## **<https://www.youtube.com/watch?v=uaeKhfhYE0U&list=PLTynGGPIFFn6GSaUUXyxhMZRtUQjpDbtX&index=1&t=1491s&ab_channel=AnujBhaiya>**

What is Version Control?

When you are working on a simple project, such as a single page html, it is fairly easy to remember the last thing you changed and where the development is headed. But tracking revisions over time, also referred to as version control, quickly becomes more complex when you are working on a large project with multiple files and multiple developers.

You not only want to record changes, but also who made the changes, and when. Managing revisions at this level requires a version control system.

Version Control Systems (VCS) help a software team manage changes to source code over time.

VCS software includes tools for saving the state of a project, viewing the history of changes, and reverting changes.

Developing software without using version control is risky, similar to not having backups.

VCS can also enhance and speed up development. Depending on the version control software used, many developers can work on the same code at the same time.

For example, one developer on the team may be working on a new feature while another developer fixes an unrelated bug, each developer making their changes in several parts of the code base.

VCS even have tools to prevent conflicts when one developer's changes are incompatible with changes made at the same time by another developer.

One of the most popular Version Control Systems (VCS) is Git.

The Git Version Control System

One of the most popular version control systems is Git, a distributed VCS.

Git is a mature, actively maintained, open-source project compatible with many operating systems and IDEs.

Git is different from other version control systems by its way of recording changes. Other systems add changes to a database, where Git records changes as a stream of snapshots.

Git has another advantage - it is distributed.

Rather than having only one single place for the full version history of a project, every developer's working copy of the code is also a repository that can contain the full history of all changes.

Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development.

The name "git" was given by Linus Torvalds when he wrote the very first version. He described the tool as "the stupid content tracker" and the name as (depending on your way):

- random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.

- stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.

- "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.

- "goddamn idiotic truckload of sh\*t": when it breaks

Reference: The readme file of the Git source code.

Initializing a Git Repository

The most effective way to run Git is through a command line prompt. That means you'll be using Terminal in Mac/Linux or Command Prompt or PowerShell in Windows.

The first step to start using Git is to initialize a repository, a location for files and their revision history. Of course, this step can only be taken if you've already installed Git on your computer. There are numerous versions and ways to install Git. Which one you use will depend on your environment. Download the version that suits you from the official website.

If you want a repository in a new folder, then you will need to create a new directory (folder) and then switch to that directory. This is done at the command prompt (often indicated with $ or C:>) with the mkdir command for creating a directory and cd for navigating to a directory.

$ mkdir my\_git\_project

$ cd my\_git\_project

With the commands above, a directory named my\_git\_project is created in the current folder and then the active directory is changed to my\_git\_project.

If you want to create the repository in an already existing folder, just navigate to that directory by typing cd directory\_name at the command line prompt.

Note that you may need to use the full path when navigating to a directory. For example, /c/users/projects or something similar.

Once inside the appropriate folder, use the git init command to turn the directory into an empty Git repository:

To see the files generated by git init, run the following command: $ ls -a

A Git repository will also be set up when you clone an existing one. You will learn more about cloning later.

Review the command line lesson to get familiar with cmd/terminal commands.

Git vs GitHub

Git is a version control system for managing your source code history.

GitHub is a hosting service for your Git repositories.

Tracking Files

In general, files in a repository can have the following statuses:

1) Not tracked

2) Staged

3) Committed

To find out the actual status of files in a repository, the git status command is used.

Let's say we have created a file readme.txt in our repository.

Running git status will have the following output:

As you can see the file readme.txt is untracked.

Untracked It means that the file is not being tracked by Git for changes. We should explicitly say which files it should follow/track.

The git add 'file name' command tells Git to track the file. This step is called staging.

Let's add our readme.txt file:

If you don't want Git to track some specific files, you can "ignore" them.

That is done using the .gitignore file. Anything listed in the .gitignore file is ignored by Git and won't be visible in the repository.

Let's create the .gitignore, as well as a sample password.txt file containing some sensitive data which we don't want to be publicly visible in the repository:

$ touch .gitignore

$ touch password.txt

The touch command is the easiest way to create new, empty files in Unix/Linux.

Now write a sample text inside the password.txt and edit the content of .gitignore adding the file name ("password.txt") we want to be ignored.

You can add as many file as you want inside your .gitignore file and write comments (#comment) to remember the reason they are ignored.

Cloning and Pushing

After committing the changes, the next step is pushing the local repository to the Git server on a remote location (such as GitHub or Bitbucket).

After creating a remote repository, we have the following options:

1) To download (clone) the repository and start making changes.

2) To initialize a local repository and then connect it with the remote one.

The clone command is used to download a remote repository:

$ git clone <https://www.github.com/user/project_name.git>

If you have already initialized a local repository, you can connect it to the remote one using the following command:

$ git remote add origin <https://www.github.com/user/project_name.git>

After making our local changes and commits, its time to push the changes to the remote repository.

The push command tells Git where to put our commits.

$ git push -u origin master

The name of our remote is origin and the default local branch name is master.

The -u tells Git to remember the parameters, so that next time we can simply run git push and Git will know what to do.

Also, keep in mind that it is not mandatory to push your project to a remote location. You are free to work on your project alone without pushing it anywhere, if you need to.

Pulling

A local repository may have commits pushed by other users, who work on the repository. Get the latest updates on the project, especially when you are not the only one working on the project.

We can check for changes on our GitHub repository and pull down any new changes by running:

$ git pull origin master

We can check what is different from our last commit by using the git diff command.

$ git diff HEAD

We want the diff of our most recent commit, which we can refer to using the HEAD pointer.

Reset and checkout

You can use diff to look at changes within files that have already been staged.

Running git diff --staged will show the changes you just staged.

A stage can also be reset using the reset command:

git reset “file name”

This removes the file from the staged status, meaning that all the changes will still remain in the file.

To reset the file to the latest committed version, the checkout command can be used:

git checkout – ‘file name’

It is good practice to regularly run git diff and reset files you accidentally changed.