**Version Control Systems**

A version control system (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

**Following are the goals of a Version Control System.**

* Allow developers to work simultaneously.
* Do not overwrite each other’s changes.
* Maintain history of every version of everything.

There are mainly 2 types of VCS-

1. Centralised (example- SVN)
2. Distributed/Decentralised (example- GIT)

**Terminology**

* **Repository:** A repository is the heart of any version control system. It is the central place where developers store all their work. Repository not only stores files but also the history. Repository is accessed over a network, acting as a server and version control tool acting as a client. Clients can connect to the repository, and then they can store/retrieve their changes to/from repository. By storing changes, a client makes these changes available to other people and by retrieving changes, a client takes other people's changes as a working copy.
* **Trunk:** The trunk is a directory where all the main development happens and is usually checked out by developers to work on the project.
* **Tags** : The tags directory is used to store named snapshots of the project. Tag operation allows to give descriptive and memorable names to specific version in the repository.
* For example, LAST\_STABLE\_CODE\_BEFORE\_EMAIL\_SUPPORT is more memorable than
* Repository UUID: 7ceef8cb-3799-40dd-a067-c216ec2e5247 and
* Revision: 13
* **Branches:** Branch operation is used to create another line of development. It is useful when you want your development process to fork off into two different directions. For example, when you release version 5.0, you might want to create a branch so that development of 6.0 features can be kept separate from 5.0 bug-fixes.
* **Working copy:** Working copy is a snapshot of the repository. The repository is shared by all the teams, but people do not modify it directly. Instead each developer checks out the working copy. The working copy is a private workplace where developers can do their work remaining isolated from the rest of the team.
* **Commit changes:** Commit is a process of storing changes from private workplace to central server. After commit, changes are made available to all the team. Other developers can retrieve these changes by updating their working copy. Commit is an atomic operation. Either the whole commit succeeds or is rolled back. Users never see half finished commit.
* **Staging Area/Index:** The working directory is the place where files are checked out. In other CVCS, developers generally make modifications and commit their changes directly to the repository. But Git uses a different strategy. Git doesn’t track each and every modified file. Whenever you do commit an operation, Git looks for the files present in the staging area. Only those files present in the staging area are considered for commit and not all the modified files.
* **Trees:** Tree is an object, which represents a directory. It holds blobs as well as other sub-directories. A tree is a binary file that stores references to blobs and trees which are also named as **SHA1** hash of the tree object.
* **Clone:** Clone operation creates the instance of the repository. Clone operation not only checks out the working copy, but it also mirrors the complete repository. Users can perform many operations with this local repository. The only time networking gets involved is when the repository instances are being synchronized.
* **Pull:** Pull operation copies the changes from a remote repository instance to a local one. The pull operation is used for synchronization between two repository instances. This is same as the update operation in Subversion.
* **Push:** Push operation copies changes from a local repository instance to a remote one. This is used to store the changes permanently into the Git repository. This is same as the commit operation in Subversion.
* **HEAD:** HEAD is a pointer, which always points to the latest commit in the branch. Whenever you make a commit, HEAD is updated with the latest commit. The heads of the branches are stored in **.git/refs/heads/** directory.
* **Revision:** Revision represents the version of the source code. Revisions in Git are represented by commits. These commits are identified by **SHA1** secure hashes.
* **URL:** URL represents the location of the Git repository. Git URL is stored in config file. **(cat .git/config)**

**Difference between CVCS and DVCS**

* In DVCS, you have a local copy of the repository (called as a staging index) where you can commit, whereas in CVCS, you can directly commit to the main repository.
* In CVCS follows a two tree architecture i.e., local workspace and repository whereas DVCS follows a three tree architecture i.e., local workspace, staging index and repository.
* Git creates one .git directory inside a repository whereas Svn creates a .svn directory inside every folder
* UI of SVN is much better than UI of Git.

**Installation of Git -** [**https://git-scm.com/downloads**](https://git-scm.com/downloads)

**Customise Git Environment:**

Git provides the git config tool, which allows you to set configuration variables. Git stores all global configurations in .gitconfig file, which is located in your home directory.To set these configuration values as global, add the **--global** option, and if you omit **--global** option

**Setting User Name:**

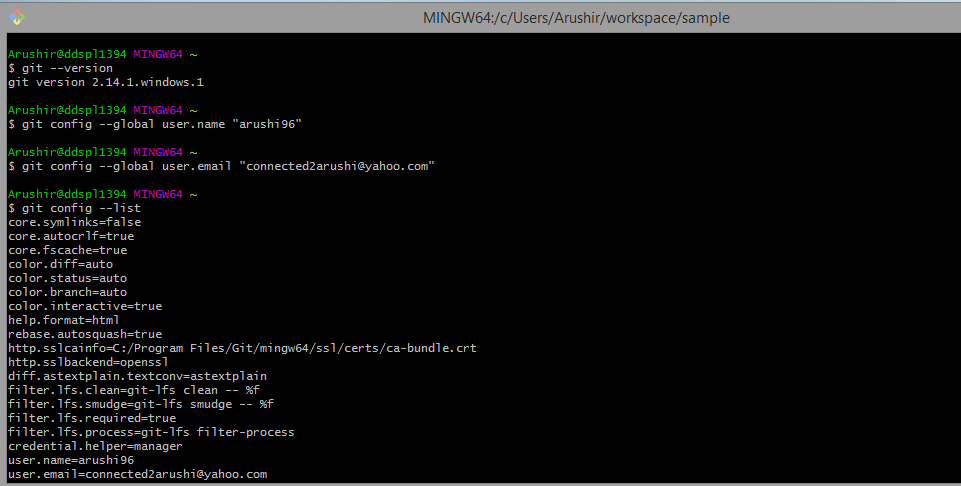
[jerry@CentOS project]$ git config --global user.name "Jerry Mouse"

**Setting User Email:**

[jerry@CentOS project]$ git config --global user.email "jerry@tutorialspoint.com"

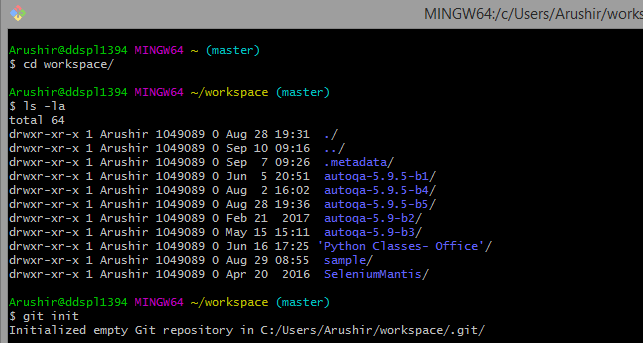
**Listing Git Settings:**

[jerry@CentOS ~]$ git config --list

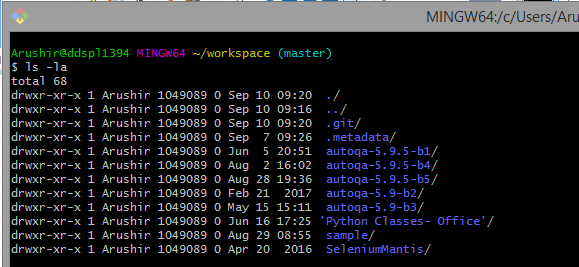
****

Executing git init creates a .git subdirectory in the current working directory, which contains all of the necessary Git metadata for the new repository. This metadata includes subdirectories for objects, refs, and template files. A HEAD file is also created which points to the currently checked out commit

Git init command helps you in tracking the project. Just go inside the repository and type git init. The project will be tracked-



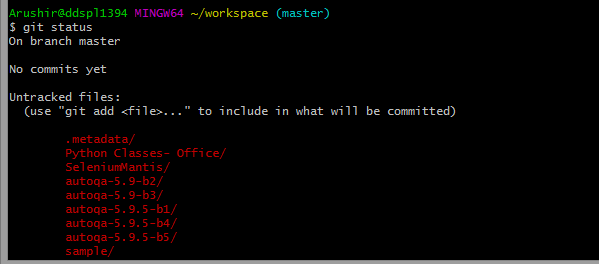
Here, you can see a .git repository gets created.



So, if you want to remove any git directory and stop tracking of your project then, rm -rf .git

Before running git commit command, always run git status command. It helps you to see the untracked files

git status command-



So, if there are some files which you want to ignore and not commit, you can use the gitignore command

touch .gitignore

And, you can mention all the file names here, like- \*.pyc

