



Lecturer: Jailani Abdul Rahman

Disclaimer

This material is derivative of:

- 'Version Control with Git' lesson by 'The Carpentries'.
<http://swcarpentry.github.io/git-novice/>
- Pro Git book, written by Scott Chacon and Ben Straub.
<https://git-scm.com/book/en/v2>
- How to teach Git, by Rachel M. Carmena.
<https://rachelcarmena.github.io/2018/12/12/how-to-teach-git.html>

Disclaimer (cont.)

- This lecture note is to give the basic idea of how Git works.
- How to use Git will be covered in the practical exercises that will be included in PBLMS.

"FINAL".doc



FINAL.doc!



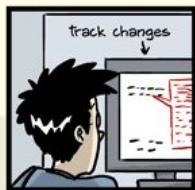
FINAL_rev.2.doc



FINAL_rev.6.COMMENTS.doc



FINAL_rev.8.comments5.
CORRECTIONS.doc



FINAL_rev.18.comments7.
corrections9.MORE.30.doc



FINAL_rev.22.comments49.
corrections.10. #@\$%WHYDID
ICOMETOGRADSCHOOL?????.doc

We've all been in this situation before.

It seems ridiculous to have multiple nearly-identical versions of the same document.

Motivation (cont.)

- It would be great if we could start with a **base version of the document** and then **save just the changes you made at each step of the way**.
- Some word processors let us deal with this a little better, such as **Microsoft Word's Track Changes**, **Google Docs' version history**, or **LibreOffice's Recording and Displaying Changes**.
- In terms of software source code, we would like to have this kind of feature as well. This is where **Version Control** comes in.

What is Version Control?

- 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.**
- **You can think of it as a tape:** if you rewind the tape and start at the base document, then you can play back each change and end up with your latest version.
- Using a Version Control System (VCS) generally means that **if you screw things up or lose files, you can easily recover.**

Typical Features in Version Control Systems

**It allows you to
revert selected files
back to a previous
state**

**Revert the entire
project back to a
previous state**

**Compare changes
over time**

**See who last
modified something
that might be
causing a problem**

**Who introduced an
issue and when**

And more.

In addition, you get all this for very
little overhead.

Types of Version Control Systems (VCS)

1

**Local Version Control
Systems**

2

**Centralised Version Control
Systems**

3

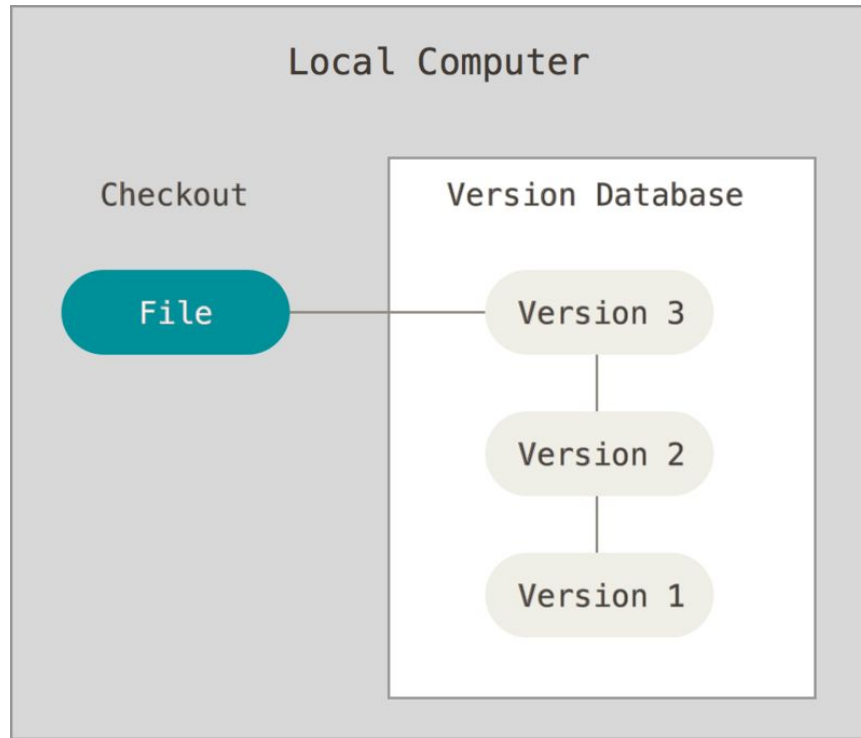
**Distributed Version Control
Systems**

Local Version Control Systems

- Many people's version-control method of choice **is to copy files into another directory** (perhaps a time-stamped directory, if they're clever).
- This approach is very common because it is so simple, but it is also **incredibly error prone**.
- It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.
- To deal with this issue, **programmers long ago developed local VCSs** that had a simple database that kept all the changes to files under revision control.

Local Version Control Systems (cont)

- One of the most popular VCS tools was a system called RCS, which is still distributed with many computers today.
- RCS works by keeping patch sets (that is, the differences between files) in a special format on disk; it can then re-create what any file looked like at any point in time by adding up all the patches.

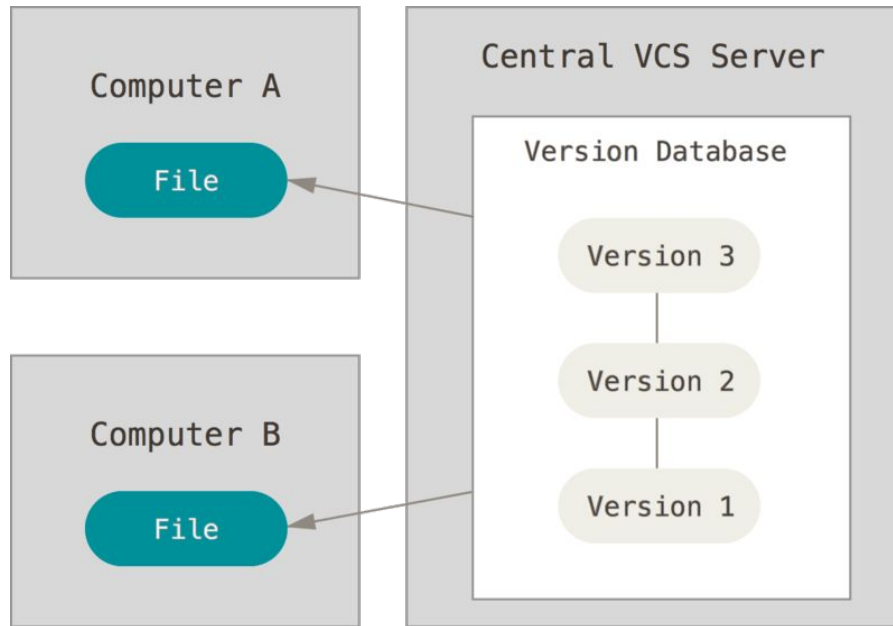


Centralized Version Control Systems

- 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 (such as CVS, Subversion, and Perforce) 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.

Centralized Version Control Systems (cont)

- This setup offers many advantages, especially **over local VCSs**.
- For example,
 - everyone knows to a certain degree what everyone else on the project is doing.
 - Administrators have fine-grained control over who can do what, and
 - it's far easier to administer a CVCS than it is to deal with local databases on every client.



Centralized Version Control Systems (cont)

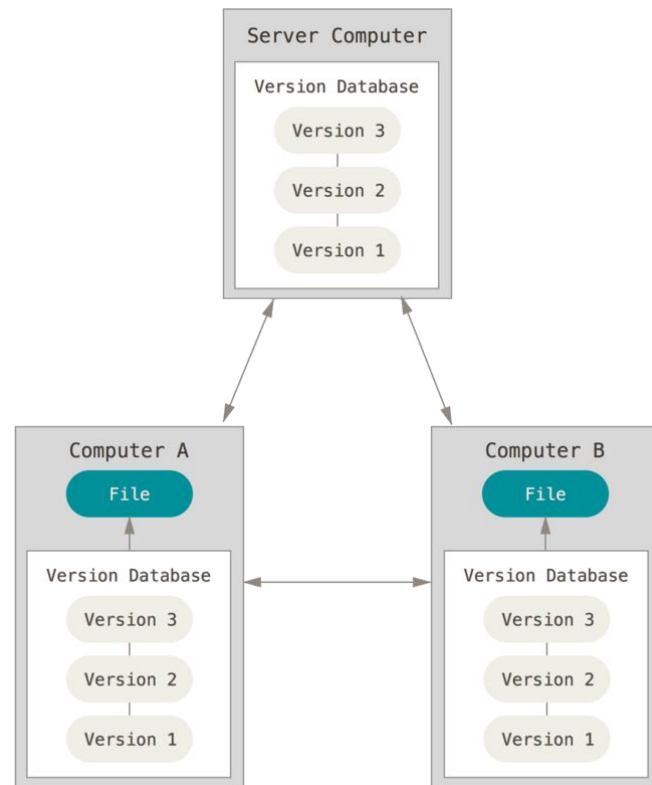
- However, **this setup also has some serious downsides.**
- The most obvious is **the single point of failure** that the centralized server represents.
- If that server goes down for an hour, then during that hour **nobody can collaborate at all or save versioned changes to anything they're working on.**
- If the hard disk the central database is on becomes corrupted, and proper backups haven't been kept, **you lose absolutely everything**—the entire history of the project except whatever single snapshots people happen to have on their local machines.
- Local VCS systems suffer from this same problem—whenever you have the entire history of the project in a single place, you risk losing everything.

Distributed Version Control Systems

- This is where Distributed Version Control Systems (DVCSs) step in.
- There are a few DVCS available such as Git, Mercurial, Bazaar or Darcs.
- But in this module we will concentrate on using **Git**.

Distributed Version Control Systems (cont.)

- In a DVCS,
 - clients don't just check out the latest snapshot of the files;
 - rather, **they fully mirror the repository, including its full history.**
 - Thus, 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.
 - Every clone is really a full backup of all the data.



What is Git?

THIS IS GIT. IT TRACKS COLLABORATIVE WORK
ON PROJECTS THROUGH A BEAUTIFUL
DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL
COMMANDS AND TYPE THEM TO SYNC UP.
IF YOU GET ERRORS, SAVE YOUR WORK
ELSEWHERE, DELETE THE PROJECT,
AND DOWNLOAD A FRESH COPY.

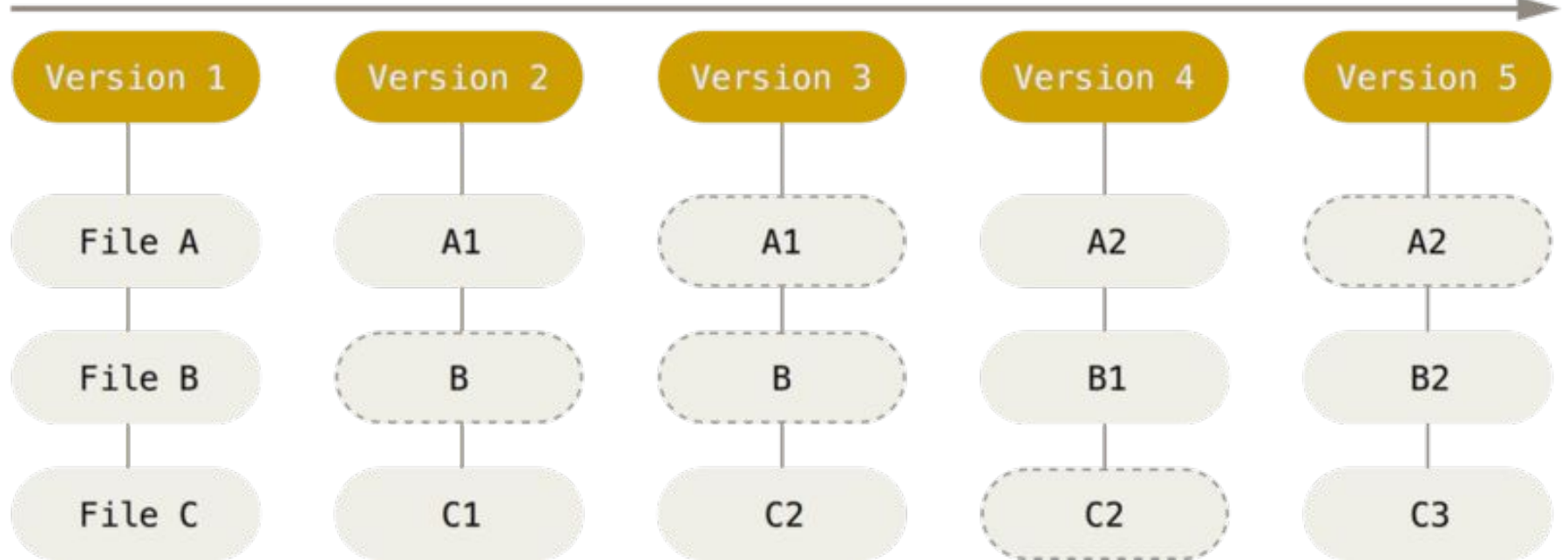


What is Git? (cont.)

- Git is one of Distributed Version Control Systems technology.
- Git **store its data like a series of snapshots** of a miniature filesystem.
- Every time you commit, or save the state of your project, Git basically **takes a picture of what all your files look like at that moment and stores a reference to that snapshot.**
- To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored.
- Git thinks about its data more like a stream of snapshots.

Git Snapshots

Checkins Over Time

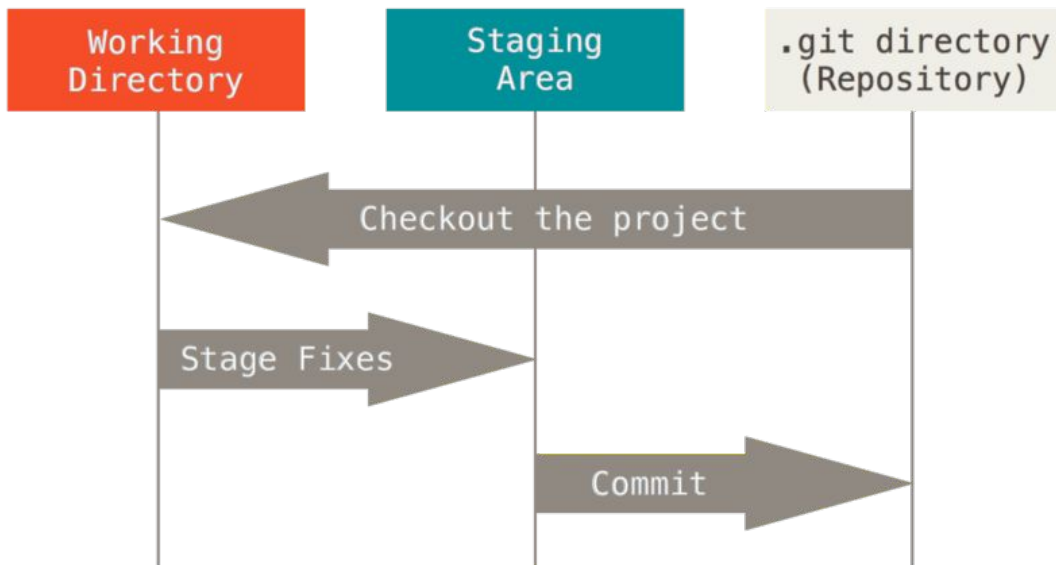


The Three States

- IMPORTANT to remember about Git.
- Git has **three main states** that your files can reside in: **modified**, **staged**, and **committed**:
 - a. Modified means **that you have changed the file but have not committed it to your database yet.**
 - b. Staged means **that you have marked a modified file in its current version to go into your next commit snapshot.**
 - c. Committed means **that the data is safely stored in your local database.**

The Three Main Sections

- This leads us to the **three main sections** of a Git project: **the working tree**, **the staging area**, and **the Git directory**.



The Three Main Sections (cont.)

1

The 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.

2

The staging area

is a file, generally contained in your Git directory, that stores information about what will go into your next commit. Its technical name in Git parlance is the "index", but the phrase "staging area" works just as well.

3

The 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.

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.

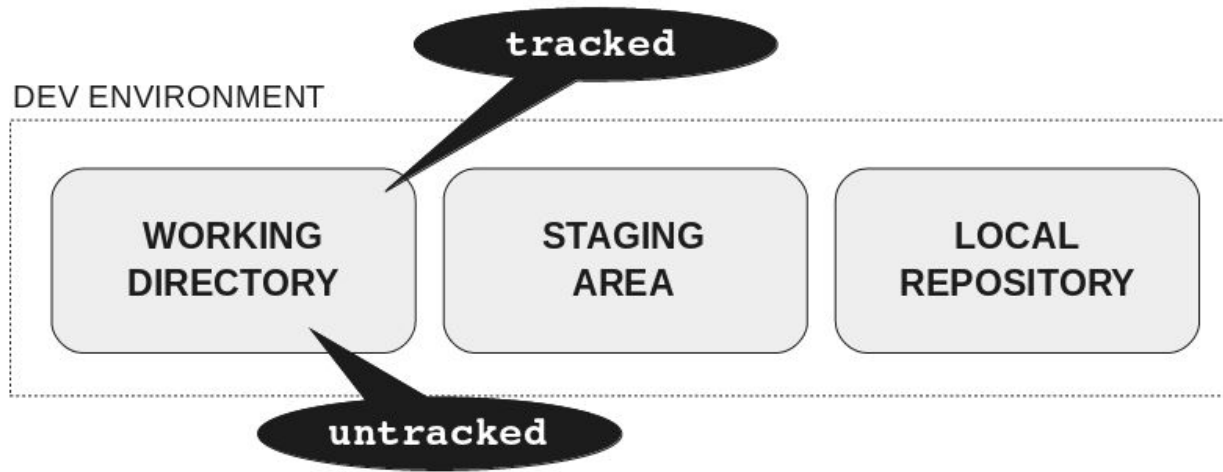
The basic Git workflow



Making changes in the working directory

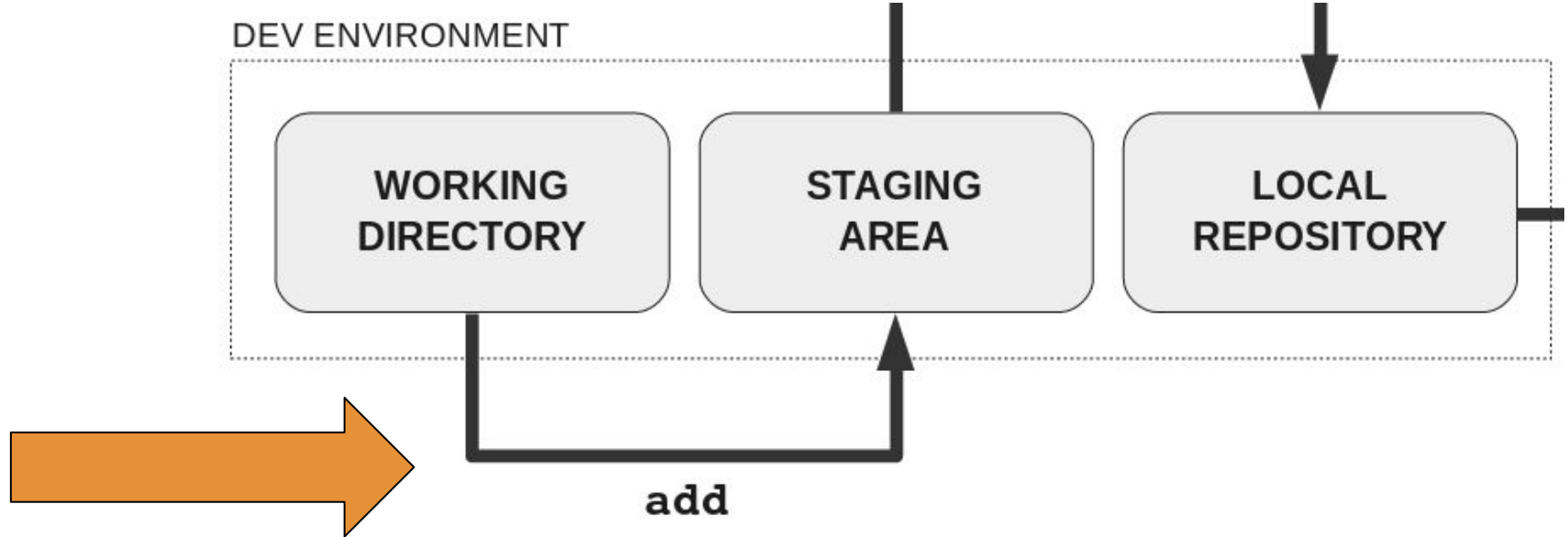
There are 2 types of files in the working directory:

1. **Tracked**: files that Git knows about.
2. **Untracked**: files that have still not been added, so Git doesn't know about.



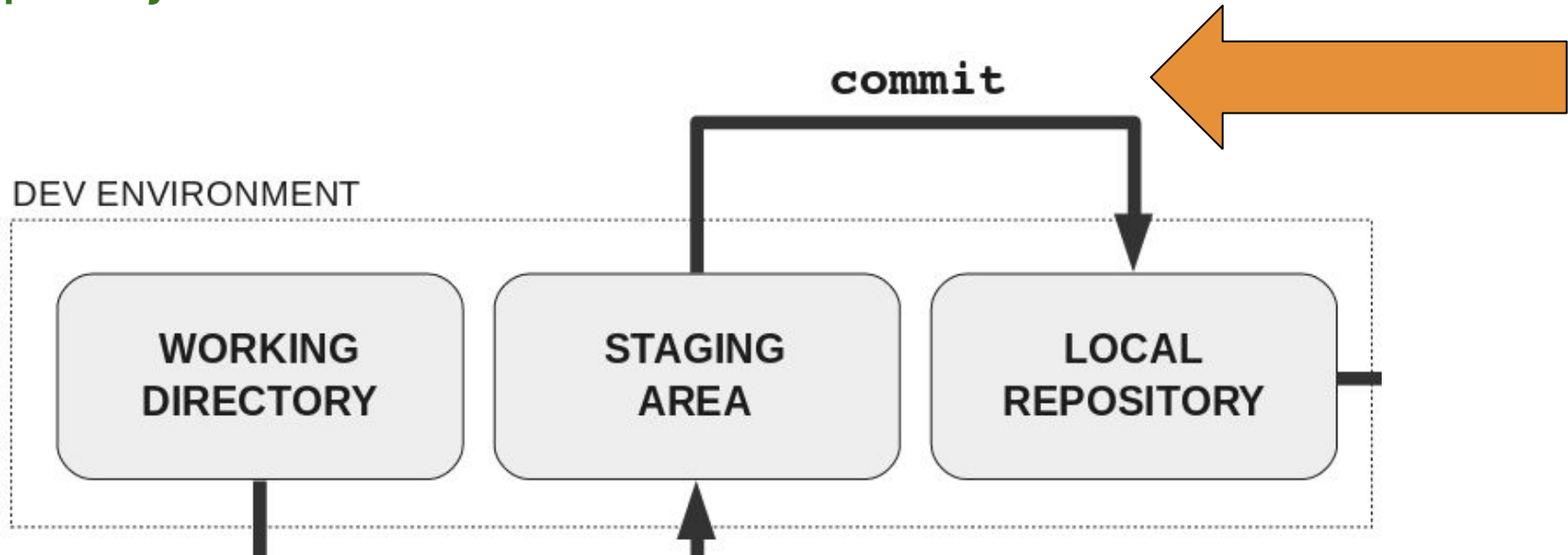
Staging the changes

- As changes are ready in the working directory, **they must be added in the staging area.**



Stores the snapshot

- When there is a set of changes with a single purpose in the staging area, it's the time to **create a commit with a message about that purpose in the local repository**.



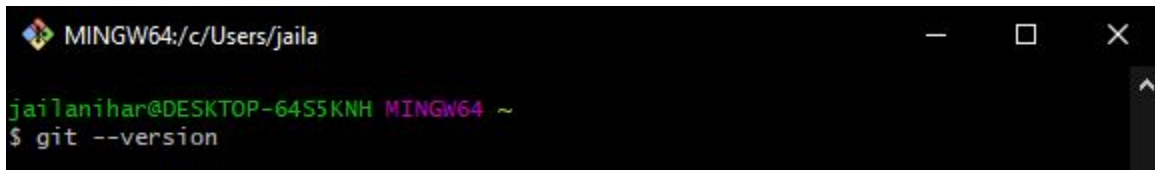
Installing Git

- Go to this link to learn how to install Git to your computer.
<https://www.linode.com/docs/guides/how-to-install-git-on-linux-mac-and-windows/>
- Or you can download the PDF version of the page provided under the module Supplementary Materials in PBLMS titled “How to Install Git on Linux, Mac or Window”.

NOTE: For Windows installation, there are some options that are not shown in the guide. You should be fine if you leave it as default.

Verifying your git installation

- Open up your **terminal**.
 - Windows: Run **Git Bash**
- Verify that you have git installed by typing the following command in your terminal and press Enter.



```
MINGW64:/c/Users/jaila
jailanihar@DESKTOP-64S5KNH MINGW64 ~
$ git --version
```

- It should print the git version number that you installed. That means your installation was a success.



```
MINGW64:/c/Users/jaila
jailanihar@DESKTOP-64S5KNH MINGW64 ~
$ git --version
git version 2.30.0.windows.1
```

Remote Repository

- Git really comes into its own when we begin **to collaborate with other people**.
- We already have most of the machinery we need to do this; the only thing **missing is to copy changes from one repository to another**.
- Git allow us to move work between any two repositories.
- In practice, though, it's easiest to **use one copy as a central hub**, and to keep it on the web rather than on someone's laptop.
- Most programmers use hosting services like **GitHub**, **Bitbucket** or **GitLab** to hold those master copies.
- We will **concentrate on GitHub** for this module.

Remote Repository (cont.)

SERVER

**REMOTE
REPOSITORY**

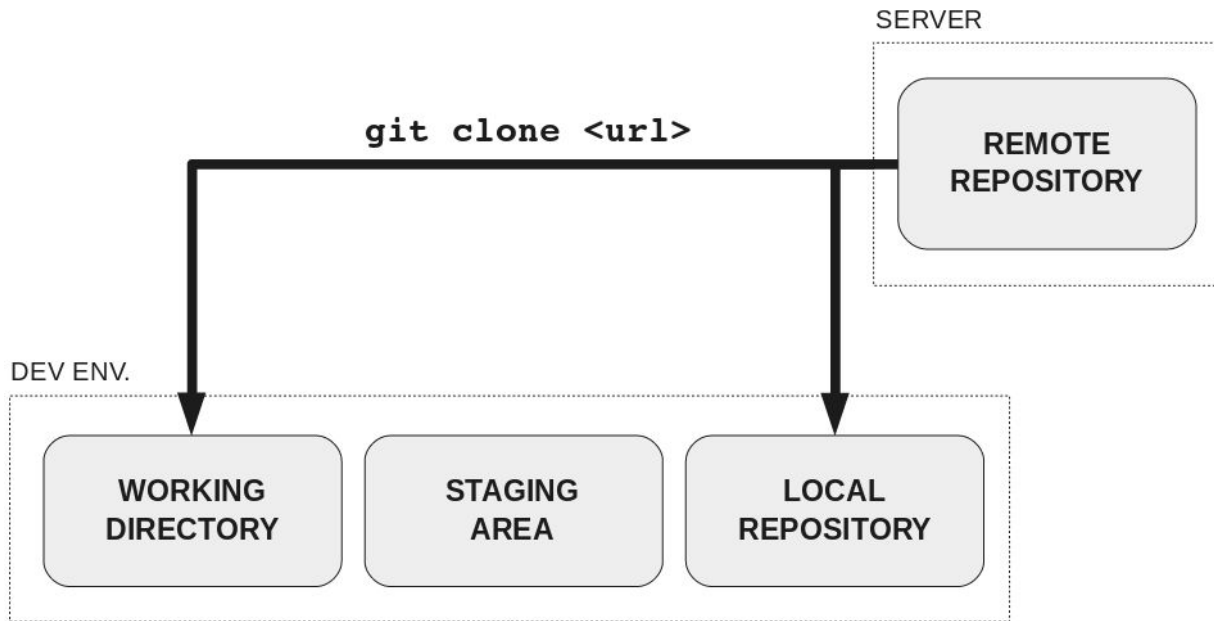
DEV ENVIRONMENT

**WORKING
DIRECTORY**

**STAGING
AREA**

**LOCAL
REPOSITORY**

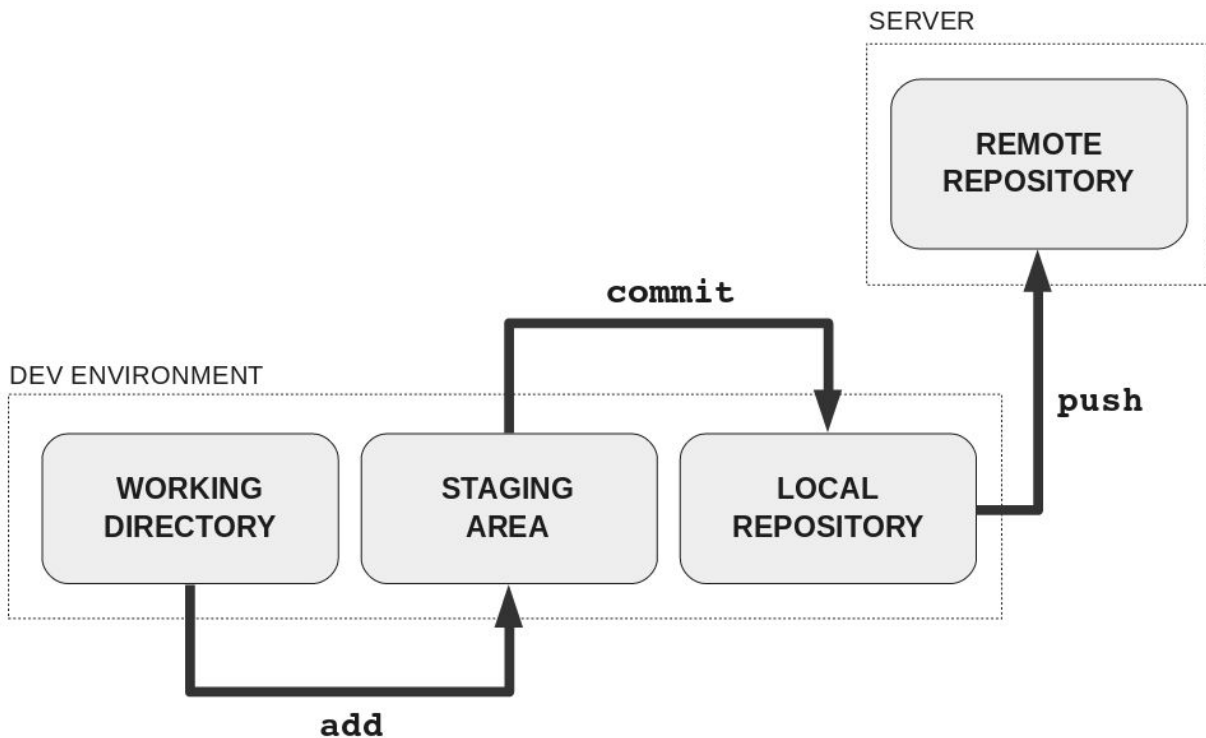
Cloning a repository



When cloning a repository, **the data from the remote repository travel to 2 areas:**

- Working directory
- Local repository

Updating the remote repository



- It still follows the basic Git workflow.
- When there are one or several commits in the local repository ready to be shared with the rest of the world, **they must be pushed to the remote repository**.

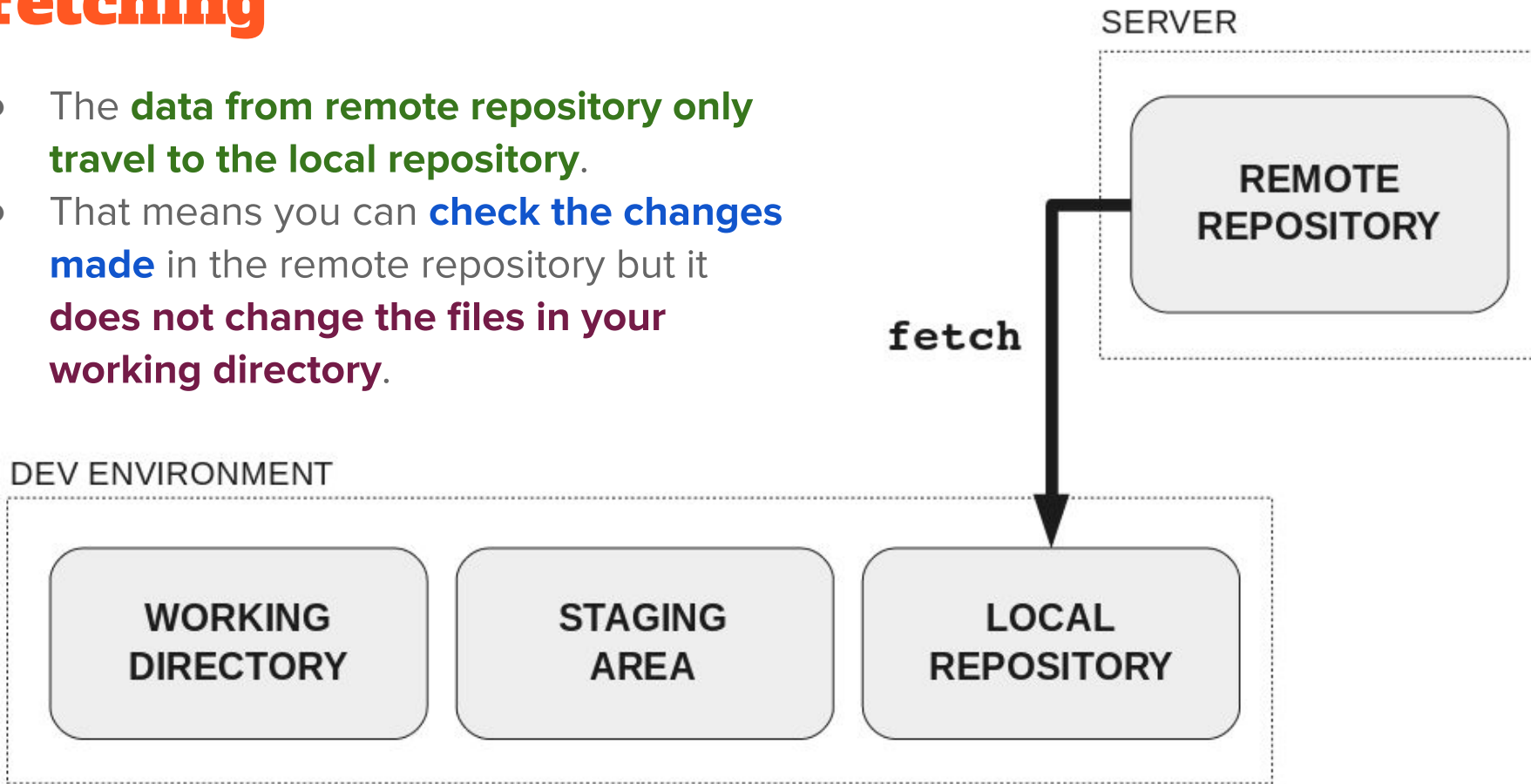
Updating the development environment

There are **two ways** to update the development environment

1. Fetching
2. Pulling

Fetching

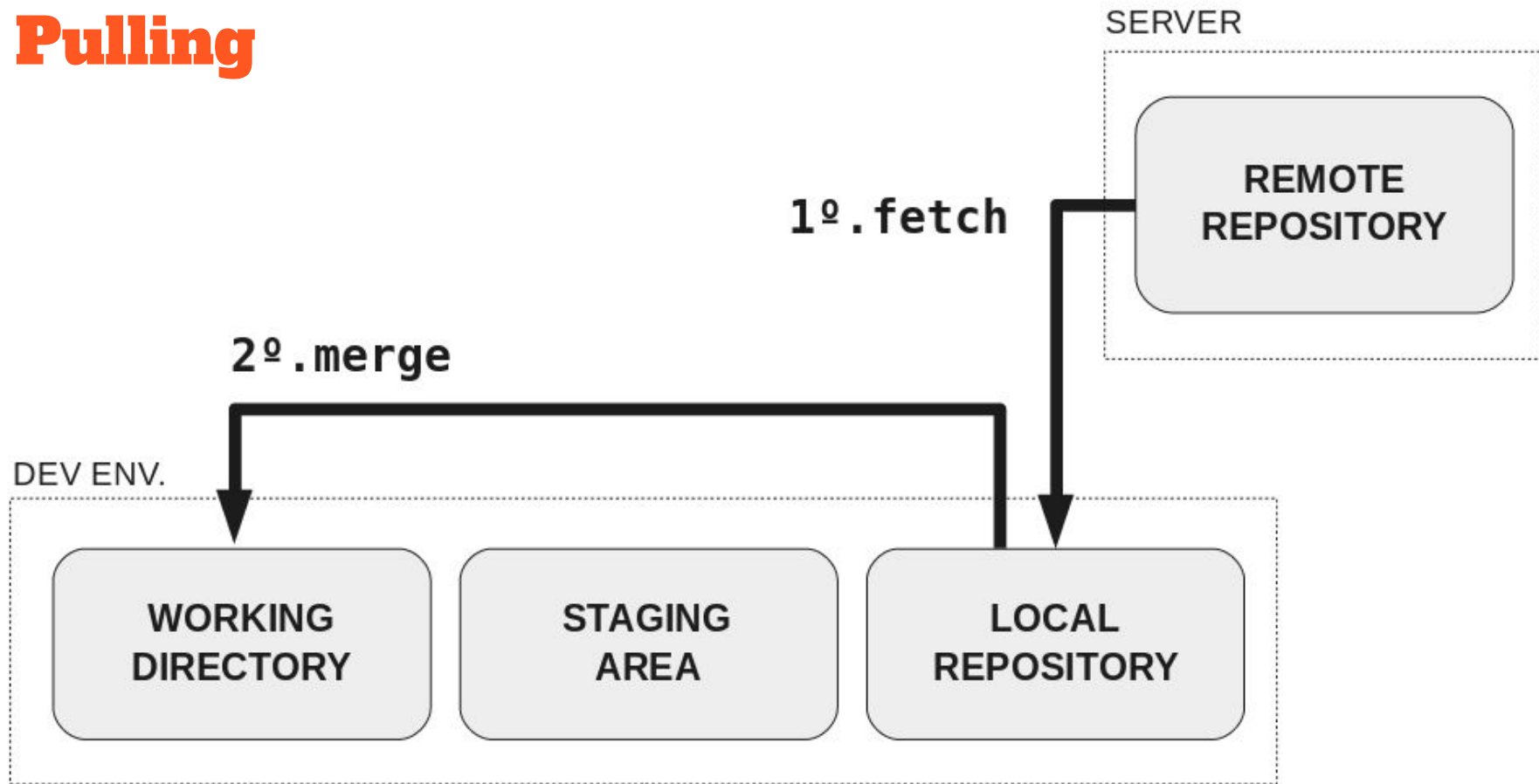
- The **data from remote repository only travel to the local repository.**
- That means you can **check the changes made** in the remote repository but it **does not change the files in your working directory.**



Pulling

- When executing git pull, **the data from remote repository travel to 2 areas:**
 - 1. To local repository: fetch**
 - 2. To working directory: merge**
- That means the **changes made in the remote repository will be reflected to your working directory.**

Pulling



Useful Git features and concepts not covered

- Git Branching
 - Branches in a Nutshell: <https://git-scm.com/book/en/v2/Git-Branching-Branched-in-a-Nutshell>
 - Basic Branching and Merging: <https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging>
 - Branch Management: <https://git-scm.com/book/en/v2/Git-Branching-Branch-Management>
 - Branching Workflows: <https://git-scm.com/book/en/v2/Git-Branching-Branching-Workflows>
 - Remote Branches: <https://git-scm.com/book/en/v2/Git-Branching-Remote-Branches>
 - Rebasing: <https://git-scm.com/book/en/v2/Git-Branching-Rebasing>
- Stashing and Cleaning:
<https://git-scm.com/book/en/v2/Git-Tools-Stashing-and-Cleaning>