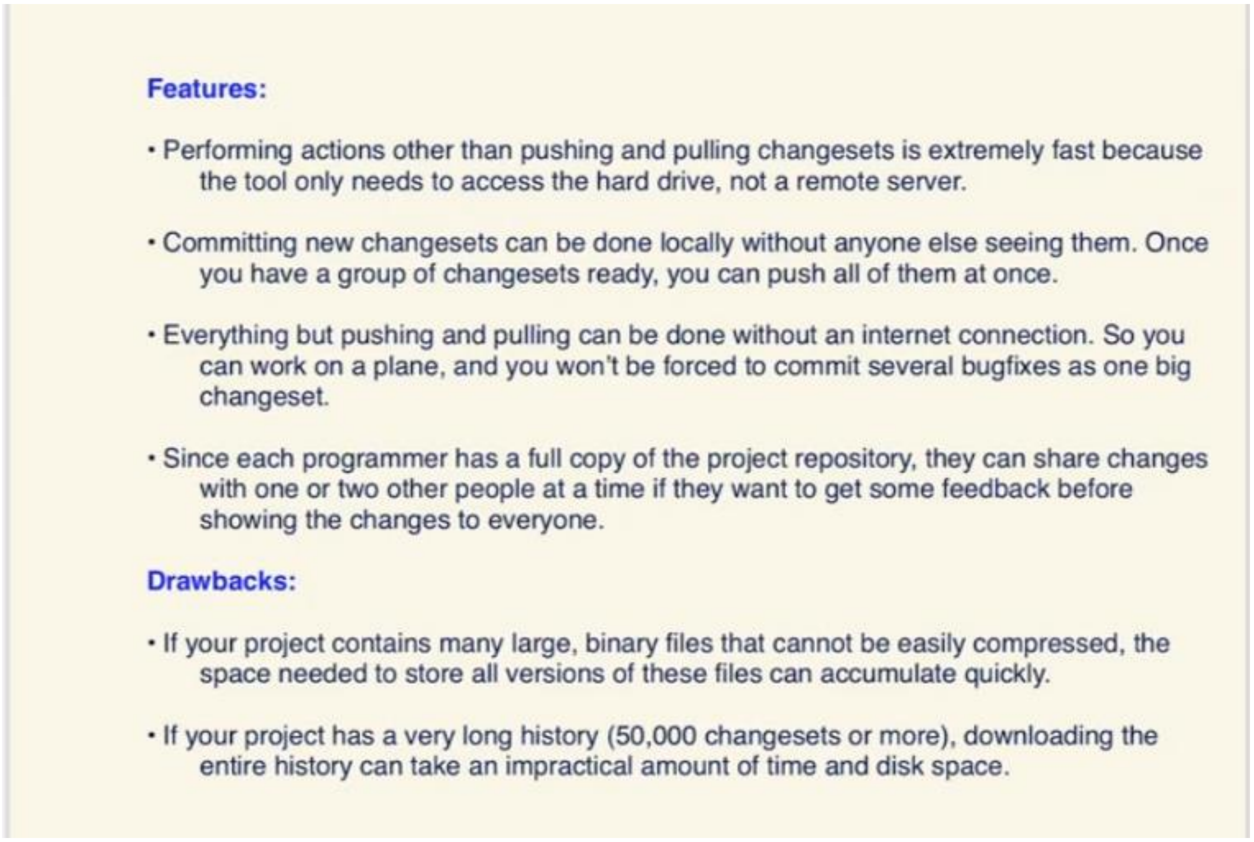
1. If the main server goes down, developers can't save versioned changes
2. Remote commits are slow
3. If the central database is corrupted, the entire history could be lost (security issues)



In distributed version control, every developer "clones" a copy of a repository and has the full history of the project on their own hard drive. This copy (or "clone") has all of the metadata of the original.

Pulling:
The act of getting new changes from a repository is usually called "pulling,"

Pushing:
The act of moving your own changes to a repository is called "pushing"



Features:

- Performing actions other than pushing and pulling changesets is extremely fast because the tool only needs to access the hard drive, not a remote server.
- Committing new changesets can be done locally without anyone else seeing them. Once you have a group of changesets ready, you can push all of them at once.
- Everything but pushing and pulling can be done without an internet connection. So you can work on a plane, and you won't be forced to commit several bugfixes as one big changeset.
- Since each programmer has a full copy of the project repository, they can share changes with one or two other people at a time if they want to get some feedback before showing the changes to everyone.

Drawbacks:

- If your project contains many large, binary files that cannot be easily compressed, the space needed to store all versions of these files can accumulate quickly.
- If your project has a very long history (50,000 changesets or more), downloading the entire history can take an impractical amount of time and disk space.

What does Git do?

- Manage projects with **Repositories**
- **Clone** a project to work on a local copy
- Control and track changes with **Staging** and **Committing**
- **Branch** and **Merge** to allow for work on different parts and versions of a project.
- **Pull** the latest version of the project to a local copy
- **Push** local updates to the main project

Working with Git?

- Initialize Git on a folder, making it a **Repository**
- Git now creates a hidden folder to keep track of changes in that folder
- When a file is changed, added or deleted, it is considered **modified**
- You select the modified files you want to **Stage**

- The **Staged** files are **Committed**, which prompts Git to store a **permanent** snapshot of the files
- Git allows you to see the full history of every commit.
- You can revert back to any previous commit.
- Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

Why Git?

- Over 70% of developers use Git!
- Developers can work together from anywhere in the world.
- Developers can see the full history of the project.
- Developers can revert to earlier versions of a project.

What is GitHub?

- Git is not the same as GitHub.
- GitHub makes tools that use Git.
- GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.
- In this tutorial, we will focus on using Git with GitHub.

Git Install

You can download Git for free from the following website: https://www.git-scm.com/

Create GitHub Account:

Create an account on GitHub by simple steps.

Using Git with Command Line:

To start using Git, we are first going to open up our Command shell.

The first thing we need to do, is to check if Git is properly installed:

Example:

```
git  --version
```

Configure Git:

Now let Git know who you are. This is important for version control systems, as each Git commit uses this information.

Example:

```
git config --global user.name "muhammadfarhan"
git config --global user.email "mohammadfarhan44500@gmail.com"
```

Note: Use global to set the username and e-mail for **every repository** on your computer.

If you want to set the username/e-mail for just the current repo, you can remove global

Initialize Git:

```
git init
```

```
Initialized empty Git repository in /Users/user/myproject/.git/
```

Git New Files:

```
ls
index.html
```

ls will **list** the files in the directory. We can see that index.html is there.

Git Status:

```
git status
```

Then we check the Git status and see if it is a part of our repo:

Now Git is **aware** of the file, but has not **added** it to our repository!

Files in your Git repository folder can be in one of 2 states:

- Tracked - files that Git knows about and are added to the repository
- Untracked - files that are in your working directory, but not added to the repository

When you first add files to an empty repository, they are all untracked. To get Git to track them, you need to stage them, or add them to the staging environment.

Git Staging Environment:

One of the core functions of Git is the concepts of the Staging Environment, and the Commit.

As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.

Staged files are files that are ready to be **committed** to the repository you are working on. You will learn more about commit shortly.

For now, we are done working with index.html. So we can add it to the Staging Environment:

```
git add index.html
```

Now you can use git status.

Now add all files in the current directory to the Staging Environment:

```
git add .
```

Git Commit:

Since we have finished our work, we are ready move from stage to commit for our repo.

Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.

When we commit, we should **always** include a **message**.

By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

Example:

```
git commit -m "First release of Hello World!"
```

Git Log:

To view the history of commits for a repository, you can use the log command:

Example**:**

```
git log
```

Git Help:

If you are having trouble remembering commands or options for commands, you can use Git help.

There are a couple of different ways you can use the help command in command line:

- git *command* -help - See all the available options for the specific command
- git help --all - See all possible commands

Let's go over the different commands.

Git Branch:

Working with Git Branches:

In Git, a branch is a new/separate version of the main repository.

Let's say you have a large project, and you need to update the design on it.

How would that work without and with Git:

- With a new branch called new-design, edit the code directly without impacting the main branch
- EMERGENCY! There is an unrelated error somewhere else in the project that needs to be fixed ASAP!
- Create a new branch from the main project called small-error-fix
- Fix the unrelated error and merge the small-error-fix branch with the main branch
- You go back to the new-design branch, and finish the work there
- Merge the new-design branch with main (getting alerted to the small error fix that you were missing)

Branches allow you to work on different parts of a project without impacting the main branch.

When the work is complete, a branch can be merged with the main project.

You can even switch between branches and work on different projects without them interfering with each other.

Branching in Git is very lightweight and fast!

New Git Branch:

Let add some new features to our index.html page.

We are working in our local repository, and we do not want to disturb or possibly wreck the main project.

So we create a new branch:

Example:

```
git checkout -b hello-world
```

Now we created a new branch called "hello-world-images"

Let's confirm that we have created a new branch:

```
git branch
```

checkout is the command used to check out a branch. Moving us **from** the current branch, **to** the one specified at the end of the command:

```
git checkout hello-world-images
```

You can check git status.

So let's go through what happens here:

- There are changes to our index.html, but the file is not staged for commit
- img_hello_world.jpg is not tracked

So we need to add both files to the Staging Environment for this branch:

```
git add --all
```

```
git commit -m "Added image to Hello World"
```

Now we have a new branch, that is different from the master branch.

Switching Between Branches:

Now let's see just how quick and easy it is to work with different branches, and how well it works.

We are currently on the branch hello-world-images. We added an image to this branch, so let's list the files in the current directory:

```
ls
```

We can see the new file img_hello_world.jpg, and if we open the html file, we can see the code has been altered. All is as it should be.

Now, let's see what happens when we change branch to master

```
git checkout master
```

Emergency Branch:

Now imagine that we are not yet done with hello-world-images, but we need to fix an error on master.

I don't want to mess with master directly, and I do not want to mess with hello-world-images, since it is not done yet.

So we create a new branch to deal with the emergency:

Example:

```
git checkout -b emergency-fix
```

```
git status
```

```
git add index.html
```

Git Branch Merge:

Merge Branches:

We have the emergency fix ready, and so let's merge the master and emergency-fix branches.

First, we need to change to the master branch:

```
git checkout master
```

```
git merge emergency-fix
```

Merge Conflict:

Now we can move over to hello-world-images and keep working. Add another image file (img_hello_git.jpg) and change index.html, so it shows it:

```
git checkout hello-world-images
```

```
git add –all
git commit -m "added new image"
git checkout master
git merge hello-world-images
git status
```

```
git commit -m "merged with hello-world-images after fixing conflicts"
git branch -d hello-world-images
```

```
git restore filename
```

How to Push project

For new or first time

1. Create Repository on GitHub
2. git init
3. git add .
4. git commit –m “message”
5. git branch –M main
6. git remote add origin repo_url
7. git push –u origin main

Existing repo

Git remote add origin repo_url
Git branch –M main
Git push

OR

git add filename
git commit –m “commit”
git push

How to Clone from existing repository ?

git clone repo_url

How to Pull global repository into Local repository?

git pull

Create a branch
Git branch –b feaured_branch

To check how many branches.
Git branch

Convert any branch
Git check master

For merge

Git checkout master
git merge featuredadd