An Introduction to Git

Prerequisites – No prior knowledge of programming needed.

Why to Learn Git –

- Do you wish to work for a development team or collaborative environment?
- How do you manage versioning of a software or application?
- Do you wish the build process to be automated?
- Would you like to reduce the pain of integrations?
- Would you like to work on different activity within same project parallel?
- Do you wish to manage a project?

There are many questions and there's a solution to it – GIT

Companies using GIT –

Official Site Definition –

"Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency."

"Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows."

"Git is foundation of many services like GitHub and GitLab. Git is easy to learn, and has fast performance."

Official Site – https://git-scm.com/

In this presentation, we will walk through with the introduction to Git. In this we will be covering the below topics –

- Installation
- Setup & Configurations
- Introduction to concepts
- Basic Commands
- Wrap up

<u>Installations</u> –

- **Installing on Linux**

RPM-based distribution	**Debian**
sudo dnf install git--all	sudo apt install git --all

- **Installing on Windows**

  Go to the official page i.e. https://git-scm.com/ and click on Download for windows as shown in the fig 1.2 below.

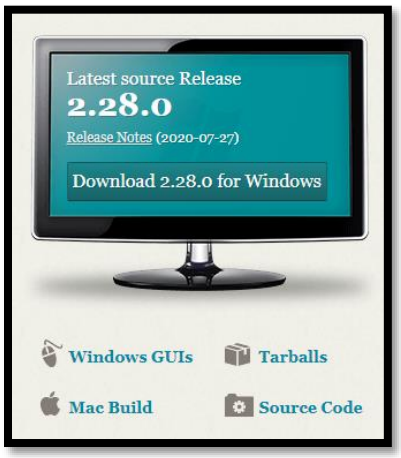
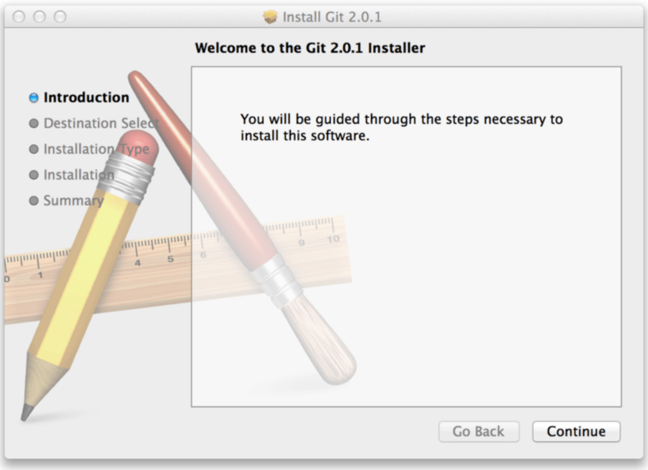


Fig – 1.2

- **Installing on Mac**

  git --version

  [If you don't have it installed already, it will prompt you to install it as shown in Fig 1.3.]



Welcome to the Git 2.0.1 Installer

You will be guided through the steps necessary to install this software.

Fig 1.3

You can also install git using brew

brew install git

https://git-scm.com/download/mac

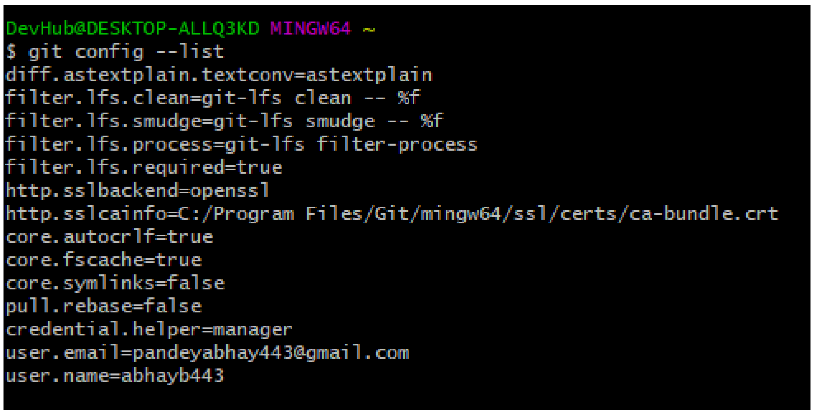
Setup & Configurations – [First-Time Git Setup]

git config --global user.name "abhayb443"

git config --global user.email pandeyabhay443@gmail.com

git config --global init.defaultBranch main

git config --list

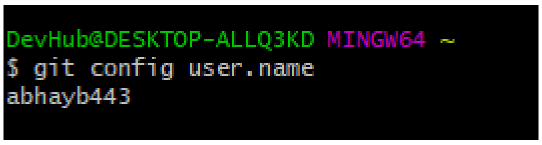


```
DevHub@DESKTOP-ALLQ3KD MINGW64 ~
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
user.email=pandeyabhay443@gmail.com
user.name=abhayb443
```

Fig 1.4

If you want to check specific settings like user's name, you can achieve the same using the below –

git config user.name



```
DevHub@DESKTOP-ALLQ3KD MINGW64 ~
$ git config user.name
abhayb443
```

Fig 1.5

Introduction to concepts

- **Version control system –** Version control systems is the category of software tools that helps record changes to files by keeping a track of modifications done to the code.

- **Repository –** It can be thought as a database of changes. It contains all the edits and historical versions (snapshots) of the project.

- **Git Repository –** A Git repository is the .git/ folder inside a project. This repository tracks all changes made to files in your project, building a history over time.

- **Git Branch –** A branch represents an independent line of development. The git branch command lets you create, list, rename, and delete branches. It doesn't let you switch between branches or put a forked history back together again. For this reason, git branch is tightly integrated with the git checkout and git merge commands.

- **Git Checkout –** The git checkout command is used to switch between branches in a repository. It checks the branches and updates the files in the working directory to match the version already available in that branch, and it forwards the updates to Git to save all new commit in that branch.

- **Git Init –** The git init is one way to start a new project with Git. To start a repository, use either git init or git clone - not both. To initialize a repository, Git creates a hidden directory called .git. That directory stores all of the objects and refs that git uses and creates as a part of your project's history.

- **Git Cloning –** git clone is a Git command line utility which is used to target an existing repository and create a clone, or copy of the target repository. Cloning a local or remote repository. Cloning a bare repository. Using shallow options to partially clone repositories.

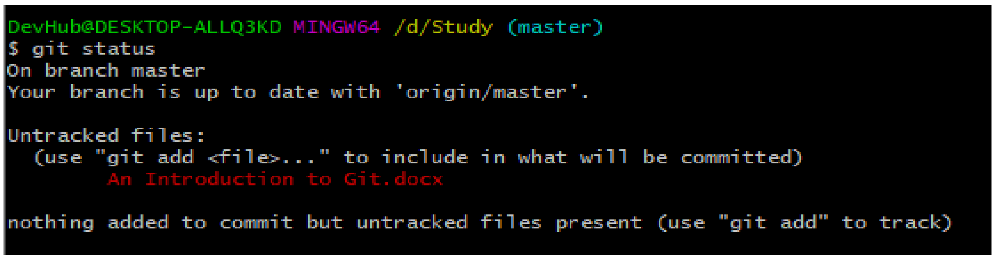
- **Git Commit –** The git commit command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as "safe" versions of a project—Git will never change them unless you explicitly ask it to. These two commands git commit and git add are two of the most frequently used.

- **Git Push –** The git push command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo. It's the counterpart to git fetch, but whereas fetching imports commits to local branches, pushing exports commits to remote branches.

- **Git Fetch –** Download objects and refs from another repository. Fetch branches and/or tags (collectively, "refs") from one or more other repositories, along with the objects necessary to complete their histories. Git fetch can fetch from either a single named repository or URL, or from several repositories at once if <group> is given and there is a remotes <group> entry in the configuration file.

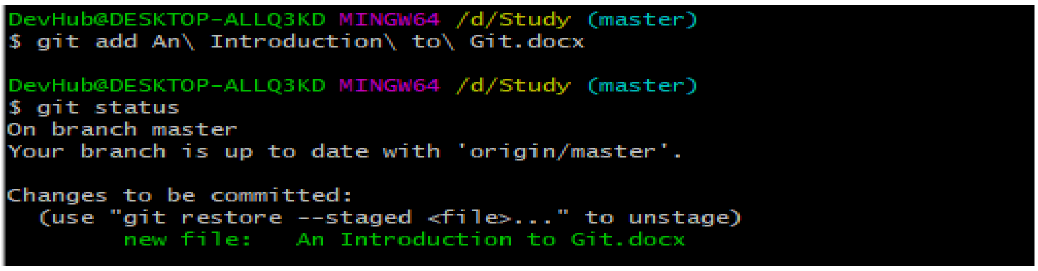
- **Git Status –** It shows the status of the project. If any new files are added, it shows it in untracked files.

- **Unstaged File i.e. Untracked File -** Untracked files are such files that are not previously staged or committed.



```
DevHub@DESKTOP-ALLQ3KD MINGW64 /d/Study (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        An Introduction to Git.docx

nothing added to commit but untracked files present (use "git add" to track)
```

- **Staged File i.e. Tracked File –** Tracked files are such files that are previously staged or committed.



```
DevHub@DESKTOP-ALLQ3KD MINGW64 /d/Study (master)
$ git add An\ Introduction\ to\ Git.docx

DevHub@DESKTOP-ALLQ3KD MINGW64 /d/Study (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   An Introduction to Git.docx
```

Glimpse of few advance commands in Git –

- **Git Stash –** Git stash is used when you want to record the current state of the working directory and the index, but want to go back to a clean working directory. The command saves your local modifications away and reverts the working directory to match the HEAD commit.

- **Git Diff –** Git diff is a multi-use Git command that when executed runs a diff function on Git data sources. These data sources can be commits, branches, files and more. ... The git diff command is often used along with git status and git log to analyse the current state of a git repo.

- **Git log –** The advantage of a version control system is that it records changes. Git log is a utility tool to review and read a history of everything that happens to a repository. Multiple options can be used with a git log to make history more specific. Generally, the git log is a record of commits.

- **Git Merge –** Git merge will combine multiple sequences of commits into one unified history. In the most frequent use cases, git merge is used to combine two branches. ... Once Git finds a common base commit it will create a new "merge commit" that combines the changes of each queued merge commit sequence.

- **Git Rebase –** Rebase is one of two Git utilities that specializes in integrating changes from one branch onto another. The other change integration utility is git merge. Merge is always a forward moving change record. Alternatively, rebase has powerful history rewriting features.

- **Git Fork –** A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

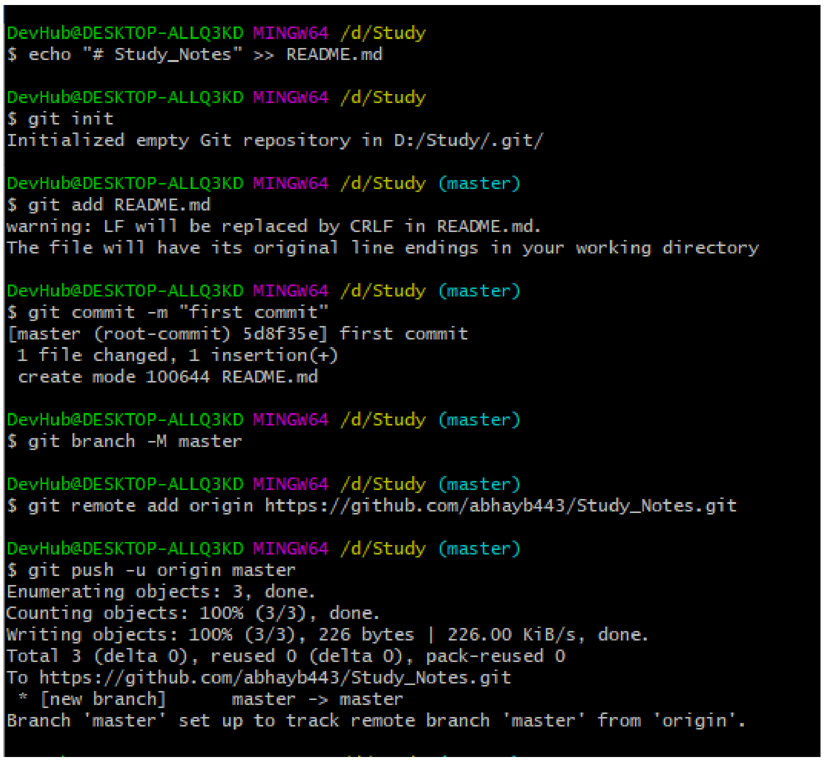
- **Git Ignore –** The git ignore file is used to specify intentionally untracked files that Git should ignore. Sometimes we want to avoid sending the files to Git service like GitHub. So, to avoid it, we need to specify files in Git to ignore.

- **Ignored Files –** Ignored files are such files that are explicitly ignored by git. We have to tell git to ignore such files.

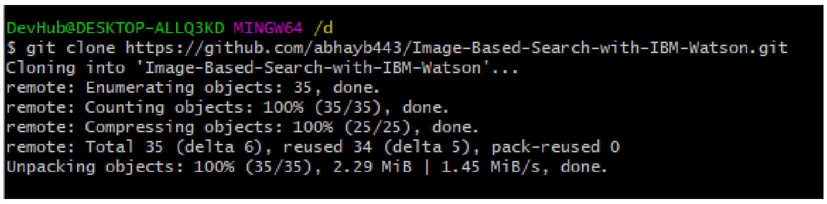
Basic Commands [Most commonly used commands in git] –

git init	[In the project directory that you want to]
git add file	[For adding a single file]
git add file1 file2	[For adding multiple files]
git add filename	[For adding all files in a go]
git commit –m “First Commit”	
git push origin master	
git clone <repo name>	[To clone a repo that has been already created, updated]

Let’s try what we just learnt with an example –



```
DevHub@DESKTOP-ALLQ3KD MINGW64 /d/Study
$ echo "# Study_Notes" >> README.md

DevHub@DESKTOP-ALLQ3KD MINGW64 /d/Study
$ git init
Initialized empty Git repository in D:/Study/.git/

DevHub@DESKTOP-ALLQ3KD MINGW64 /d/Study (master)
$ git add README.md
warning: LF will be replaced by CRLF in README.md.
The file will have its original line endings in your working directory

DevHub@DESKTOP-ALLQ3KD MINGW64 /d/Study (master)
$ git commit -m "first commit"
[master (root-commit) 5d8f35e] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md

DevHub@DESKTOP-ALLQ3KD MINGW64 /d/Study (master)
$ git branch -M master

DevHub@DESKTOP-ALLQ3KD MINGW64 /d/Study (master)
$ git remote add origin https://github.com/abhayb443/Study_Notes.git

DevHub@DESKTOP-ALLQ3KD MINGW64 /d/Study (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 226 bytes | 226.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/abhayb443/Study_Notes.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```



```
DevHub@DESKTOP-ALLQ3KD MINGW64 /d
$ git clone https://github.com/abhayb443/Image-Based-Search-with-IBM-Watson.git
Cloning into 'Image-Based-Search-with-IBM-Watson'...
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 35 (delta 6), reused 34 (delta 5), pack-reused 0
Unpacking objects: 100% (35/35), 2.29 MiB | 1.45 MiB/s, done.
```

Wrap up –

For more details, follow the git atlas Sian cheat sheet –

https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet

https://github.com/abhayb443/Study_Notes

Note – Definitions are referred from the official site mentioned above.