Open Source Software(OSS)

Lecturer :Taghreed Alreffaee

Lecture: #3

Contents:

Why Version Control?

Scenario 1:

- Your program is working
- You change “just one thing”
- Your program breaks
- You change it back
- Your program is still broken--why?

Has this ever happened to you?

Why Version Control? (part 2)

Scenario 2:

- Your program worked well enough yesterday
- You made a lot of improvements last night...
- ...but you haven't gotten them to work yet
- You need to turn in your program now

- **Has this ever happened to you?**

Why Version Control for teams

Scenario 1:

- You change one part of a program--it works
- Your co-worker changes another part--it works
- You put them together--it doesn't work
- Some change in one part must have broken something in the other part

What about all changes?

Teams (part 2)

Scenario 2:

- You make a number of improvements to a class
- Your co-worker makes a number of different improvements to the same class
- How can you merge these changes?

What is “Version Control System”, and why should you care?

Version Control System(VCS) is a system which allow professionals to record the changes that have been made to the documents and files by tracking the modifications made to the program code.

One of the main reasons why VCS is necessary is often, software programs are developed by a group of developers who may be working from various parts of the world. Since all of them make some contributions to the software code and keep making changes, it is important that the changes be communicated to other team members. This improves the management and efficiency while developing the software

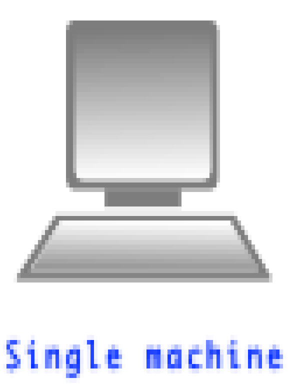
Version Control System

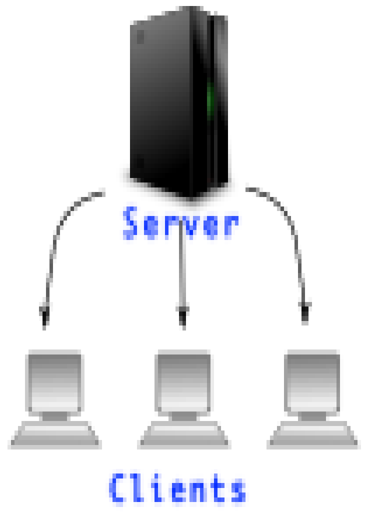
A version control system (often called a source control system) does these things:

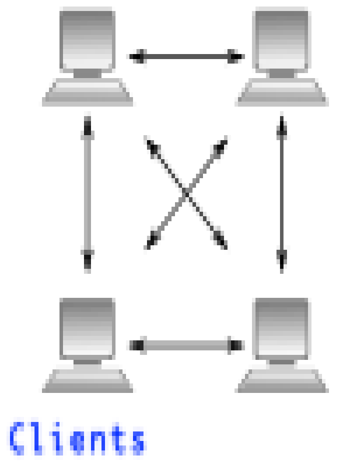
- ❑Keeps multiple (older and newer) versions of everything (not just source code) and Displays differences between versions and compare changes over time.
- ❑Requests comments regarding every change and trace each change with a message describing the purpose and intent of the change.
- ❑Allows "check in" and "check out" of files so you know which files someone else is working on.

- It allows you to revert selected files back to a previous state, (A complete history of every file, which enables you to go back to previous versions to analyze the source of bugs and fix problems in older versions).
- see who last modified something that might be causing a problem.
- who introduced an issue and when, and more.
- Using a VCS also generally means that if you screw things up or lose files, you can easily recover.
- The ability to work on independent streams of changes, which allows you to merge that work back together and verify that your changes conflict.
- For working with others:
  - Greatly simplifies concurrent work, merging changes

VCS Types:-



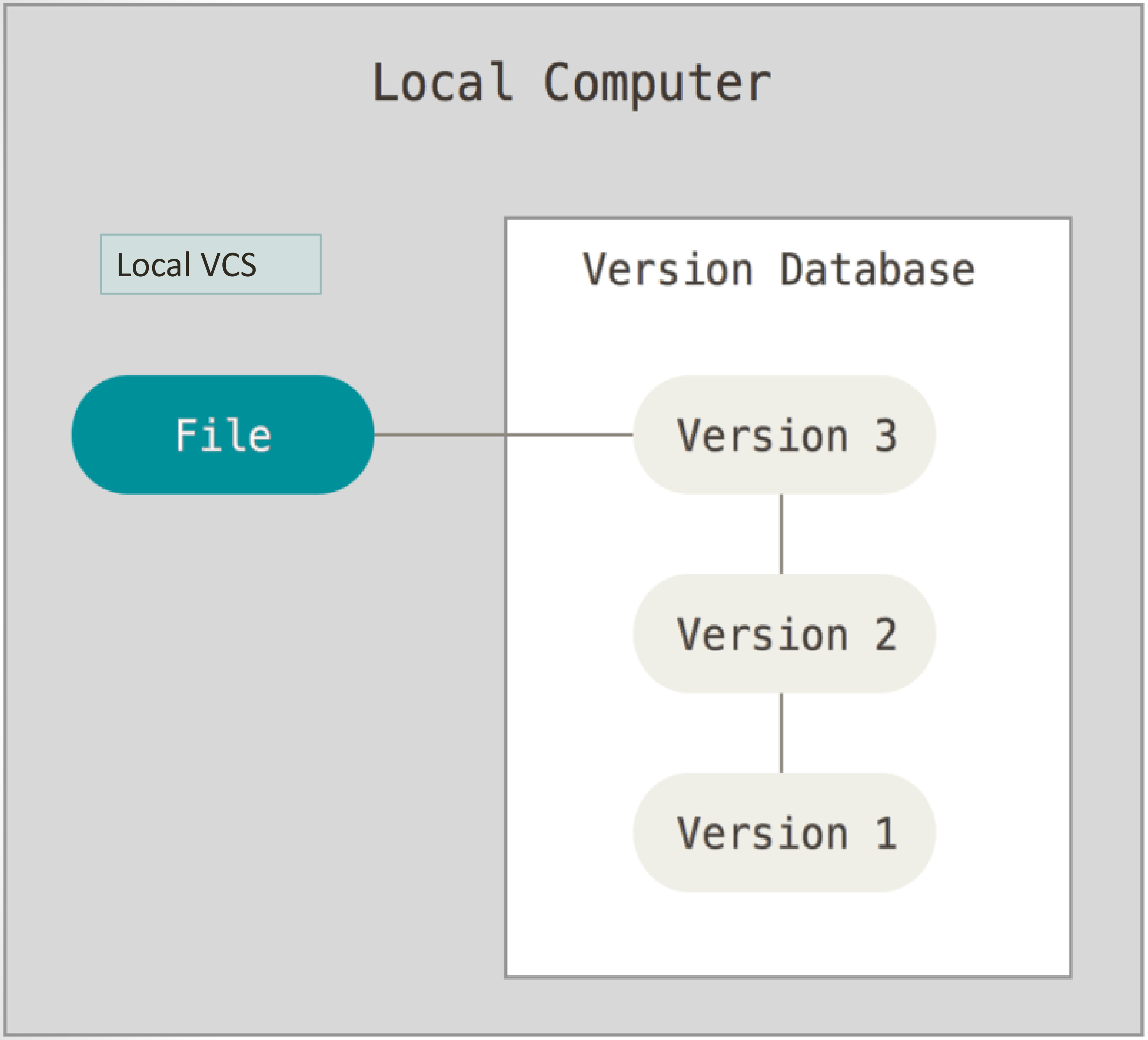




Local Centralised Distributed

Local Version Control System

- A local version control system is a local database located on your local computer, in which every file change is stored as a patch. **Every patch set contains only the changes made to the file since its last version.** In order to see what the file looked like at any given moment, it is necessary to add up all the relevant patches to the file in order until that given moment.
- The main problem with this is that everything is stored locally. If anything were to happen to the local database, all the patches would be lost. If anything were to happen to a single version, all the changes made after that version would be lost.
- Also, collaborating with other developers or a team is very hard or nearly impossible.

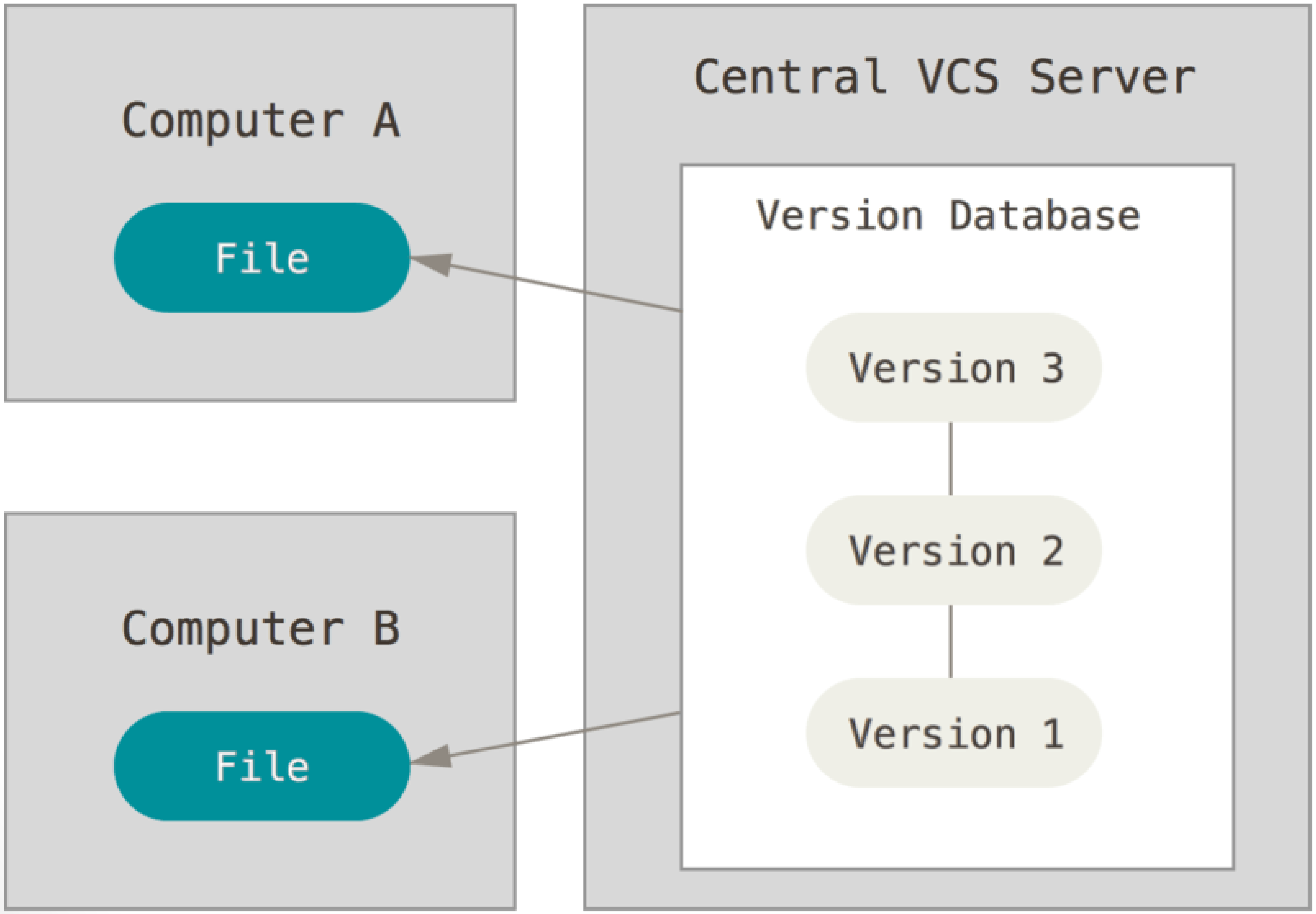

Local Computer
Local VCS
File
Version Database
Version 3
Version 2
Version 1

Centralized Version Control Systems

- The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed.
- These systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place. For many years, this has been the standard for version control
- A centralized version control system has a single server that contains all the file versions. This enables multiple clients to simultaneously access files on the server, pull them to their local computer or push them onto the server from their local computer. This way, everyone usually knows what everyone else on the project is doing. Administrators have control over who can do what.
- This allows for easy collaboration with other developers or a team.

- The biggest issue with this structure is that everything is stored on the centralized server. If something were to happen to that server, nobody can save their versioned changes, pull files or collaborate at all. Similar to Local Version Control, if the central database became corrupted, and backups haven't been kept, you lose the entire history of the project except whatever single snapshots people happen to have on their local machines.

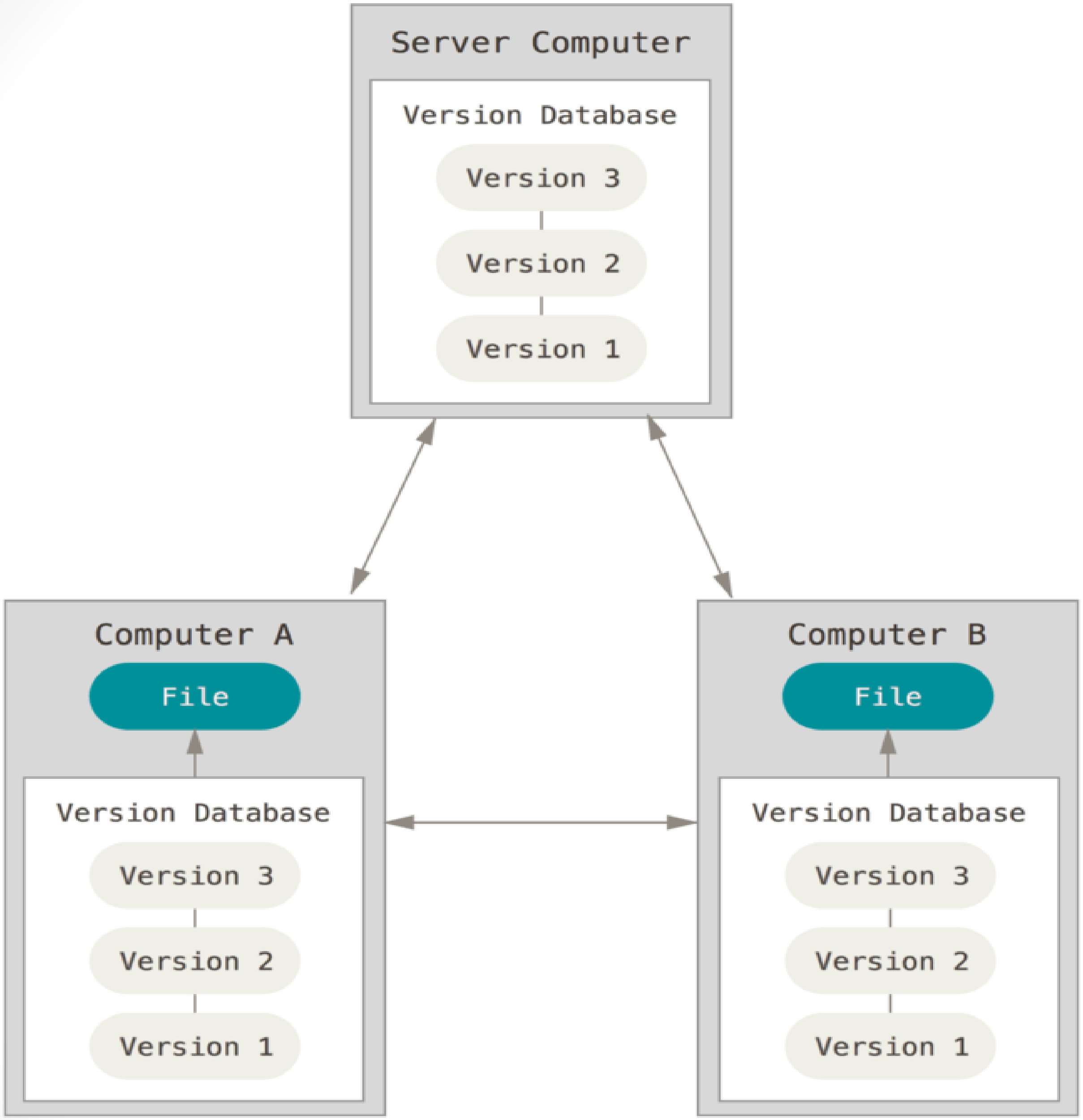
- The most well-known examples of centralized version control systems are Microsoft Team Foundation Server (TFS) and SVN.


Computer A
File
Central VCS Server
Version Database
Version 3
Version 2
Version 1
Computer B
File

Distributed Version Control Systems

- With distributed version control systems, clients don’t just check out the latest snapshot of the files from the server, they fully mirror the repository, including its full history. Thus, everyone collaborating on a project owns a local copy of the whole project, i.e. owns their own local database with their own complete history. With this model, if the server becomes unavailable or dies, any of the client repositories can send a copy of the project's version to any other client or back onto the server when it becomes available. It is enough that one client contains a correct copy which can then easily be further distributed.

- **Git is the most well-known example of distributed version control systems**


Server Computer
Version Database
Version 3
Version 2
Version 1
Computer A
File
Version Database
Version 3
Version 2
Version 1
Computer B
File
Version Database
Version 3
Version 2
Version 1

git

GitHub

What is Git?

- Git is a free and open source (DVCS) designed to handle everything from small to very large projects .
- it is one of the most popular version control tools that is being used by 90% of the world's IT companies
- software that manages different versions of files
- Keep careful track of changes in your files
- Collaborate with others on your projects more easily
- Test changes without losing the original versions
- Revert back to older versions when/if needed

What is the Github

- GitHub is a web-based platform for version control and collaborative software development. It provides a way for developers to store, manage, and share their code with others. GitHub allows developers to create and host repositories, which are collections of code files and other resources related to a specific project. These repositories can be public or private, depending on the developer's preferences.

- GitHub: web-based hosting service for git
- Provides a "remote" location for storing your git workspaces
- Useful if you lose/break your computer, etc.

Why use Git?

- Since the development and release of Git, it has gained huge popularity among the developers and being open source have incorporated many features. Today, a many number of projects use Git for version control, both commercial and personal. Let's see why Git has become so popular by discussing its main features:

- **Performance**: Git provides the best performance when it comes to version control systems. Committing, branching, merging all are optimized for a better performance than other systems.
- **Security**: Git handles your security with cryptographic method SHA-1. The algorithm manages your versions, files, and directory securely so that your work is not corrupted.

- **Staging Area**: Git has an intermediate stage called "index" or "staging area" where commits can be formatted and modified before completing the commit.
- **Distributed**: Git is distributed in nature. Distributed means that the repository or the complete code base is mirrored onto the developer's system so that he can work on it only.
- **Free and open-source**: No need to purchase and can be used anywhere.
- **Fast and small**: No need to connect to the central server. All the operations can be performed on the local copy stored on the developer machine making it really fast.
- **Implicit backup**: Every checkout is a full backup of the repository, hence multiple copies are available.
- **Easy branching**: Git branches are easy and cheap to merge. Every small change to your code creates a new branch

Why use GitHub?

1. Free hosting for open source projects
2. Interface for browsing and editing code
3. Workflows for collaborating with others
4. API for doing other fancy things
5. popularity

Homework

What is the difference between Git and GitHub

Git Workflow

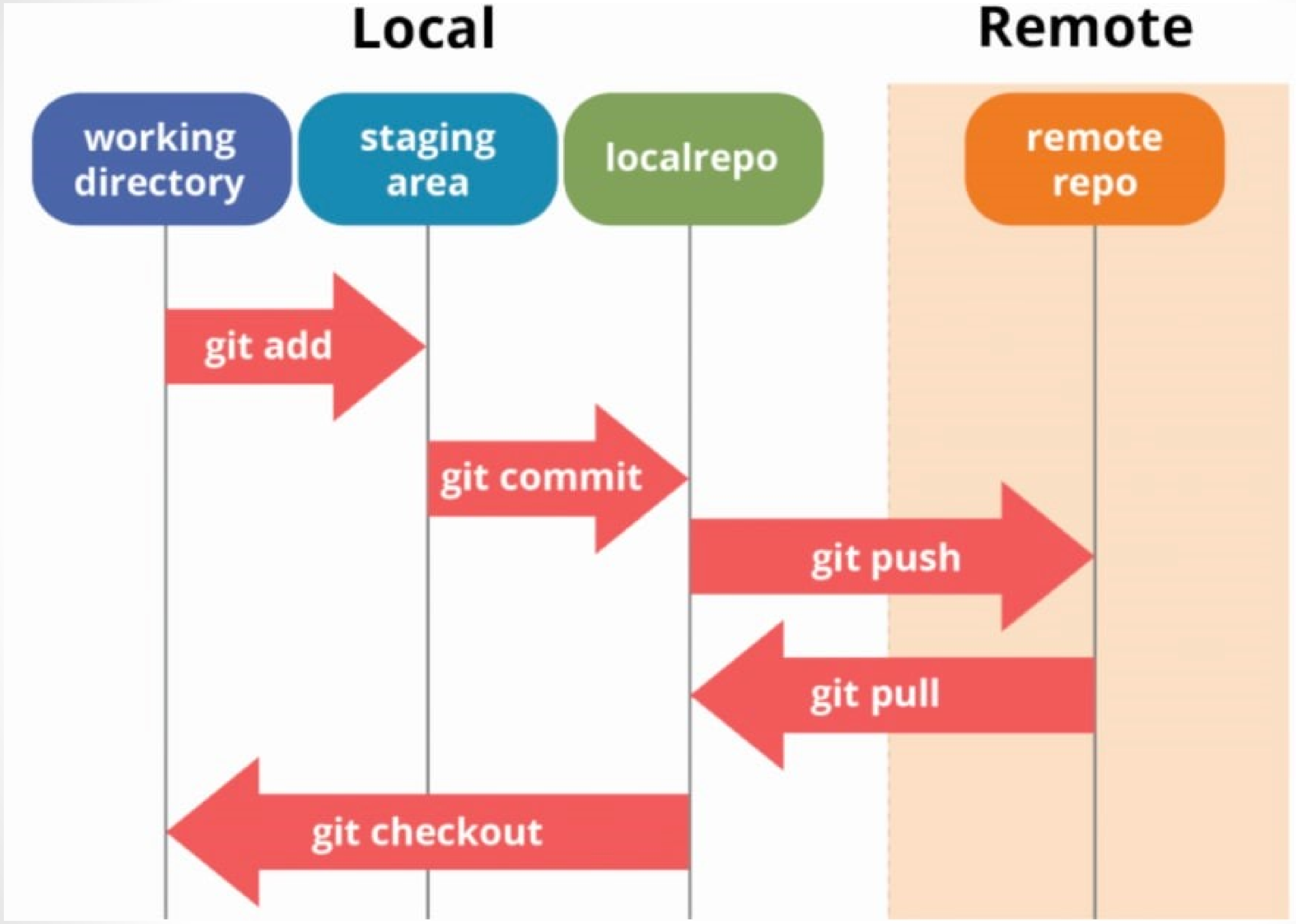
- Git has a 3 layered workflow and these layers describe the different states that your files can reside in.
- **Working directory:**
- **Staging area or Index:**
- **Git repository:**

How Basic Git Workflow Works?

- The working directory is where all your files are present, where you make changes in your files.

- The staging area is an intermediate layer where you stage those changes that you want to be part of your next commit, so you add only those changes to the staging area.

- Once you commit, it takes these files from the staging area and stores them permanently to the Git repository


Local
Remote
working directory
staging area
localrepo
remote repo
git add
git commit
git push
git pull
git checkout

Using Git:-

- Installation :

https://git-scm.com/downloads

- How it works :

- ○ Create a "repository" (workspace) for your project
- ○ Add/remove/save/edit files
- ○ Push local files online to GitHub / pull remote files from GitHub to your local workspace
- ○ And more!

Basic Git command

Introduce yourself to Git:

-On your computer, open the **Git Bash** application.
- Enter these lines:

```
git config --global user.name “Taghreed Riyadh"
git config --global user.email taghreed@uomosul.edu.iq
```

You only need to do this once

To make a directory for a project

- mkdir FirstProject
- pwd // path
- cd FirstProject   // to go inside the dir
- cd ..   // exit from and dir

- Create FirstFile.txt inside FirstProject   dir
- cat FirstFile.txt   // go inside file
- ls   //list all files inside FirstProject dir

init and the .git repository

- Now that Git is installed and the user information established, you can begin creating new repositories. From a command prompt, To initialize the directory as a Git repository by typing the following command:

- git init
- ls –a   //list hidden files
- ls -la .git   //every thing inside .git

- When you said git init in your project directory, or when you cloned an existing project, you created a repository

- The repository is a subdirectory named .git containing various files.
- The dot indicates a "hidden" directory.
- You do *not* work directly with the contents of that directory; various git commands do that for you

The repository

- Your top-level **working directory** contains everything about your project
  - The working directory probably contains many subdirectories : source code, binaries, documentation, data files, etc.
  - One of these subdirectories, named .git, is your repository
  - At any time, you can take a "snapshot" of everything in your project directory, and put it in your repository
  - This "snapshot" is called a commit object
  -Commit objects do *not* require huge amounts of memory
    -You can work as much as you like in your working directory, but the repository isn't updated until you commit something

Making commits

 You do your work in your project directory and files , as usual

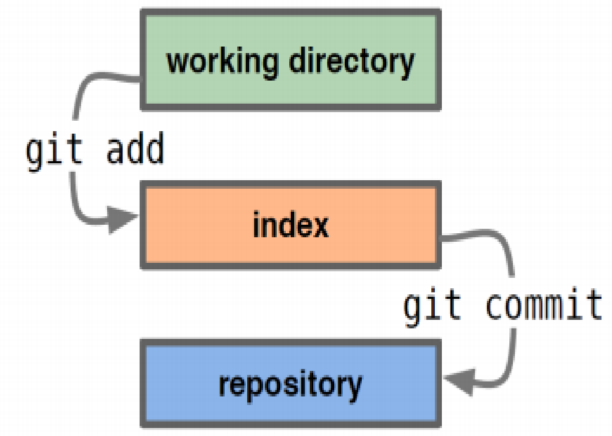
- git add *FirstFile.txt*

- Committing makes a “snapshot” of everything being tracked into your repository

- A message telling what you have done is required
- git commit –m “adding FirstFile.txt in my Project”

Typical workflow

- `git status`
  - See what Git thinks is going on
  - Use this frequently!
- Work on your files
- `git add your editfiles`
- `git commit -m "What I did"`



- mkdir FirstProject
- cd FirstProject
- touch README.txt ,py , …..
- create or add project files // working stage
- git init //builds a .git directory that contains all the metadata and repository history. Unlike many other version control systems, Git uniquely stores
  everything in just a single directory at the top of the project.
- git add . // staging index and begin to tracked files and (add .) To add and track all files inside the directory.
- git add myfirstfile.txt // to add this file only
- git commit -m “adding FirstFile.txt” // repo // commit -am
- git status // to show untracked files(no add).

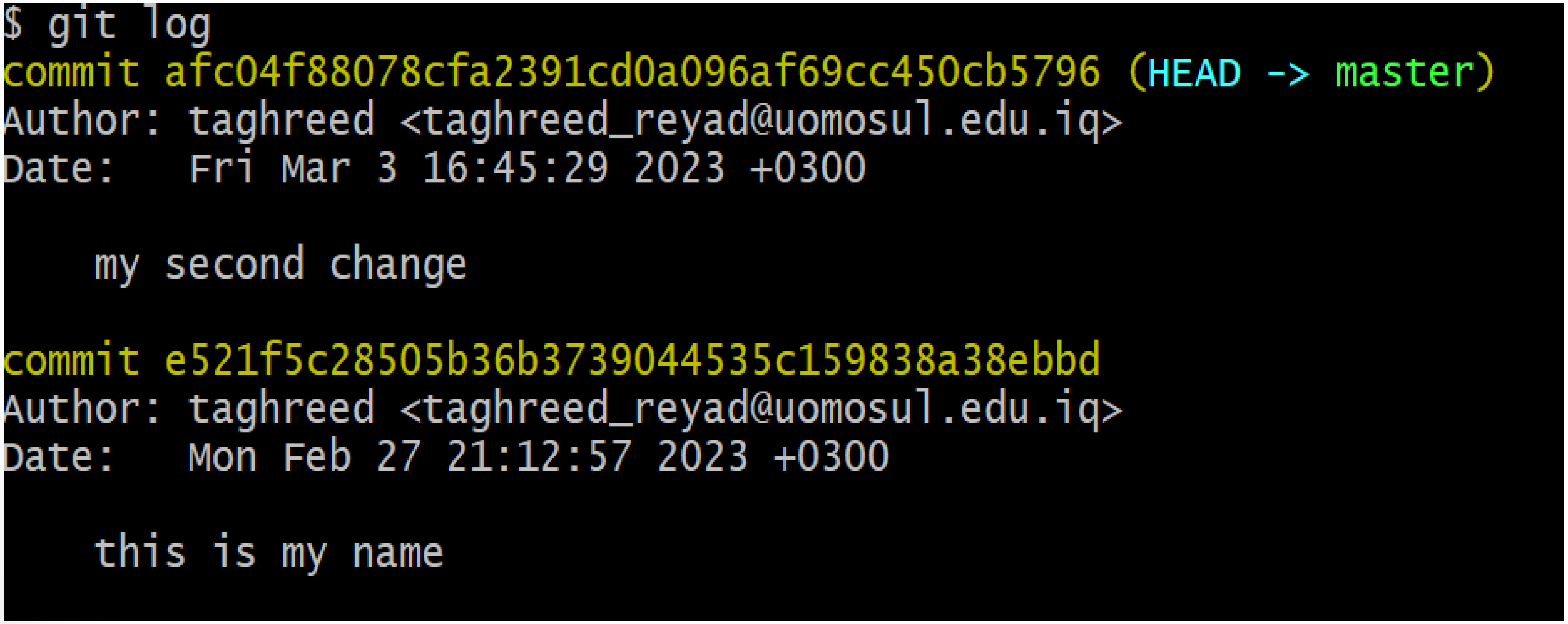
Git Log and checksum

- Git log // display checksum of any change and history of commits made in the repository from newest to oldest

  Using sh-1 algorithm to generate 40 hexa character according to any changes on files(tracking)

- Head : pointer storing the path which the change will beginning from

Git log



```
$ git log
commit afc04f88078cfa2391cd0a096af69cc450cb5796 (HEAD -> master)
Author: taghreed <taghreed_reyad@uomosul.edu.iq>
Date:   Fri Mar 3 16:45:29 2023 +0300

    my second change

commit e521f5c28505b36b3739044535c159838a38ebbd
Author: taghreed <taghreed_reyad@uomosul.edu.iq>
Date:   Mon Feb 27 21:12:57 2023 +0300

    this is my name
```

Determine type of change

- git diff display the details of change when file is on working tree
- git diff myfirstfile.txt // to determine which file wanted to display the changing on it
- git diff --staged // display change on file when the files is on stage index area **(add but no commit)**
- Or git diff --cached //in old version

Git Create New Remote Branch From Local Repository And Push to GitHub

- mkdir Testfile
- cd Testfile // open it using vscode then create new file git.txt
- Git add .
- Git commit –m “ adding new file”

Go to github then create new repo , copy the link on your repo

- Git branch –M Test
- git remote add origin https://github.com/taghreedgithub/taghreed.git.
- git push -u origin Test

HomeWork

What About :

- Merge
- pull

Thank You for Listening