Three conceptual area of Git-Repo:

* Working Tree : What we see in file system. we add delete in working tree.
* Staging Area : Also known as index. It gives control to what may be committed. When our staging area is correct, we make a commit.
* Git-History : this is kept in hidden directory .git. This is equivalent to commit graph.
  + If you copy the .git folder it will have complete git project and history

Creating a git-repo:

Create a standard folder in file system.

Execute init command inside this folder.

$git init

This creates an empty hidden folder .git

Config :

config is administrative task for configuring various parameters. like name, email etc.

Whenever we make a commit, git includes our name, email and time-stamp.

This is important for tracking purpose.

Once we config it, it gets saved in configuration file

$git config --global user.name "Manish Kumar"

$git config --global user.email "Manish.Kumar@softwareag.com"

$git config --list

as flag --global is used so this configuration will work for any repo in this system.

For particular repo if you want to use different name or email then use flag --local

With this configuration, git will never ask it with every commit.

Making a commit:

Create new files/folders in working tree. this new file are initially untracked.

with "git status" command, you can list untracked files.

we stage (adding to Staging Area) untracked files using add command.

* $git add file-name
* $git add -A [Staging all the untracked files]
* $git add . [Staging all the untracked files]

unstage:

To unstage a tracked file use rm command

$git rm --cached file-name

committing the files:

$git commit -m "commit comment"

commits all the files which are present at staging area.

git performs the SHA-1 hash (it used files and meta-data to create this 40 character hash) for every commit. Every commit will have unique hash value

$git log [This command tells us our commit graph]

For every commit, it has 4 info :

* Hash,
* Author,
* Time-stamp and
* Message we provided for commit

Most recent commit-info shows at the top

Showing the difference

$git diff

This command shows the difference between tracked file in working-Tree and Staging-Area.

$git diff --staged

This command shows the difference between Staging-Area and most recent commit.

Removing a file from git-repo

$git rm file-name

This does two things:

- Removes file file-name from working-Tree.

- It also stage this Removal. So file-name is removed from Staging-Area as well.

You can verify this with $git status

Finally commit this removal of file-name

$git commit

Undo in Git :

Undo a working-Tree change: checkout

$git checkout -- file-name

checkout command replaces the file file-name from Staging-Area to working-Tree.

verify this with $git diff

We can't recover the changes made in working-Tree after checkout.

Undo Staging of files: reset

Modify any existing file in git project and add it to make it in Staging-Area.

$git reset HEAD file-name

reset command will restore the file only in Staging-Area (From Git-History) not in Working-Tree.

Working-Tree will have modified version of file. verify this with status and diff commands.

If you want to restore Working-Tree also then, follow-up command is checkout.

Recovering (restoring) a file from prior commit: using commit-hash

$git checkout 9ddb9 -- file-name

the above command will put the file-name back into Staging-Area as well as Working-Tree.

Now you can commit it back to the Git-History.

$git commit -m "commit message"

Note, you can find the commit hash from commit-history using log command as

--$git log -- file-name

.getignore

In this hidden file you can put all the files/folders which need to be ignored while commiting.

\*.exe

\*.class

build/

Conflict:

Conflicts will most likely happen when working in a team environment.

A conflict arises when two separate branches have made edits to the same line in a file, or

when a file has been deleted in one branch but edited in the other.

Merge Conflict:

A merge conflict is an event that occurs when Git is unable to automatically resolve differences in code between two commits.

When all the changes in the code occur on different lines or in different files, then Git automatically resolves conflict.