Version Control System

Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

There are many version control system used by companies for managing their versions of code like Subversion, Git, Perforce etc. Subversion and Git are open source version system but Perforce is a paid tool used by many MNC companies.

We will be discussing about GIT and how it help developers to work simultaneously and maintain a history of every version.

Before starting GIT we will have a basic difference between Subversion (SVN) and GIT.

SVN is a centralized version control system (CVCS) which uses a central server to store all the files. But the major drawback of CVCS is its single point of failure, i.e. failure of central server. Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all. And even in a worst case, if the disk of the central server gets corrupted and proper backup has not been taken, then you will lose the entire history of the project.

Here **GIT** comes in to picture.

GIT is a distributed version control system (DVCS) which does not rely on the central server for its operations. If the sever goes down, then the repository from any client can be copied back to the server to restore it. Every checkout is a full backup of the repository.

DVCS Terminologies:

Local/Working Repository:

Developers work on their project in their local repository, all the code development is done in their own repository. In that case every developer has its own local repository.

Staging Area:

The Staging area is the place where developer checkout their files .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.

Basic workflow of Git.

Step 1: You modify a file from the working directory.
Step 2: You add these files to the staging area.
Step 3: You perform commit operation that moves the files from the staging area. After push operation, it stores the changes permanently to the Git Repository.

Suppose you modified two files namely “sort.c” and “search.c” and you want two different commits for each operation. You can add one file in the staging area and do commit. After the first commit, repeat the same procedure for another file.

ENVIRONMENT SETUP

Before you can use Git, you have to install and do some basic configuration Changes . Below are the steps to install Git client on Ubuntu and Centos Linux.

Installation of Git Client

On Ubuntu Linux
Run the following command:
sudo apt-get install git-core

Once installed check the Git setup from following command:

git --version

On Centos Linux
yum -y install git-core

Customize 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.

Setting username

git config –global user.name "Jerry Mouse"

Setting email id

git config –global user.email jerry@tutorialspoint.com

Once username and email setting are done, list Git Settings

git config –list

Create Git user and Group

 # add new group
groupadd dev
 # add new user
useradd –G dev –d /home/gituser gituser
#change passwd
passwd gituser

Create a Repository

mkdir gitproject
chmod –R 775 gitproject/
cd gitproject/
Initialize the git project from the following command
git init
Once done a .git hidden directory would be created in the folder. Check it using ls –a

Generate Public/Private RSA Key Pair

Before pushing your code we have to generate a RAS key pair for authentication purpose.

Open a terminal and enter the following command and just press enter for each input. After successful completion, it will create .ssh directory inside the home directory.

tom@CentOS ~]$ pwd
/home/tom
[tom@CentOS ~]$ ssh-keygen
The above command will produce the following result.

Generating public/private rsa key pair.
Enter file in which to save the key (/home/tom/.ssh/id_rsa): Press Enter
Only
Created directory '/home/tom/.ssh'.
Enter passphrase (empty for no passphrase): ----------> Press Enter
Only
Enter same passphrase again: ----------> Press Enter
Only
Your identification has been saved in /home/tom/.ssh/id_rsa.

Your public key has been saved in /home/tom/.ssh/id_rsa.pub.
The key fingerprint is:
df:93:8c:a1:b8:b7:67:69:3a:1f:65:e8:0e:e9:25:a1 tom@CentOS

ssh-keygen has generated two keys, first one is private (i.e., id_rsa) and the
second one is public (i.e., id_rsa.pub).

We will use the public key in our Github so use it as following:
cat /home/tom/.ssh/id_rsa.pub.

copy the key from here and paste it in github
Login to github account -> Go to your Profile->Edit Profile->SSH and GPG keys->New SSH key
And paste the copied key in Key box.

Push Changes to the Repository

Get back to your repository created:
cd gitproject/
In your gitproject we would have our project files which have to be pushed to github repository.
[tom@CentOS tom_repo]$ git add .

[tom@CentOS tom_repo]$ git commit –m “Initial code committed”

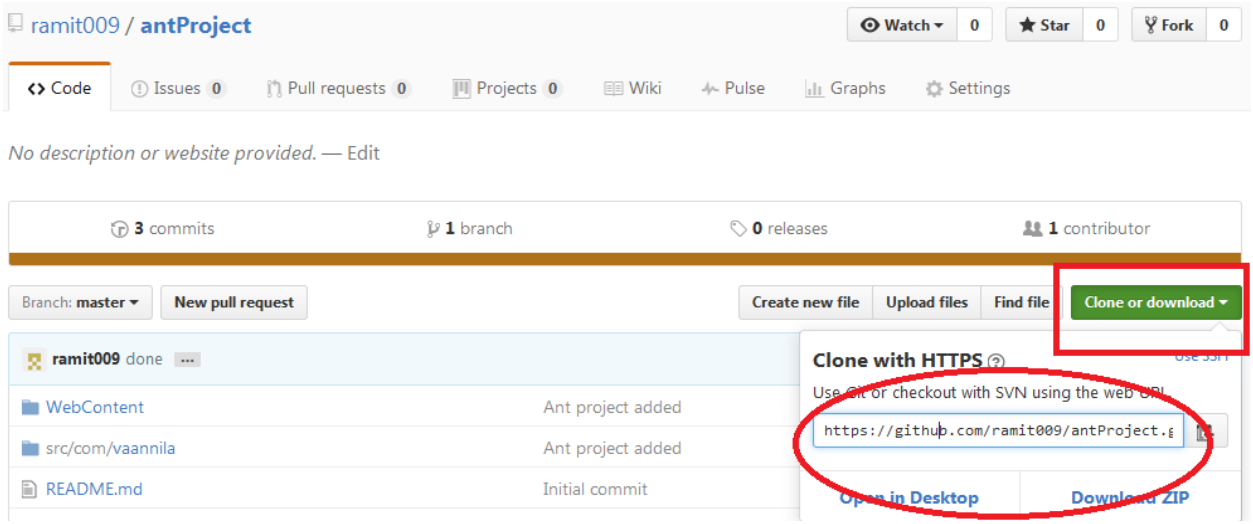
The above command will produce the following result.
[master (root-commit) 19ae206] Initial commit
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 README

After committing to check the status of your code check through the status command.
[tom@CentOS tom_repo]$ git status

Tom committed his changes to the local repository. Now, it’s time to push the
changes to the remote repository. But before that, we have to add the
repository as a remote, this is a one-time operation. After this, he can safely push the changes to the remote repository.

[tom@CentOS tom_repo]$ git remote add origin <github url>
Here origin is the name of the remote which could be modified according to the user and github url is taken from your project folder in github as below:

1. Click on your project and then on the right hand side click on Clone or download as below :



So your url becomes as follow:

[tom@CentOS tom_repo]$ git remote add origin https://github.com/ramit009/antProject.git

Next and the last step we have to push the code to our github repository as :

[tom@CentOS tom_repo]$ git push origin master

Where master is the name of the default branch.

The above command will produce the following result.
Counting objects: 3, done.
Writing objects: 100% (3/3), 242 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
* [new branch]
master –> master
Now, the changes are successfully committed to the remote repository.

CLONE OPERATION

Now we have a repository on the Git server and Tom also pushed his first version. Now, Let's say Jerry another user can view Tom changes. The Clone operation creates an instance of the remote repository.

With clone operation Jerry can make a copy of the code of Tom at his local repository and can push his changes to the same repository.

Jerry creates a new directory in his home directory and performs the clone operation.

```
[jerry@CentOS ~]$ mkdir jerry_repo
[jerry@CentOS ~]$ cd jerry_repo/
[jerry@CentOS jerry_repo]$ git clone  https://github.com/ramit009/antProject.git
```

The above command will produce the following result.

```
Initialized empty Git repository in /home/jerry/jerry_repo/project/.git/
remote: Counting objects: 3, done.
Receiving objects: 100% (3/3), 241 bytes, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
```