**Installation (for Windows)**

Git version 2.26.2 latest -20th April 2020

Git installation installs GUI and command line interface

The command line interface can use Linux commands.

Installation from <https://git-scm.com>

**Documentation**

Documentation at <https://git-scm.com/doc>

**Reference for this document**

Useful book on Git <https://git-scm.com/book/en/v2>

**Git benefits:**

Allows to create many branches

It is fast as it is doesn’t use centralized server as operations are performed locally

It is distributed and every clone has complete information with the history (Perforce, CVS, SVN were centralized systems) so we don’t risk losing all the data. Also, many workflows can be created by collaborating with the non-central users.

Git stores snapshots and not differences like other VCS

It has staging area (index) not present in other versioning systems along with a working directory or working tree (after checkout) and the compressed repository to which commits are done. The three states of a file are committed, modified or staged.

**Configuration files in increasing priority order**

1. Location of gitconfig file for storing username and email for Git activities throughout the system for all users

C:\Program Files\Git\etc\gitconfig

1. Location of .gitconfig file for storing username and email for the user logged in

C:\Users\$USER\.gitconfig

1. Location of git config for the particular repo

.git\config

**Gitignore file**

.gitignore file keeps a record of those files which should not be tracked by Git in the working directory. Useful for binaries, libraries, etc. generated during builds which should not be committed to the repo. Please note that it is possible to have multiple .gitignore files for a project

A list of .gitignore files for various requirements is listed below at  
<https://github.com/github/gitignore>

**Some definitions**

*Author*: one who makes a patch

*Committer*: one who applies a patch

*Remote repository*: version of your local repo which is held somewhere else over the internet (or on your PC itself)

*Local repository*: The folder with .git name is the local repo. This is different from working tree.

*Working directory:* Working tree. The actual folder which contains the files that you edit.

*Staging area*: A virtual area where the changes to be committed are held.

*Branches*: A repository has many branches. E.g. ‘master’.

Tags: Tags are used to mark milestones in project development and are usually used for versioning of software. They are of two types: annotated and lightweight tags. The former one is recommended for tagging.

**Information**

Anything that is committed to the repo can always be recovered. Even if it was a commit that amended later or deleted later. Anything lost that was not committed will be lost forever.

If a file is modified, staged and modified again, the commit will be done only for the changes which have been staged and not the latest ones in the working directory

|  |  |
| --- | --- |
| **Configuration** | **Description** |
| git config –-system user.name “Ritesh Udhani”  git config –-global user.name “Ritesh Udhani”  git config –-local user.name “Ritesh Udhani” | Sets username used for all commits.  Updates the file #1, #2 and #3 above. |
| git config –-system user.email “\*\*\*\*@gmail.com”  git config –-global user.email “\*\*\*\*@gmail.com”  git config –-local user.email “\*\*\*\*@gmail.com” | To set email address for all commits.  Updates the file #1, #2 and #3 above. |
| git config --list --show-origin | To see where the configurations are stored. Listed in the order system, global and local with increasing priority |
| git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"  git config --global core.editor emacs | To set up your favorite text editor for git files |
| git help <command>  git <command> --help | To get help on a command |
| **Commands** | **Description** |
| **Getting a git repo** |  |
| git init <project name>  git clone <project path> | To create a local repo afresh or from a remote repository. The .git folder is the repository and the files are a part of working directory (working tree). |
| **Recording changes in a git repo** |  |
| git add <filename> | It means “add precisely this content to the next commit”. This command is used to add a file to start tracking, staging and resolve merge conflicts |
| git restore –-staged <filename>  git reset HEAD <filename> | To unstage a file in the staging area (index) |
| git restore <filename>  git checkout -- <filename> | To discard modifications done in a file |
| git status | To check the status of each file whether it is modified, staged or non-modified. The output has two regions, staged area and non-staged area |
| git status -s | For a simplified output |
| git diff | To know the exact changes made but not staged. Compares staging area with working tree changes |
| git diff –-staged  git diff --cached | Also, to know the exact changes one is about to commit from the staging area of each file |
| git difftool | To use a different tool for comparison like vimdiff, emerge etc |
| git commit –m “message” | Commit changes to the local repo |
| git commit | The commit message is to be entered through the default text editor for git |
| git commit –a –m “message” | To commit directly from working tree to the repository by skipping the staging area. Use carefully as it can be risky |
| git rm <filename> | To remove and existing file from the repository. It also deletes file from the working tree and get the file deletion to the staging area for commiting. |
| git mv <filename1> <filename2> | To rename a file in git |
| **Viewing commit history** |  |
| git log | To see the history of commits in reverse chronological order. This lists commit id, author name, email, date of commit and commit messages |
| Git log –-pretty=oneline | Shows commit information (not file information) in one line. Shows a better readable formatting when many commits are present. |
| Git log –-pretty=short | Shows commit information (not file information) in short. Same as git log, but no commit date |
| Git log –-pretty=full | Shows committer and author information both in commit information (not file details) |
| Git log –-pretty=fuller | Committer and author information and the corresponding dates in commit information (not file details) |
| Git log –-pretty=format:”%h %ar” | To get the commit id, author, committer information, date, commit message in a specific format. Shows commit information (not file information) |
| Git log --relative-date | The time relative to now when the commit was made instead of the absolute date (shows commit information and not file information) |
| Git log --stat | In addition to commit information it displays file information like filename, no of insertions and deletions |
| Git log --shortstat | In addition to commit information it displays file information with stats summary line for file changes |
| Git log –-name-only | In addition to commit information it displays file information with only file name in stat |
| Git log –-name-status | In addition to commit information it displays file information with filename and the action done on that file, like additions, modification or deletion. |
| git log --graph | To view branches and merges information |
| git log –p -2 | Filter out commits. In addition to commit information and file information it displays modification information to show the patch details of the last two commits |
| git log –p -1 <commitid> | Filter out commits. In addition to commit information and file information it displays modification information to show patch details of a specific patch with commit id |
| git log –-author=”<part of author name>” | Filter out commits. To see commits by a specific author |
| git log –-committer=”<part of committer name>” | Filter out commits. To see commits by a specific comitter |
| Git log –since=”2 weeks” (“2 years, 2 minutes, 2 hours or specify a specific date "2008-01-15" etc.) | Filter out commits. By time boundation. |
| Git log –until=”2 weeks” (“2 years, 2 minutes, 2 hours or specify a specific date "2008-01-15" etc.) | Filter out commits. By time boundation. |
| Git log –grep “message” | Filter out commits. Greps message in commit messages and filters the commits |
| Git log <filename/directory name> | Filter out commits. Show only those commits which are related to a specific file or directory |
| Git log --no-merges | Filter out commits. Show only those commits which are not merge commits |
| **Undoing Things** |  |
| Git commit --amend | To amend previous commit by additionally committing staging area contents. It can also help to modify the commit message. The previous commit is deleted and replaced by a new commit. |
| **Working with remotes** |  |
| Git remote | To see the remote servers configured. If a repo is cloned then we should see atleast one remote repo. The default name of the repo server is ‘origin’ from which you cloned. |
| Git remote -v | To see the url of the remote servers |
| git remote add <shortname> <url> | To add a remote server. Shortname is usually ‘origin’. But it can be changed too. |
| Git remote rename <currentname> <newname> | To rename a remote repository |
| git remote remove <remote repo name>  git remote rm <remote repo name> | To remove a remote repo. E.g. remote repot name is ‘origin’. |
| Git fetch <remote repo name> | Merges data from the remote repository to the local repository. No change is made to the working directory. They have to be merged manually |
| Git pull <remote repo name> | It does the job of fetch and additionally tries to merge the remote repo contents to the working directory |
| git push <remote> <branch> | To push your changes upstream to the remote repository branch. E.g. of remote is ‘origin’ and branch is ‘master’. It will work only if your local repo is up to date with the remote else the push request is rejected. Use fetch/pull to keep local repo updated. |
| Git remote show <remote repo name> | To show the relationship between local and remote repositories, i.e., if one pushes the changes from a local branch, the command helps identify the remote repo and the corresponding branch the change will go to. Similarly, vice-versa for git pull. Example ‘remote repo name’ is ‘origin’ |
| **Tagging** |  |
| Git tag | Lists all the tags for a repository. |
| Git tag | grep “pattern\*” | List only those tags which have a pattern |
| Git tag –a <tagname> -m “<commit message>” | Creates annotated tag <tagname> (e.g. v1.0) with commit message and it also records taggers information |
|  |  |
| Git show <tagname> | To show information about a particular annotated tag |