# VCS

Version Control System records changes to a file or a set of file over a time so that they can recall specific versions later.

## Local VCS

In this system, all the changes to a file are maintained at the local system either by creating time stamped directories or by creating database which stores the file changes.

## Centralized VCS

It takes the VCS to the next level as it makes collaboration possible. Many people can work on the same set of files together. Here the files are stored on a central server. Developers check-out a file on which they want to work and check-in after their work is done. The problem with this approach is you need to be on-line to commit and if the centralized server is corrupted or destroyed then you will have to go to the backup server which may not have the latest files which may result in loss of data.

## Distributed VCS

Here the programmer working on a file not only checks-out that particular file but whole repository is available on every system.

# Basics

The biggest difference between GIT and other VCS' is that most of the other VCS stores the changes of to the file on the server. Whereas GIT stores the snapshot of the file on server as well as local. And to see the changes it does a local comparison of the snapshots. The advantage of storing complete snapshot instead of just changes will be seen later. The advantage of storing everything on local is that we can work easily when we are off-line and even commits can be made. We need to go on-line just for pushing and pulling.

## Use of checksum

Git makes extensive use of checksum for all the files and directories. This checksum is a 40 character string which is computed on the basis of the content of the file or directory. Hence whenever content changes, the checksum also change and GIT knows about it. In Git database, files are not stored by its name, but by checksum.

## Git generally only adds data

Almost all the actions that you do on Git adds data to the database and thus all the tasks are undo-able in Git.

# States in Git

**Modified**

A file which is checked-out has been modified

**Staged**

The file has been marked in its current version to go into your next commit snapshot.

**Committed**

Commit means the file is safely stored in your local database.

Corresponding to these three states there are three sections in Git.

**Working Directory**

It is a single checkout of one version of the project. These files are placed on to the disk for use and modification.

**Staging Area**

This stores the information about what will go into your next commit. It sometimes referred to as “index”.

**Git Repository**

It stores the metadata and object database for your project. This is what is copied when you clone a repository from another computer.

Hence basic git workflow is as follows:

1. You modify file in your working directory.

2. You stage the files adding their current snapshots to the staged area.

3. You do a commit which takes the files snaphsots from staging area to git repository.

If a particular version of a file is in git directory then its considered committed. If its modified but not staged then its modified and if it is added to the staging area then its staged.

# First time Git Setup

There are 3 files which are used for saving configuration of git. In command prompt these files can be accessed using “git config” tool.

## /etc/gitconfig

It saves configuration for every user and every repository in the system. It can be accessed using

git config –system

## ~/.gitconfig

It is for a specific user and can be accessed using

git config –global

## config file in Git directory

It is for a specific repository.

Priority is highest of config file in in git directory and then for the user and then at last the system wide.

# Adding properties to these files

Git config –global/--system <property-name> <property-value>

property value can be in double quotes if it contains space

--global or –system can be used to tell which file has to be modified

At the time of initial setup name and email should be setup so that the same can be used in every commit.

Git config –global user.name “Romit Chhabra”

git config –global user.email [romit.chhabra@reflexsolution.com](mailto:romit.chhabra@reflexsolution.com)

“git config –system/--global” gives all the options which can be used with these files.

Similarly default editor can be set

git config –global core.editor “Path of exe file of editor in windows or name of editor in linux”

git config –global/--system –list is used to read the content of the corresponding file

# Getting help

There are three ways to get help offline

git help <verb>

git <verb> --help

man git-<verb>

Where verb is a git tool like config

Online help can be taken from

# Initializing a repository in an existing directory

This method is used when you already have a project in place which you want to start track using git.

Go to the directory which you want to track using git and run the following command.

git init

It will create a hidden directory .git which will contain some files. Till now nothing in the project is being tracked. You have to begin tracking files in your project and do an initial commit so that added files are version-controlled. The following commands does it for you.

git add \*.java

git add \*.class

git commit –m “Initial project version”

This adds all the java and class files and commit with the given message.

git add \*

This adds all the files and directories for tracking

You can also give name of the directory after add command. It will add all the files in the directory recursively i.e. including the sub-directories.

# Cloning an existing directory

git clone <url>

It gets you all the data that server has i.e. not just the latest files but all the history and other data as well. The data cloned is so much complete that it can be used as a backup if server is corrupted.

git clone <url> <directory-name>

This command gets the data from the url and puts it into the directory named directory-name instead of the root directory of repository.

Tracked files are the files which were there in the last snapshot or are currently in the staging area. All other files are untracked.

# Checking status

git status

It shows the current branch. List of modified, untracked and staged files.

# Staging modified files

When you modify a file its state is modified. To add this file to the next commit you have to stage it. File is staged or added to the next commit using

git add

command. It takes the file name, directory name, regular expression, \* (for adding everything).

When you modify a file, the following message is displayed in git status

Changes not staged for commit:

(use "git add &lt;file&gt;..." to update what will be committed)

(use "git checkout -- &lt;file&gt;..." to discard changes in working directory)

modified: CONTRIBUTING.md

After adding the CONTRIBUTING.md file to the staging area using git add command, the following message is displayed.

Changes to be committed:

(use "git reset HEAD &lt;file&gt;..." to unstage)

new file: README

modified: CONTRIBUTING.md

Note that earlier file was in the “Changes not staged for commit” section. After running git add \* it goes to “Changes to be committed section”.

Hence git add adds the files for tracking as well as adds the modified files to staging area.

Please note that git adds the snapshot to the staging area on running git add and not the file. In the above example if you make some change again in CONTRIBUTING.md file before committing and run git status then it will show in both staged as well as modified state. Now if you will commit, it will take the snapshot of the file before the last changes i.e. changes only till the file was staged and not after that. To commit the latest changes you have to run git add command again. This shows that git actually works on snapshots.

**Short Git Status**

By default git shows the complete status, but if needed short status can be shown with the following command.

Git status -s

or

git status –short

M README

MM Rakefile

A lib/git.rb

M lib/simplegit.rb

?? LICENSE.txt

The above is an example of status output. The status has 2 columns. Left column shows status in staging area and right column shows in working directory.

M – modified

A – added in staging area

?? - new file not added

README is in modified stated but not yet added in staging area

Rakefile is in modfied state in staging area and modified in working directory as well. It means that it was added to staging area and then modified again.

Lib/git.rb is a new file which has been added to staging area

lib/simplegit.rb is a modfied file in staging area

LICENSE.txt is a new file which is not yet added to staging area

**Ignoring files**