Lesson: Introduction To Git

- MNNIT COMPUTER CLUB

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

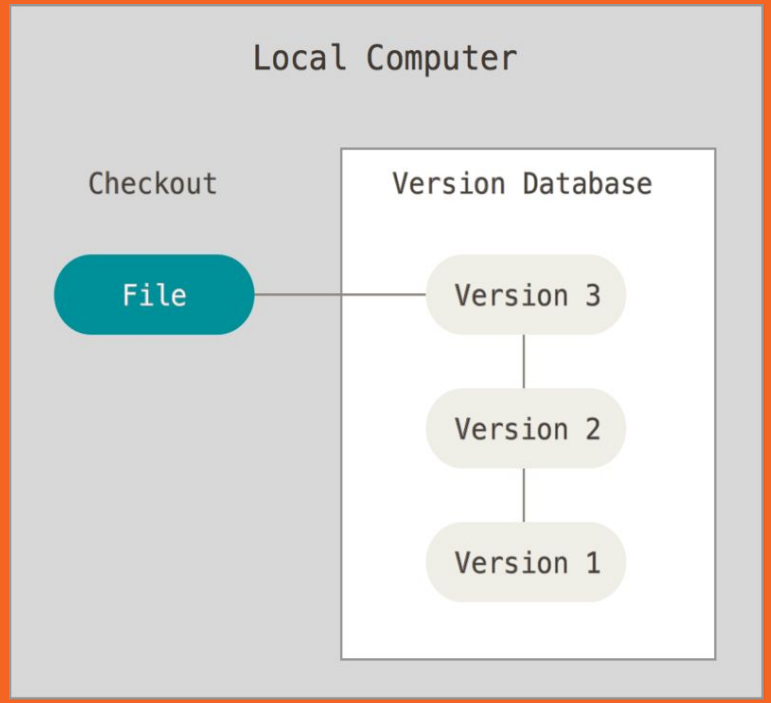
- Pro Git Book - by Scott Chacon and Ben Straub
- How to use Git And Github - Udacity Course
- Git started with Github - Udemy Course
- Lots of online tutorials ;-)

What 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.



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.

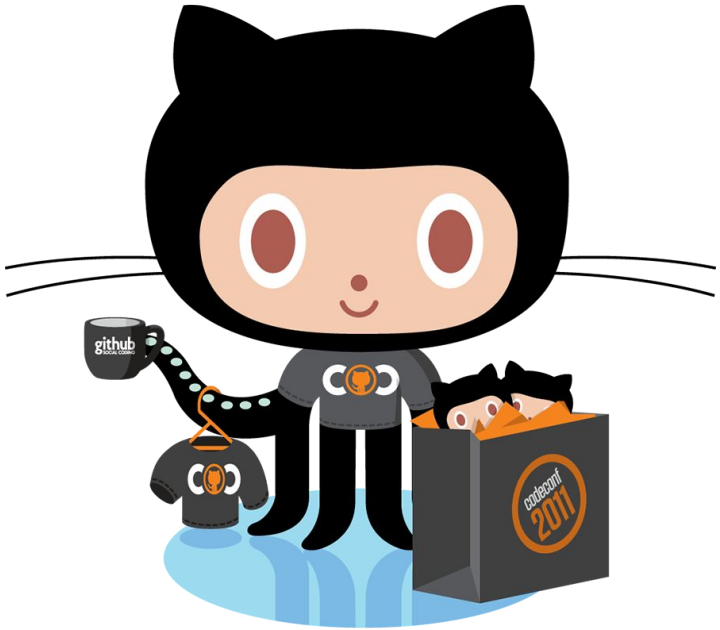
Because It's Cool -(01)

- 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)

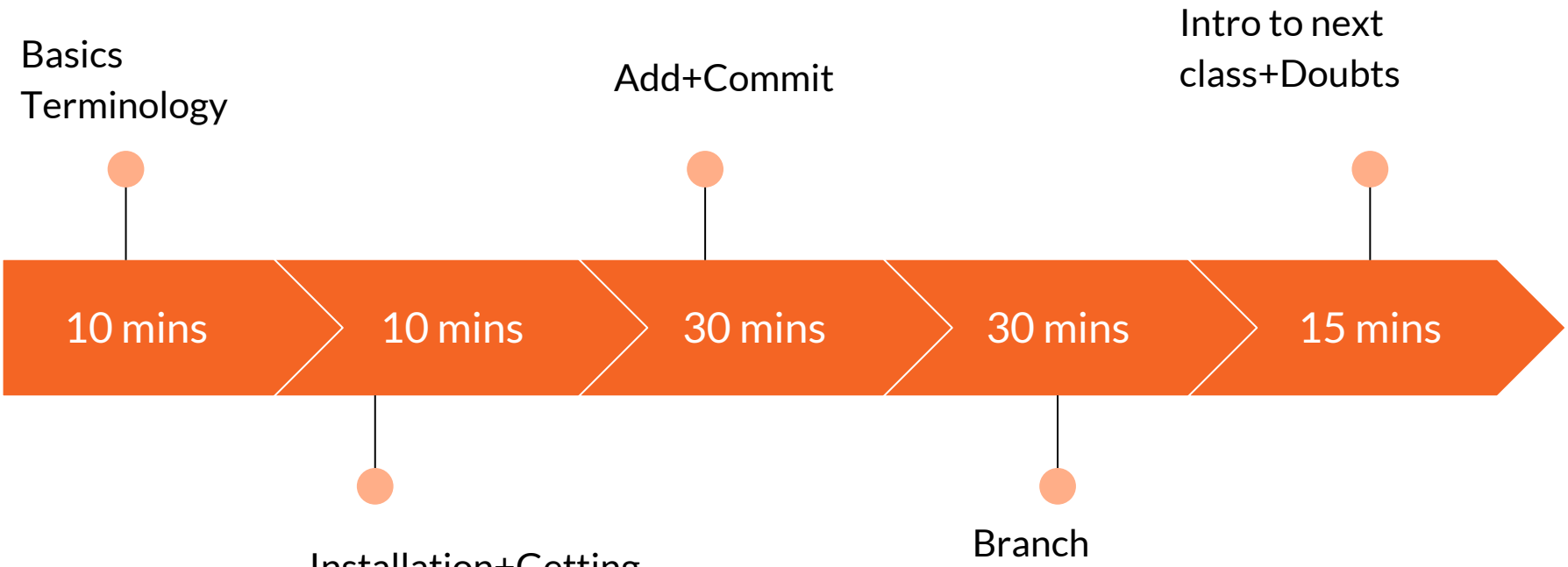
- Everything in Git is check-summed before it is stored and is then referred to by that checksum. This means it's impossible to change the contents of any file or directory without Git knowing about it.
- Uses 40 character SHA-1 Hash like 24b9da6552252987aa493b52f8696cd6d3b00373

Because It's Cool -(11)

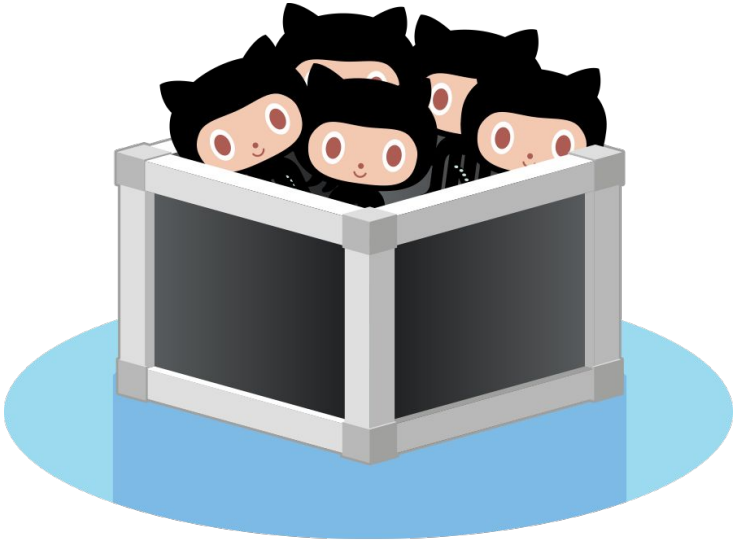


As with any VCS, you can lose or mess up changes you haven't committed yet, but after you commit a snapshot into Git, it is very difficult to lose, especially if you regularly push your database to another repository.

How to use Git?

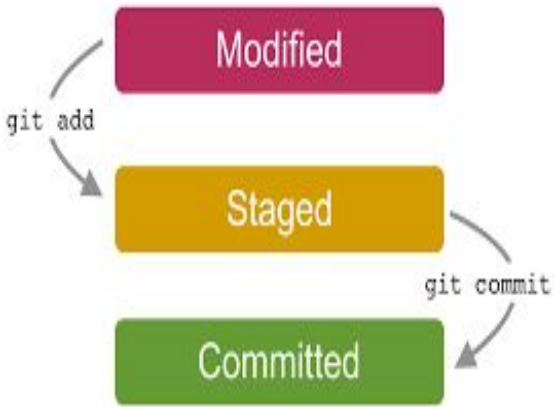

Basics
Terminology
Add+Commit
Intro to next
class+Doubts
10 mins
10 mins
30 mins
30 mins
15 mins
Installation+Getting
the Git repo
Branch

Basic Terminology



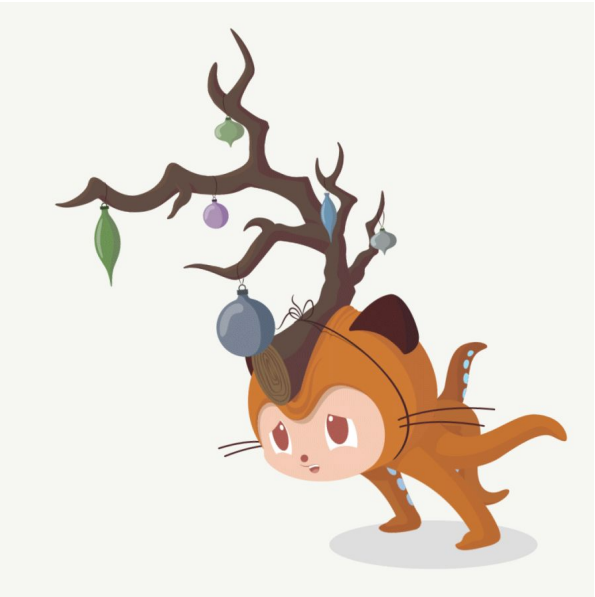
- **Git Directory-** is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you clone a repository from another computer. In standard terminology it is often referred as **Repository.**

Basic Terminology (Contd.)



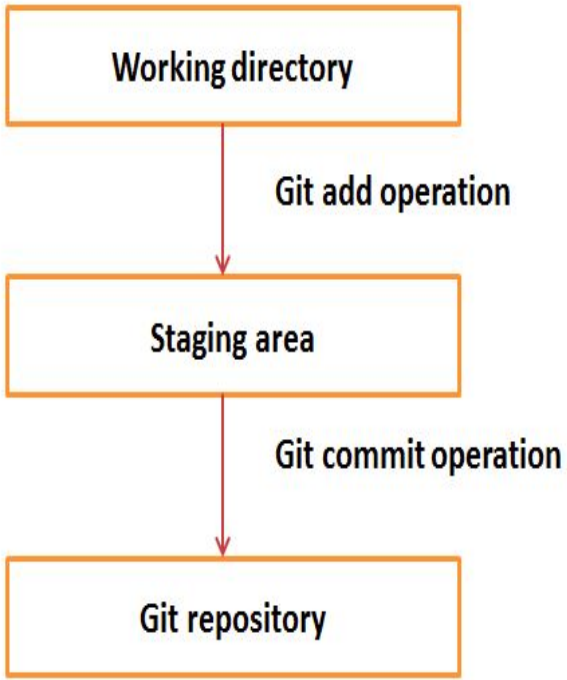
- **The Three States**-
  - **Committed** means that the data is safely stored in your local database.
  - **Modified** means that you have changed the file but have not committed it to your database yet.
  - **Staged** means that you have marked a modified file in its current version to go into your 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.

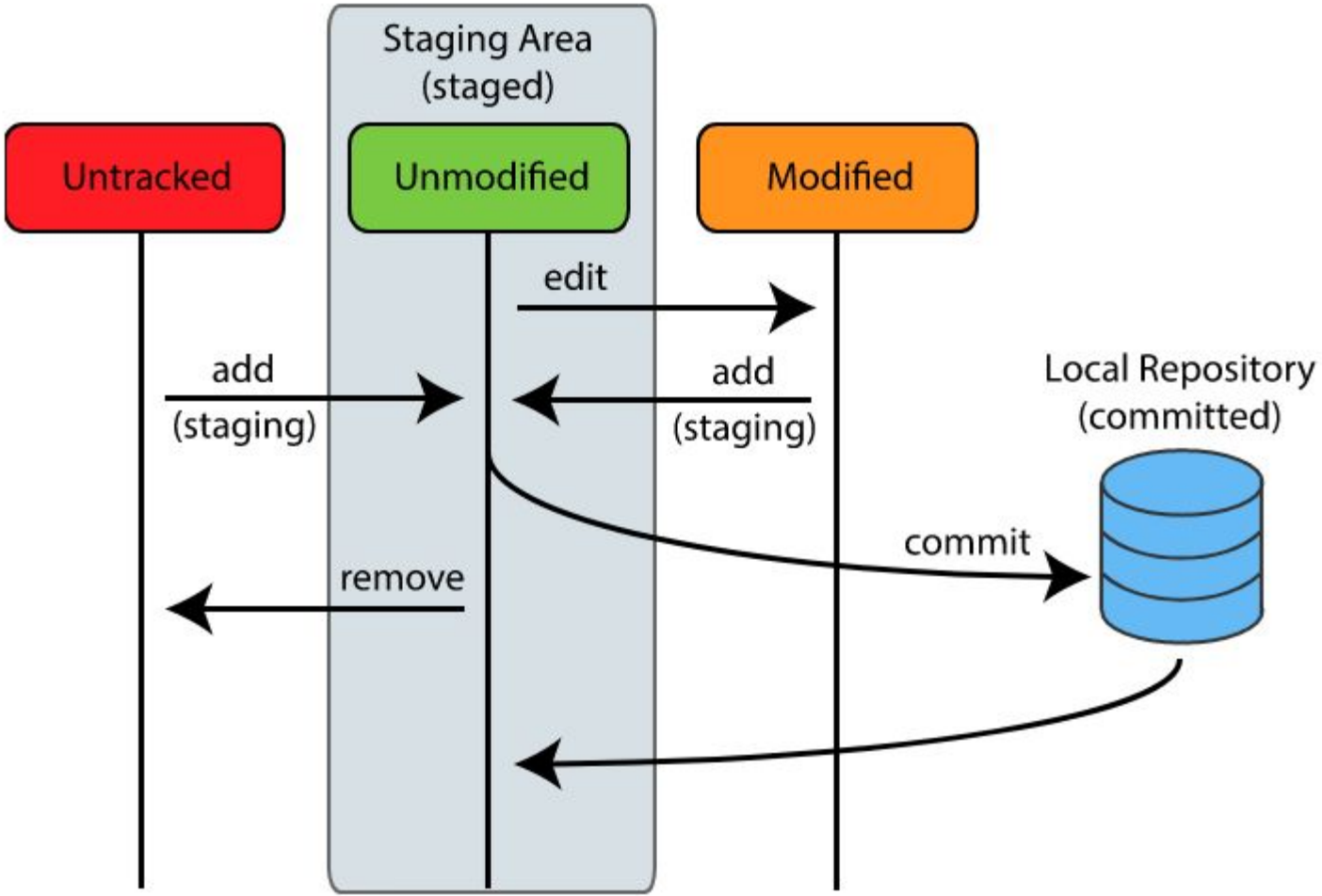
Basic Workflow



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

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

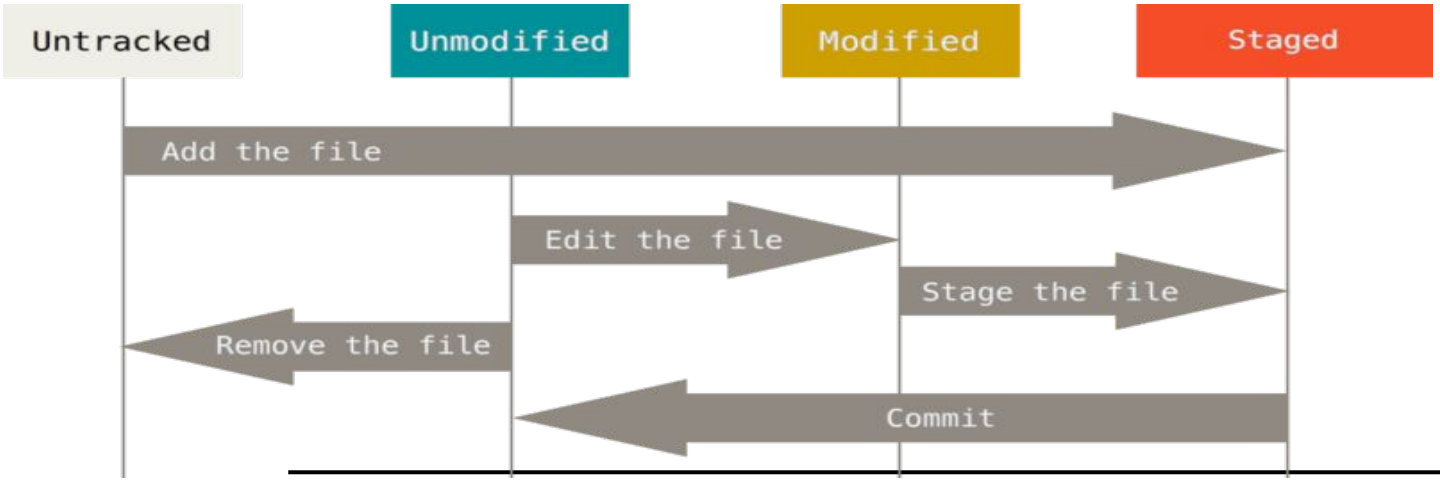
Creating & Getting a Git Repository

- 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

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

Lesson: Git Advanced Features

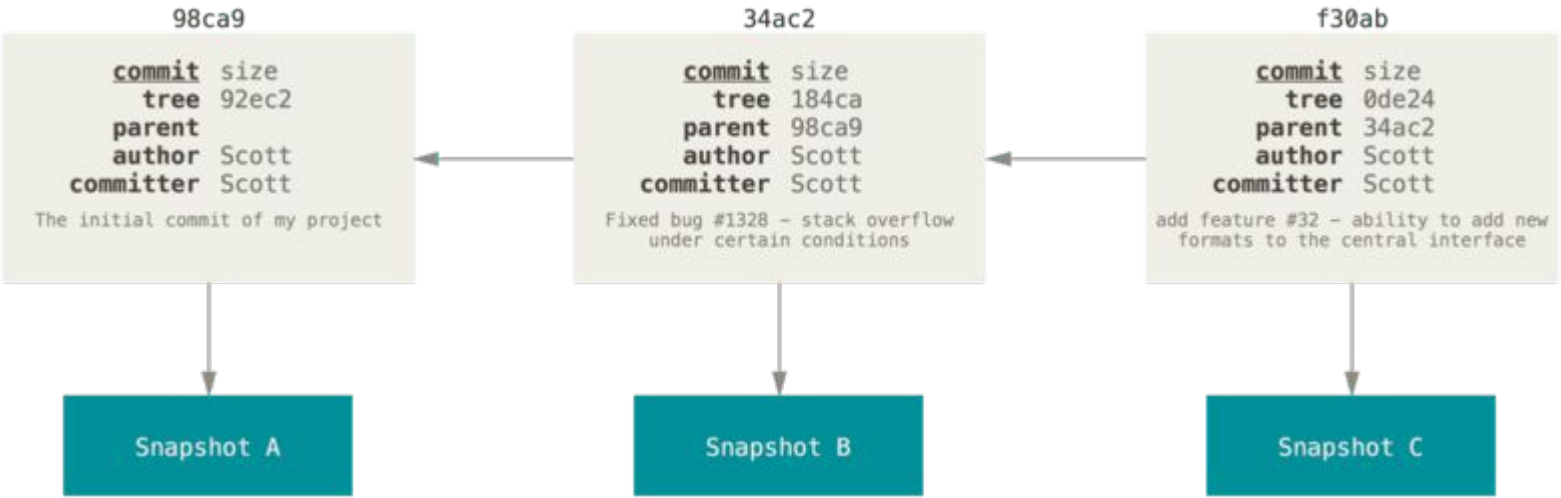
- MNNIT COMPUTER CLUB

March 8, 2018

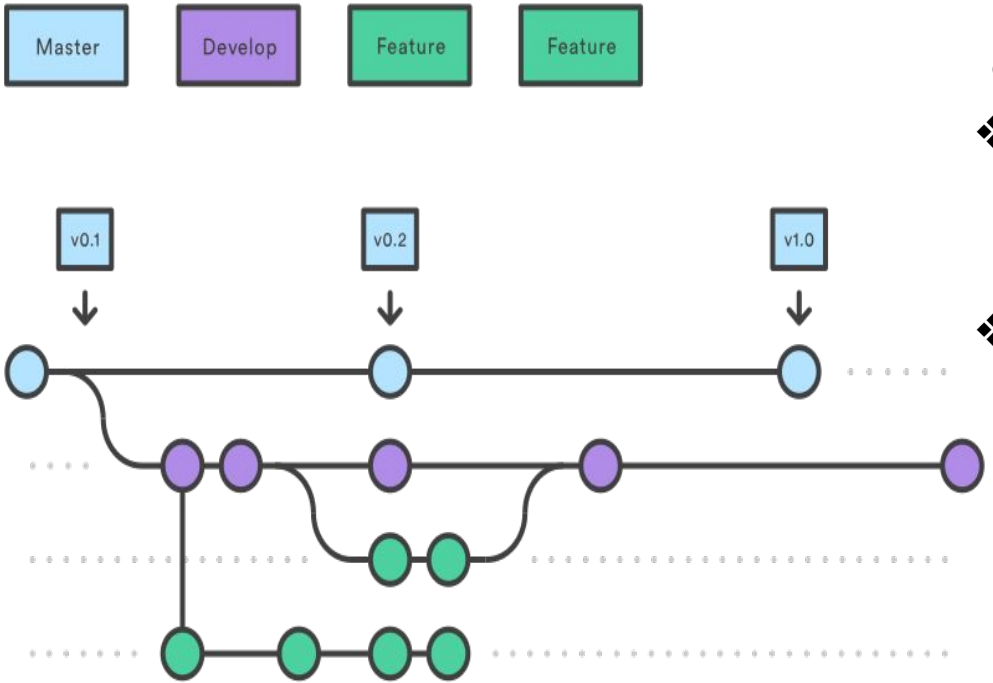
Git Branching

- When you make a commit, Git stores a commit object that contains a pointer to the snapshot of the content you staged. This object also contains the author's name and email, the message that you typed, and **pointers to the commit or commits that directly came before this commit** (its parent or parents): zero parents for the initial commit, one parent for a normal commit, and multiple parents for a commit that results from a merge of two or more branches
- A **branch** in Git is simply a lightweight movable pointer to one of these commits. Every time you commit, it moves forward automatically.
- The default branch name in Git is master

Git Branching

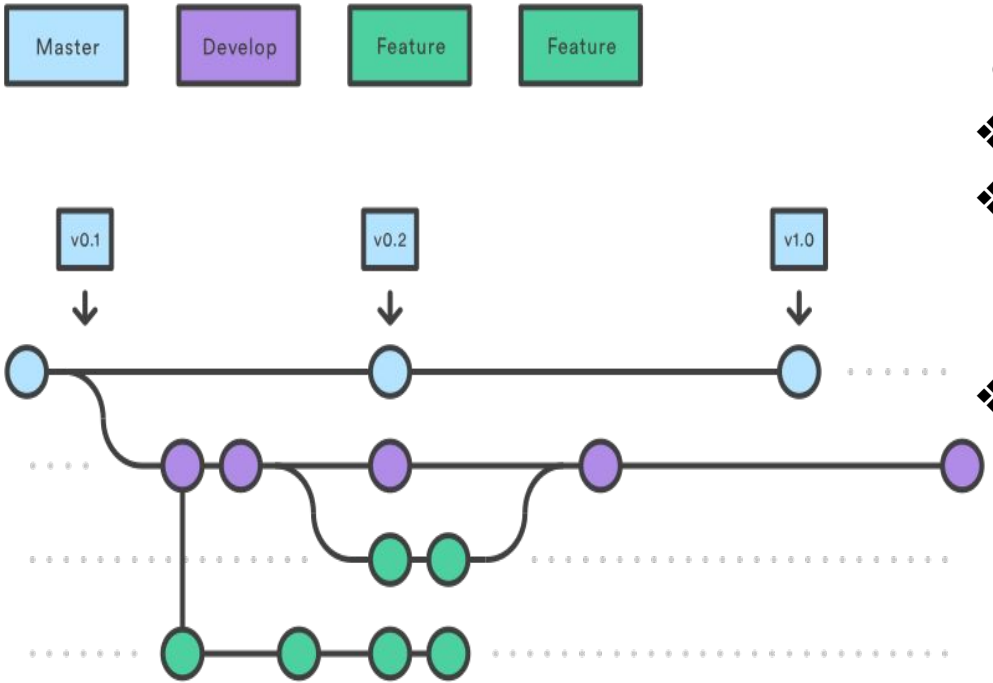


Creating a Branch



- git branch <branch_name>
- How does Git know what branch you're currently on? It keeps a special pointer called HEAD.
- Example: git branch testing

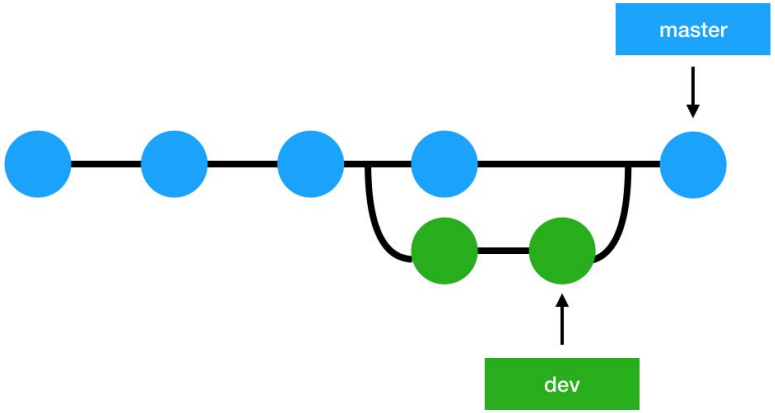
Switching a Branch



- git checkout <branch_name>
- Example: git checkout testing
- git log --oneline --decorate - to see where the branch pointers are pointing
- git checkout -b <branch_name> - to create and switch

Note that if your working directory or staging area has uncommitted
changes that conflict with the branch you're checking out, Git won't let you switch branches (Stashing and Cleaning helps!!).

Git Merge



- git merge <new_branch_name> : merges the new_branch_name with the current working branch.

❖ When you try to merge one commit with a commit that can be reached by following the first commit's history, Git simplifies things by moving the pointer forward because there is no divergent work to merge together — this is called a "**fast-forward**."

➢ git branch -d <branch_name> : deletes branch <branch_name>

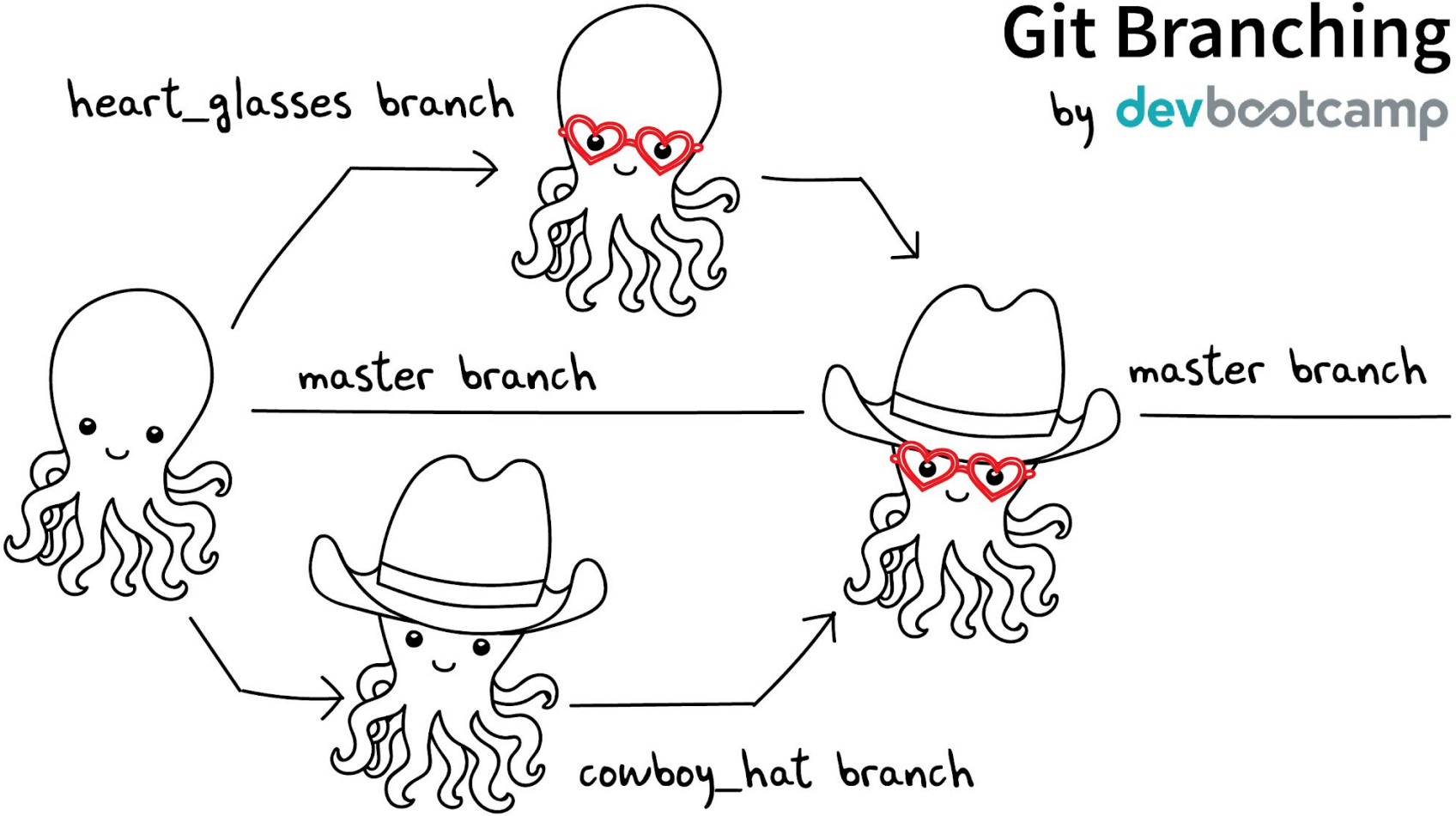
➢ git branch : lists the branches

Basic Merge Conflicts

- If you change the same part of the same file differently in the two branches you're merging together, Git won't be able to merge them cleanly.
- Anything that has merge conflicts and hasn't been resolved is listed as unmerged.
- After you've resolved each of these sections in each conflicted file, run git add on each file to mark it as resolved. Staging the file marks it as resolved in Git.
    - Read git mergetool yourself
    - git branch --merged: shows branches merged into the current branch
    - git branch --no-merged: shows branches not merged into the current branch(They can only be deleted forcefully(-D))

Git Branching

by devbootcamp



Remote Repositories

- Remote repositories are versions of your project that are hosted on the Internet or network somewhere.
- Remote repositories can be on local machine.
- In layman terms, remote is a kind of remote controller for any repository hosted anywhere else.
- git remote command is used for listing the remote servers. For cloned repositories, its origin.

Git Fetch

- The command goes out to that remote project and pulls down all the data from the remote project that you don’t have yet.
- After running this command, you have references to all the branches of that remote repo.

❖ git fetch <remote>

Git Pull

- The command goes out to that remote project and pulls down all the data from the remote project that you don’t have yet and merges the remote branch into your current branch in case there are no merge conflicts.

❖ git pull <remote> <branch>

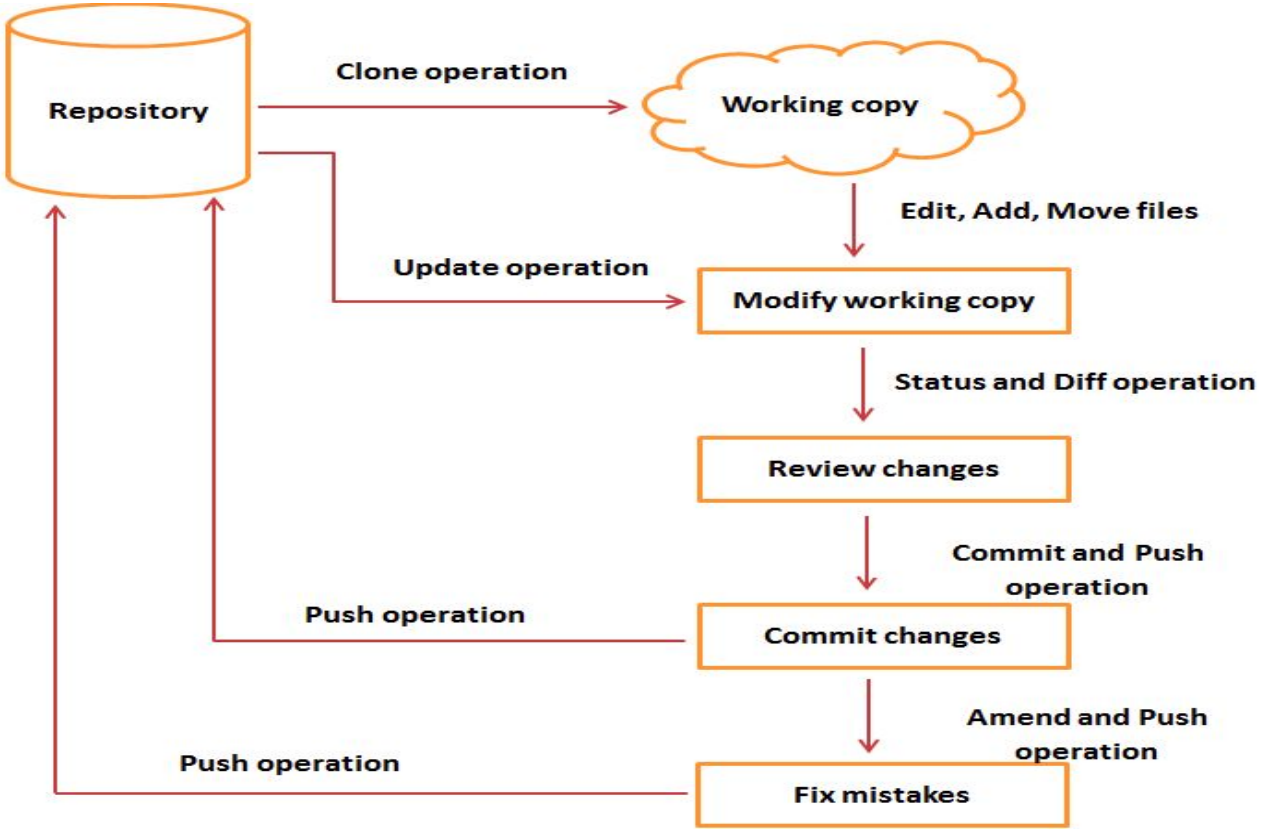
Git Push

- This command is used for adding your commits/changes to the remote repository.
- For this command to work, the working tree should remain updated with the most recent changes of the remote repo.

❖ git push <remote> <branch>

❖ **Read yourself:** git stash

Summary


Repository
Clone operation
Working copy
Edit, Add, Move files
Update operation
Modify working copy
Status and Diff operation
Review changes
Commit and Push operation
Push operation
Commit changes
Amend and Push operation
Push operation
Fix mistakes