Lesson: Introduction To Git

- **MNNIT COMPUTER CLUB**

March 7, 2018

Audience

- Everyone having a knack for development
- !(Point 1) too

Objective

To make you answer the following questions yourselves:

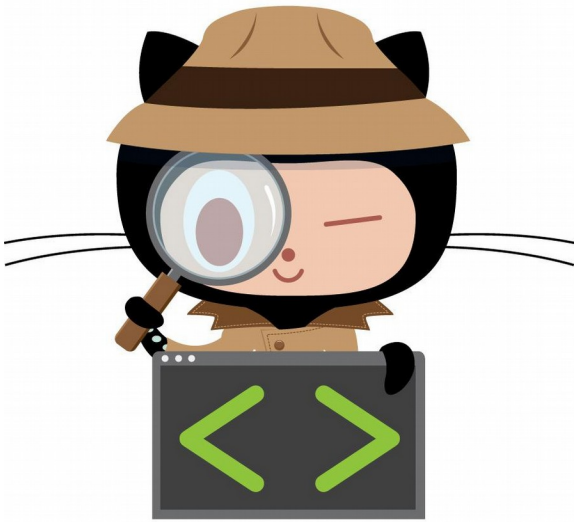
- What is Git?
- Why Git?
- How to use Git?
- Is it worth learning Git?

And finally to turn you into a cool git ninja like the one shown on left.

Reference

- Pro Git Book - by Scott Chacon and Ben Straub
- How to use Git And Github - Udacity Course
- Git started with Github - Udemy Course
- Lots of online tutorials ;-)

What is git ?



According to Oxford English Dictionary git is a noun which means an unpleasant man or a contemptible person.

But what this unpleasant man has to do with computer science?

The story finds it way back to Linus Torvald, creator of Linux and git. Linus often calls himself an egotistical bastard and loves to name his projects after himself. Thus giving us the name git.

What Actually git Is ?

Git is a Distributed Version Control and Source Code Management System with an emphasis on speed.

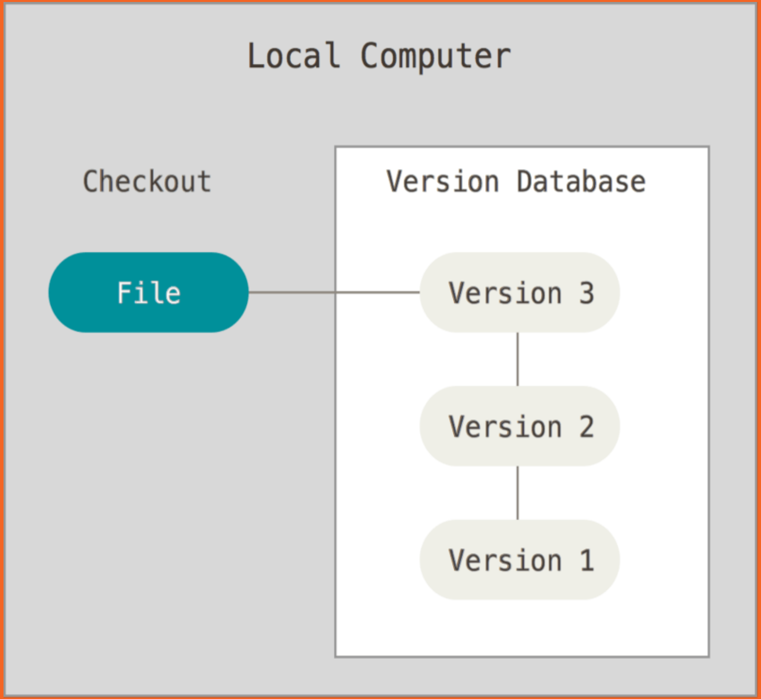
Git was initially designed and developed by Linus Torvalds for Linux kernel development.

Git is a free software distributed under the terms of the GNU General Public License version 2.

Git serves the need to collaborate with developers on other systems.

Version Control System

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

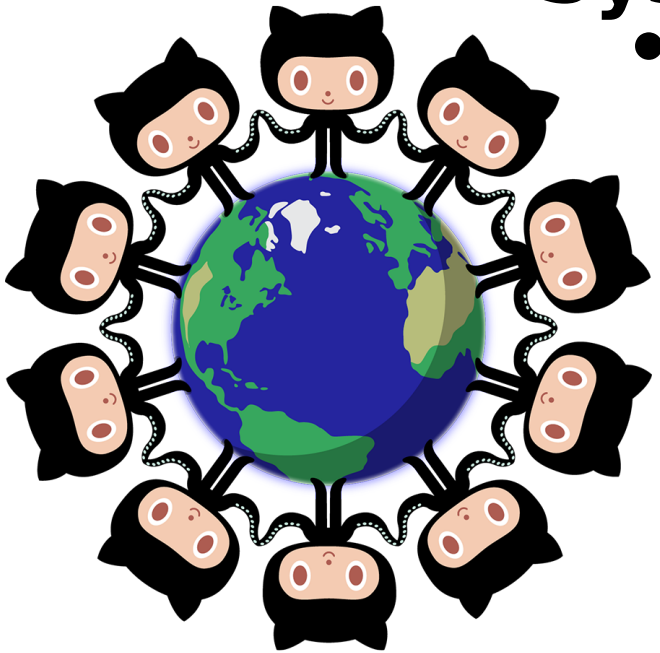


Types Of Version Control System

- **Local version control system :**

Local VCSs had a simple database that kept all the changes to files under revision(version) control. Ex- RCS, SCCS, etc.
Key point : Whenever you have the entire history of the project in a single place, you risk losing everything.

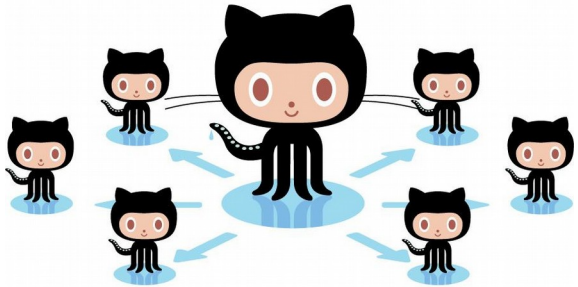
Types of Version Control System



- **Centralized version control system(CVCS)**

They have a single server that contains all the versioned files, and a number of clients that check out files from that central place.
Ex- CVCS, Integrity, etc.
Key Point : Single point of failure

Types of Version Control System



Distributed/Decentralized version control system (DVCS)

DVCS is a form of version control where the complete codebase - including its full history - is mirrored on every developer's computer.
Eg. Git, Mercurial, Bazaar, etc.
Key Point : If any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it.

Why Git?

Because It’s Cool - (00)

Other systems (CVS, Subversion, Perforce, Bazaar, and so on) think of the information they store as a set of files and the changes made to each file over time (this is commonly described as delta-based version control).

Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. Git thinks about its data more like a stream of snapshots.

Because It’s Cool - (01)

- Most operations in Git need only local files and resources to operate — generally no information is needed from another computer on your network.
- CVCS have that network latency overhead.
- This makes Git surprisingly fast.

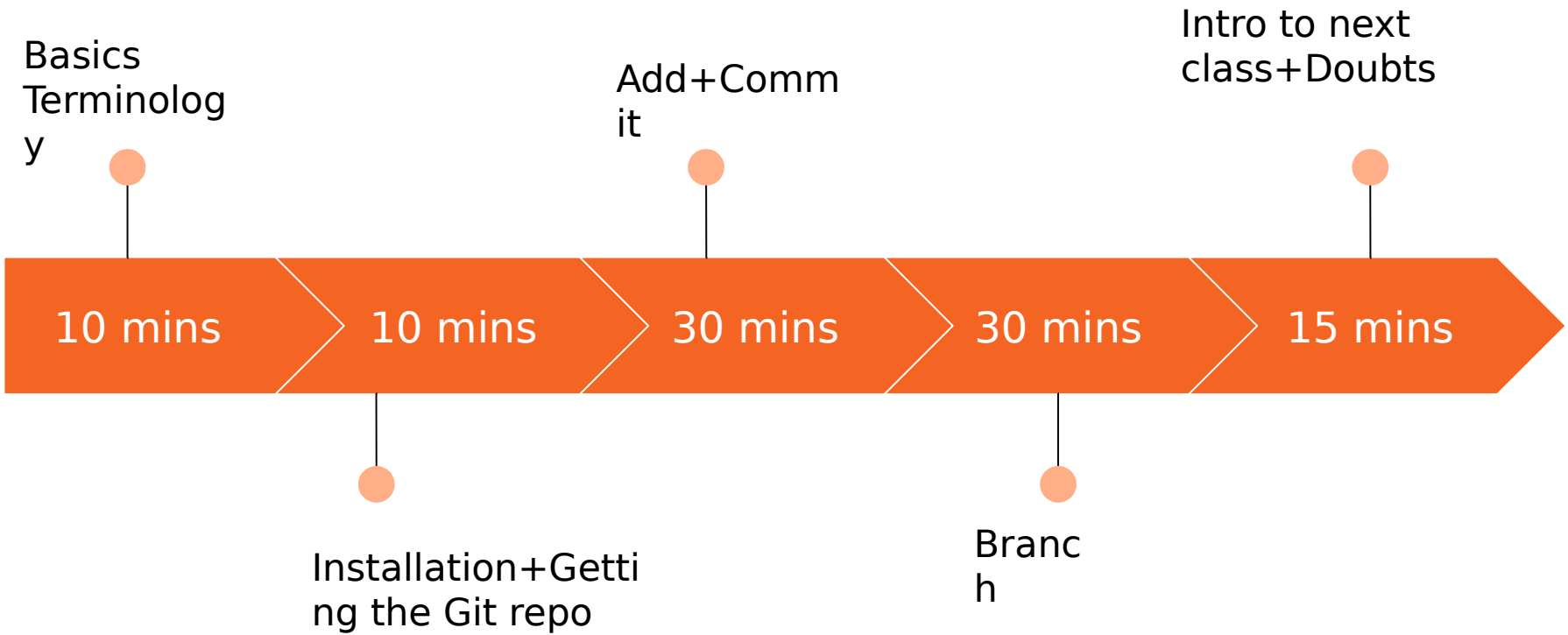
Because It’s Cool - (10)

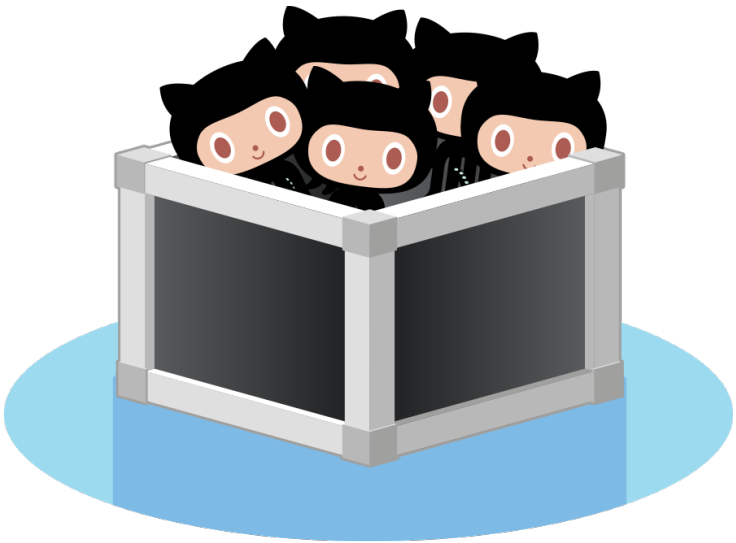
- Everything in Git is check-summed before it is stored and is then referred to by that checksum. This means it’s impossible to change the contents of any file or directory without Git knowing about it.
- Uses 40 character SHA-1 Hash like 24b9da6552252987aa493b52f8696cd6d3b00373

Because It’s Cool - (11)

As with any VCS, you can lose or mess up changes you haven’t committed yet, but after you commit a snapshot into Git, it is very difficult to lose, especially if you regularly push your database to another repository.

How to use Git?

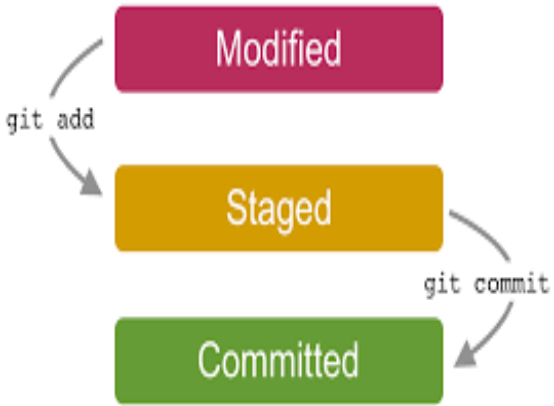

Basics Terminology
Add+Commit
Intro to next class+Doubts
10 mins
10 mins
30 mins
30 mins
15 mins
Installation+Getting the Git repo
Branch



Basic Terminology

- **Git Directory** is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you clone a repository from another computer. In standard terminology it is often referred as **Repository.**

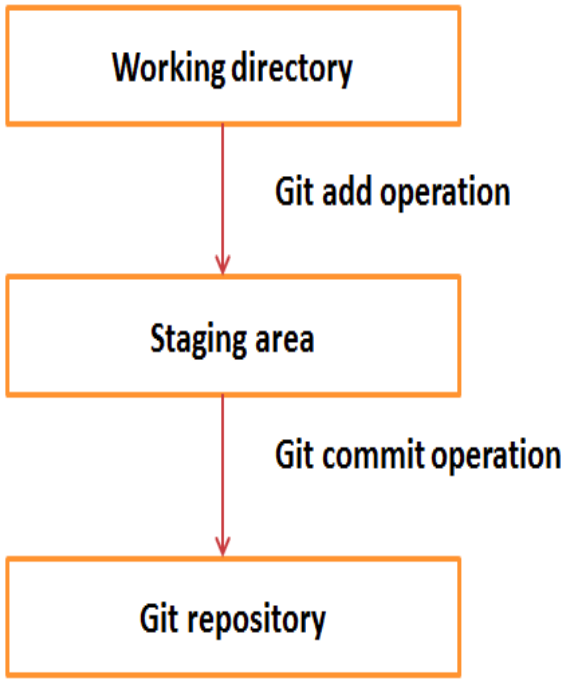
Basic Terminology (Contd.)



- **The Three States**-
  - **Committed** means that the data is safely stored in your local database.
  - **Modified** means that you have changed the file but have not committed it to your database yet.
  - **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot. The **staging area(Index)** is a file, generally contained in your Git directory, that stores information about what will go into your next commit.

Basic Terminology (Contd.)

- **Working Tree-** is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.
- **Commit**- Commit holds the current state of the repository. A commit is also named by SHA1 hash. It is the snapshot of a repository at a particular instance.
- **Branch**- Branches are used to create another line of development.

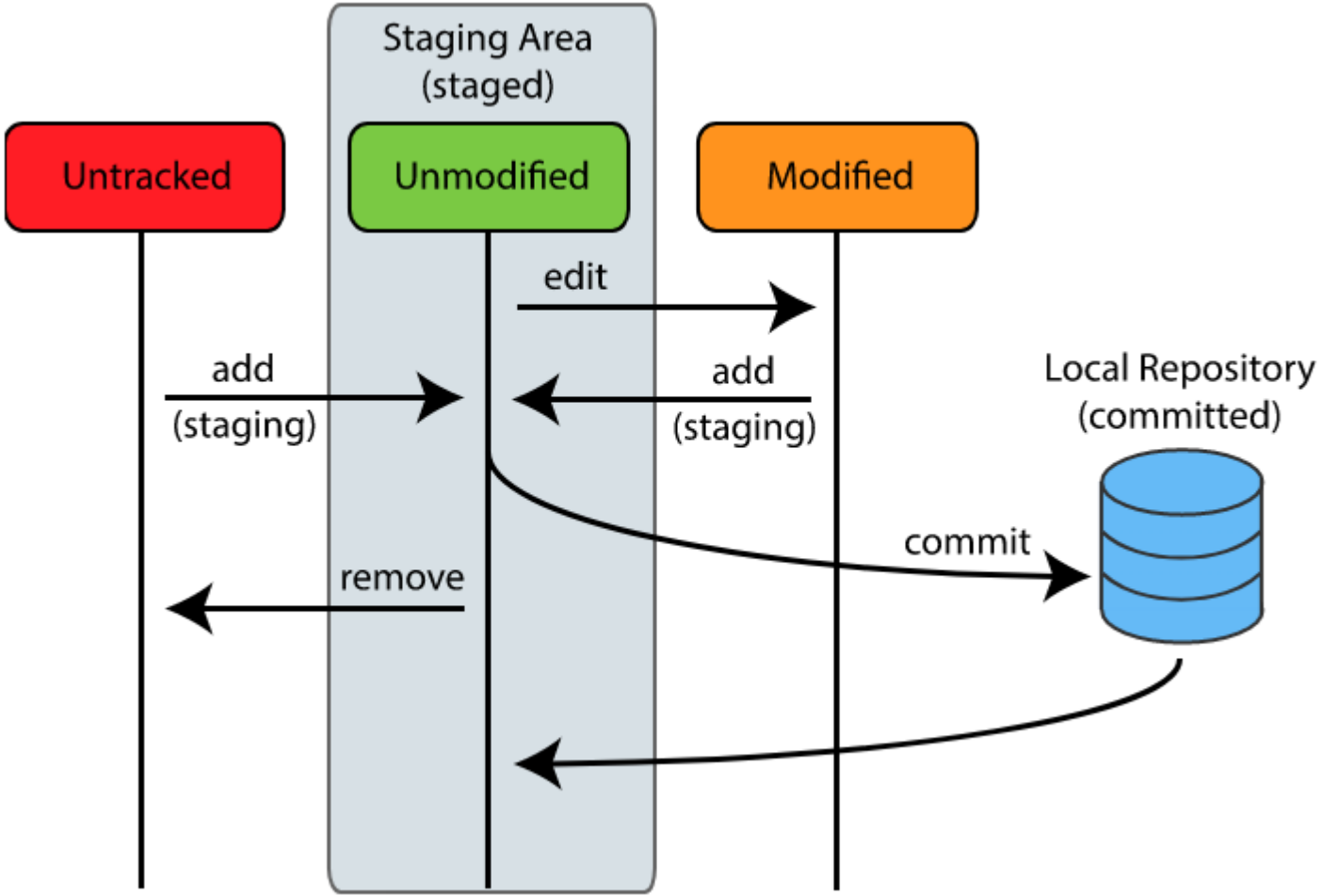


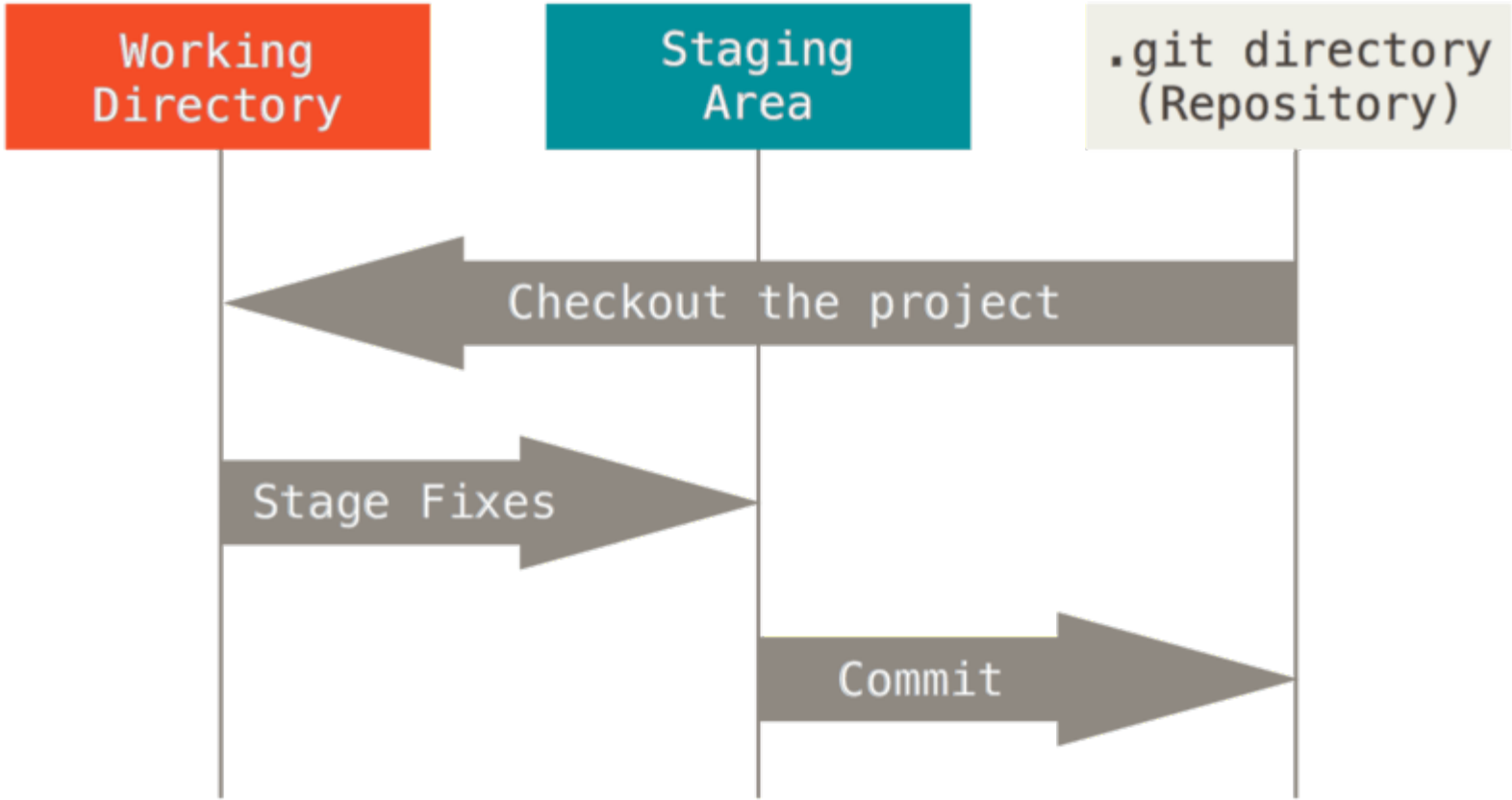
Basic Workflow

1. You modify files in your working tree.

2. You selectively stage just those changes you want to be part of your next commit, which adds only those changes to the staging area.

3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.


Staging Area
(staged)
Untracked
Unmodified
Modified
edit
add
(staging)
add
(staging)
Local Repository
(committed)
commit
remove


Working Directory
Staging Area
.git directory (Repository)
Checkout the project
Stage Fixes
Commit

Installation

1. Fedora or CentOS

   sudo dnf install git-all

2. Debian-based distribution like Ubuntu

   sudo apt-get install git-all

3. Arch-based distribution like Manjaro <3

   sudo pacman -S git

Set-up

- git config --global user.name "<your username>"
  - Check- git config user.name → gives <your username> as output
- git config --global user.email “<your email>”
- git config --global user.password “<your password>”
- Optional:
  - git config --global core.editor emacs
  - git config --list - lists settings

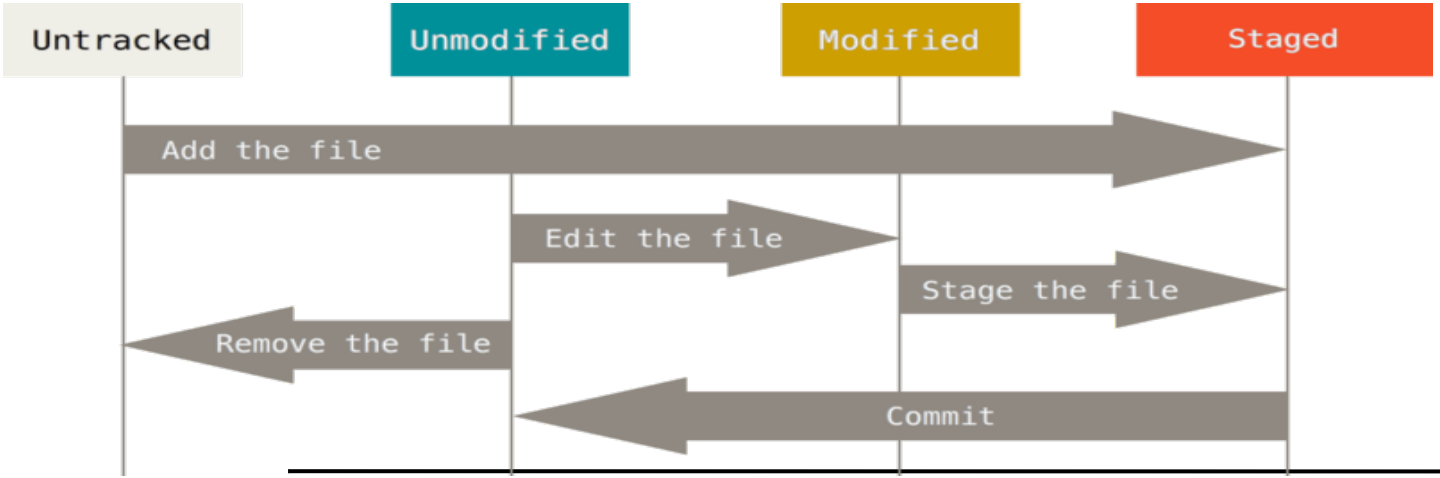
Creating & Getting a Git Repository

- If you want to get a copy of an existing Git repository
  - git clone https://github.com/username/repo_name.git
  - This topic will be covered in detail later(during GitHub classes).

- If you want to control an existing local directory using Git VCS:
  - Open the terminal in the desired directory
  - Type: git init
  - This creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton.

Recording Changes

- **Untracked Files**- Files neither in the last snapshot(commit) nor in the staging area
- **Tracked Files**- These can be in three states: unmodified, modified and staged.



Important commands for Recording and all...

- git status- This command can be used to check the status of recorded changes.
- git add <file_name/direcrory_name>- This command can be used to add a particular file/directory to staging area.
- git add -A- This command adds all changes to the staging area.

❖ Read yourself git status -s, gitignore

git diff [<a> <b>]

- [] → optional
- Compares state <b> to <a> → That is whatever is added to <b> is shown with ++ and removed with --
- If a=Φ, b=Φ → compares the content of working directory to staging area that is, if something is added to the working directory it is shown with ++ and so on. Its equivalent to a= staging area and b=working directory
- If a=commit_old, b=commit_new → Shows what has been added to commit_new as compared to commit_old
- If git diff --staged → compares staging area to last commit

It’s important to note that git diff by itself doesn’t show all changes made since your last commit— only changes that are still unstaged. If you’ve staged all of your changes, git diff will give you no output.

Committing Changes

- The commit records the snapshot you set up in your staging area.
- Type: git commit
- Hit Enter key
- This will open an Editor window where you can write the commit message describing the commit.
- git commit -m “Message for commit” → is used for inline committing
- git commit -a -m “Skips the usage of the staging area” → Directly adds the track files to the staging area and commits

Removing Files

git rm <file_name> allows you to delete the file from the staging area and working directory.

- How is it different from rm <file_name>?
  - <file_name> shows up under the “Changes not staged for commit” (that is, unstaged) area of your git status output
  - git rm --cached <file_name> → for keeping file in hdd and but avoiding it from staging area (like gitignore)
  - See git rm -f <file_name>

Commit History

- git log lists the commits made in that repository in reverse chronological order — that is, the most recent commits show up first.
- Commits are in the SHA form.
- git commit --amend - for adding the new staged changes to the last commit
- git reset HEAD <file_name> - for unstaging a file
- git checkout <commit_id/name> - for moving to any previous commit. The repo. structure will become same as the snapshot corresponding the commit. But remember commits after that commit will then be erased from the commit log.
- git show <commit_id/name>- shows description of the commit

Assignment

1. Create Github account.
2. Create a local repository on your system named <Your Github UserName>.github.io
3. Add an index.html file in that repo and then write a brief description about yourself in that file.

(Please make sure that you commit at regular intervals while editing index.html file)