**Name:** Rohit Harawade

**Roll Number:** A09

**Batch:** A1

**Id:** TU4F2122009

**Experiment - 2**

**Aim**: To perform version control on websites/software using GIT/GITHUB with push and pull commands.

**Requirement**: PC, GitHub account.

**Lab Objective:**

To be aware of different Version Control tools like GIT, CVS or Mercurial.

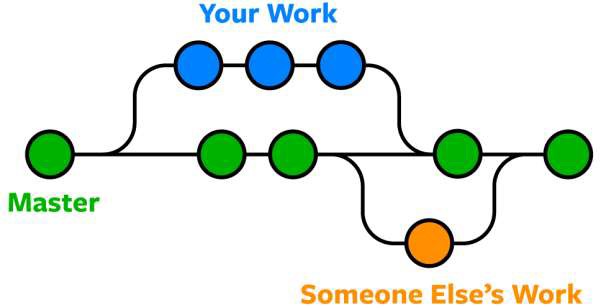
**Theory:**

Git is a distributed version control system. This means that every copy of a Git repository contains the entire history of the project. This makes it easy for multiple people to work on the same project simultaneously, and to collaborate on changes.

Git stores changes to files in a series of commits. Each commit contains a snapshot of the files at a particular point in time, along with a description of the changes that were made. This allows you to track the history of your project and to revert to previous versions if necessary.

Git also supports branches. A branch is a copy of the main line of development. This allows you to work on different features or bug fixes without affecting the main code. When you are finished working on a branch, you can merge it back into the main line of development.

Git is a powerful tool that can be used to manage the development of any project. It is easy to learn and use, and it is free and open source.



**History:**

Git development was started by Torvalds in April 2005 when the proprietary source-control management (SCM) system used for Linux kernel development since 2002, BitKeeper, revoked its free license for Linux development. Torvalds wanted a distributed system that he could use like BitKeeper, but none of the available free systems met his needs. He cited an example of a source-control management system needing 30 seconds to apply a patch and update all associated metadata, and noted that this would not scale to the needs of Linux kernel development, where synchronizing with fellow maintainers could require 250 such actions at once. For his design criterion, he specified that patching should take no more than three seconds, and added three more goals:

* Take the Concurrent Versions System (CVS) as an example of what not to do; if in doubt, make the exact opposite decision.
* Support a distributed, BitKeeper-like workflow.
* Include very strong safeguards against corruption, either accidental or malicious.

These criteria eliminated every version-control system in use at the time, so immediately after the 2.6.12-rc2 Linux kernel development release, Torvalds set out to write his own.

The development of Git began on 3 April 2005. Torvalds announced the project on 6 April and became self-hosting the next day. The first merge of multiple branches took place on 18 April. Torvalds achieved his performance goals; on 29 April, the nascent Git was benchmarked recording patches to the Linux kernel tree at a rate of 6.7 patches per second. On 16 June, Git managed the kernel 2.6.12 release.

**Commands:**

1. git --version

If Git is installed, it should show something like git version X.Y



1. git config --global user.name ""

To let Git know, who we exactly are.



1. git config --global user.email ""

To let Git know, what our default mail address is.



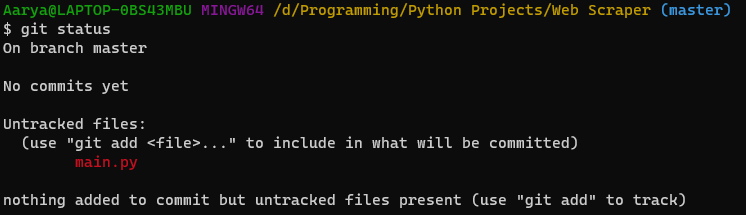
1. git init

You just created your first Git Repository!



1. git status

The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git.



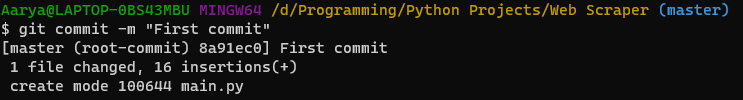
1. git add ""

This command updates the index using the current content found in the working tree, to prepare the content staged for the next commit. It typically adds the current content of existing path as a whole, but with some options it can also be used to add content with only part of the changes made to the working tree files applied, or remove paths that do not exist in the working tree anymore.



1. git commit -m ""

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.



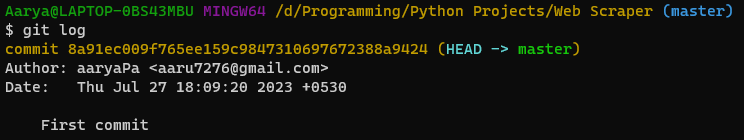
1. git status –short

And check the status of our repository. But this time, we will use the --short option to see the changes in a more compact way:



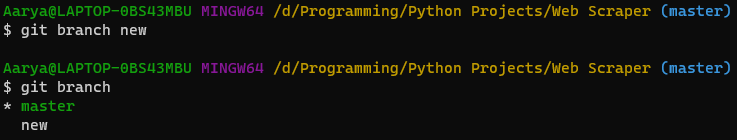
1. git log

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



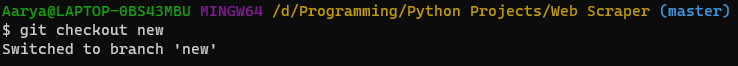
(10) git branch ""

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:



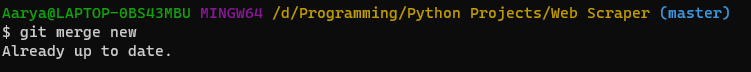
(11) git checkout ""

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:



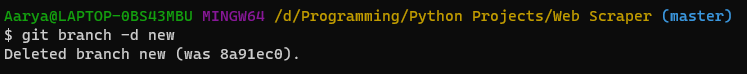
(12) git merge ""

Command is used to merge different branches into the main branch.



(13) git branch -d ""

Command is used to delete a branch.



**Lab Outcome:**

To obtain complete knowledge of the “version control system” to effectively track changes augmented with Git and GitHub.

**Conclusion**:

Thus, we have understood about all the commands in Git.