Git and Github

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

fit@hcmus

What is Version Control

- Version control systems (VCSs) are tools used to track changes to source code (or other collections of files and folders).
  - Help maintain a history of changes
  - Facilitate collaboration.
  - 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.
- Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.

What is Version Control

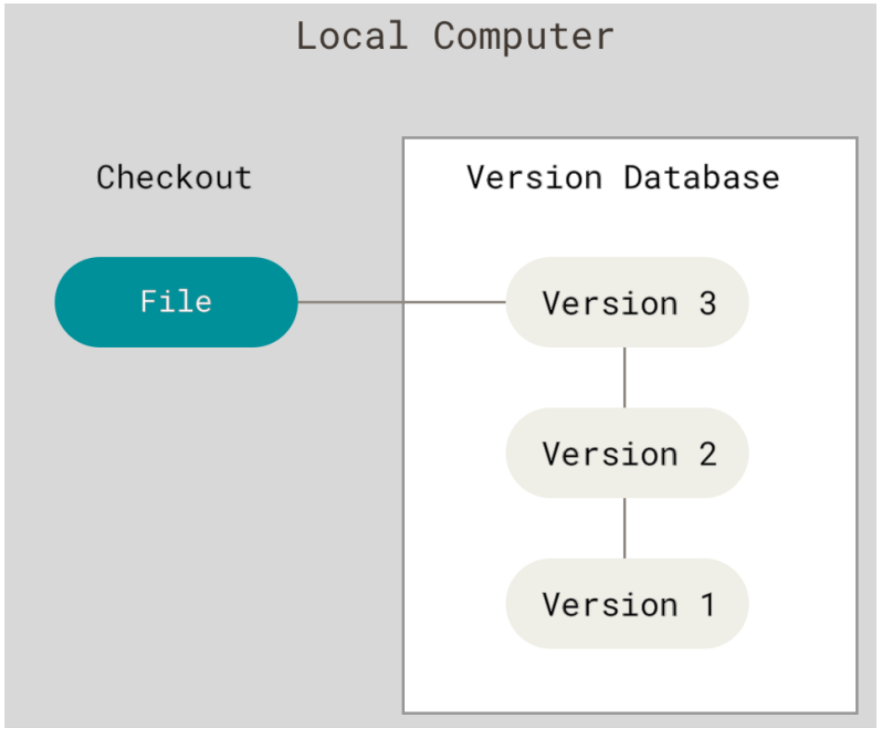
- Modern VCSs also let you easily (and often automatically) answer questions like:
  - Who wrote this module?
  - When was this particular line of this particular file edited? By whom? Why was it edited?
  - Over the last 1000 revisions, when/why did a particular unit test stop working?

Local Version Control Systems

- Many people's version-control method of choice is to copy files into another directory
  - Simple, but error prone
- Programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.

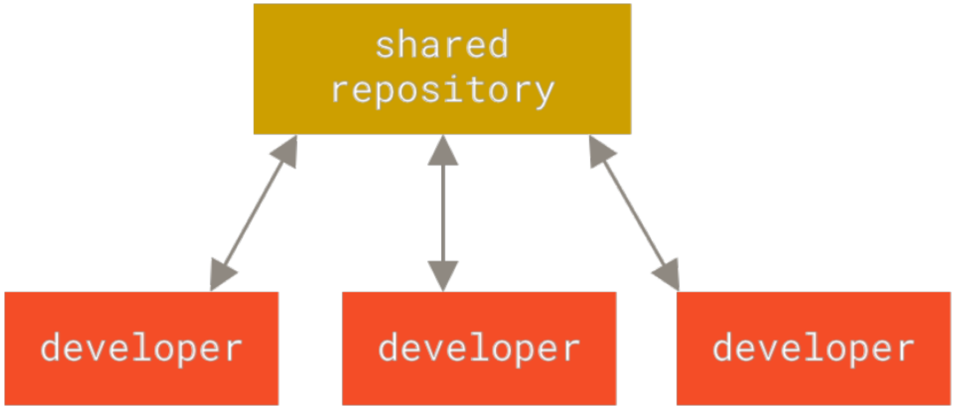
Local Version Control Systems

- Ex: RCS
- Keeping **patch sets** (the differences between files) in a special format on disk
- Can then re-create what any file looked like at any point in time by adding up all the patches



Centralized Version Control Systems

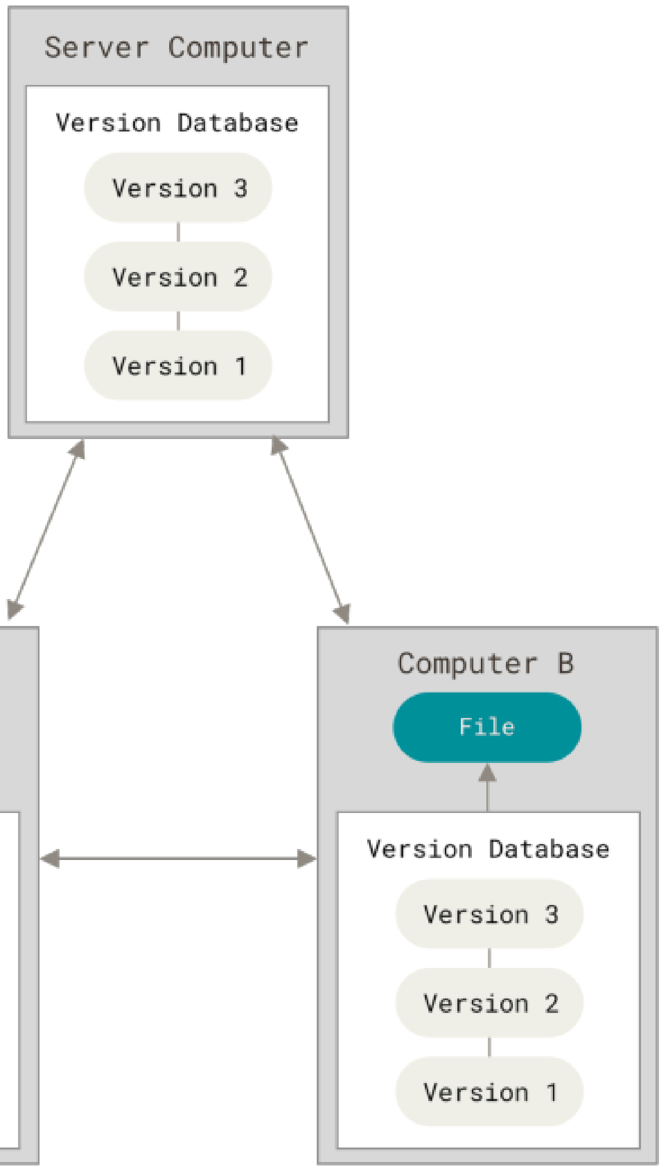
- To **collaborate** with developers on other systems
- These systems (*CVS*, *Subversion*, and *Perforce*) have a single server that contains all the versioned files, and a number of clients that **check out** files from that central place.



Centralized Version Control Systems

- **Advantages**
  - 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 local databases on every client.
- **Drawback**
  - *Single point of failure*: 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

Distributed Version Control Systems



Distributed Version Control Systems

- In a DVCS (such as **Git**, Mercurial, Bazaar or Darcs), clients don’t just check out the latest snapshot of the files; rather, they *fully mirror the repository*, including its full history
- Every clone is really a full backup of all the data.
- Can collaborate with different groups of people in different ways simultaneously within the same project - Hierarchical models

A Short History of Git

- ☐ The Linux kernel is an open source software project of fairly large scope.
- ☐ 1991–2002: changes to the software were passed around as patches and archived files.
- ☐ In 2002, the Linux kernel project began using a DVCS called BitKeeper.
- ☐ In 2005, the tool’s free-of-charge status was revoked.

➔ Linux community (and in particular Linus Torvalds) to develop their own tool based on some of the lessons they learned while using BitKeeper

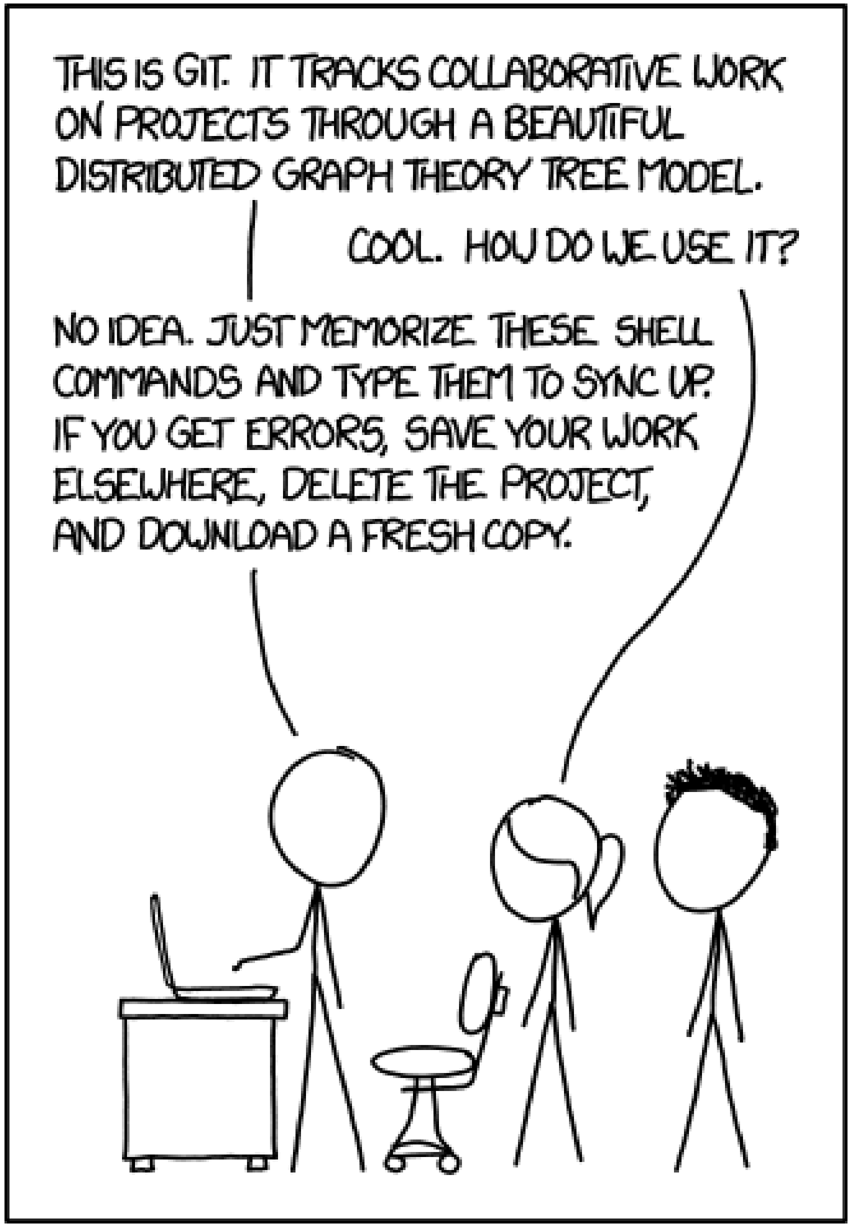
A Short History of Git

- Some of the goals of the new system were as follows:
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)
- Since its birth in 2005, Git has evolved and matured to be easy to use and yet retain these initial qualities.
- It’s amazingly fast, it’s very efficient with large projects, and it has an incredible branching system for non-linear development

What is Git

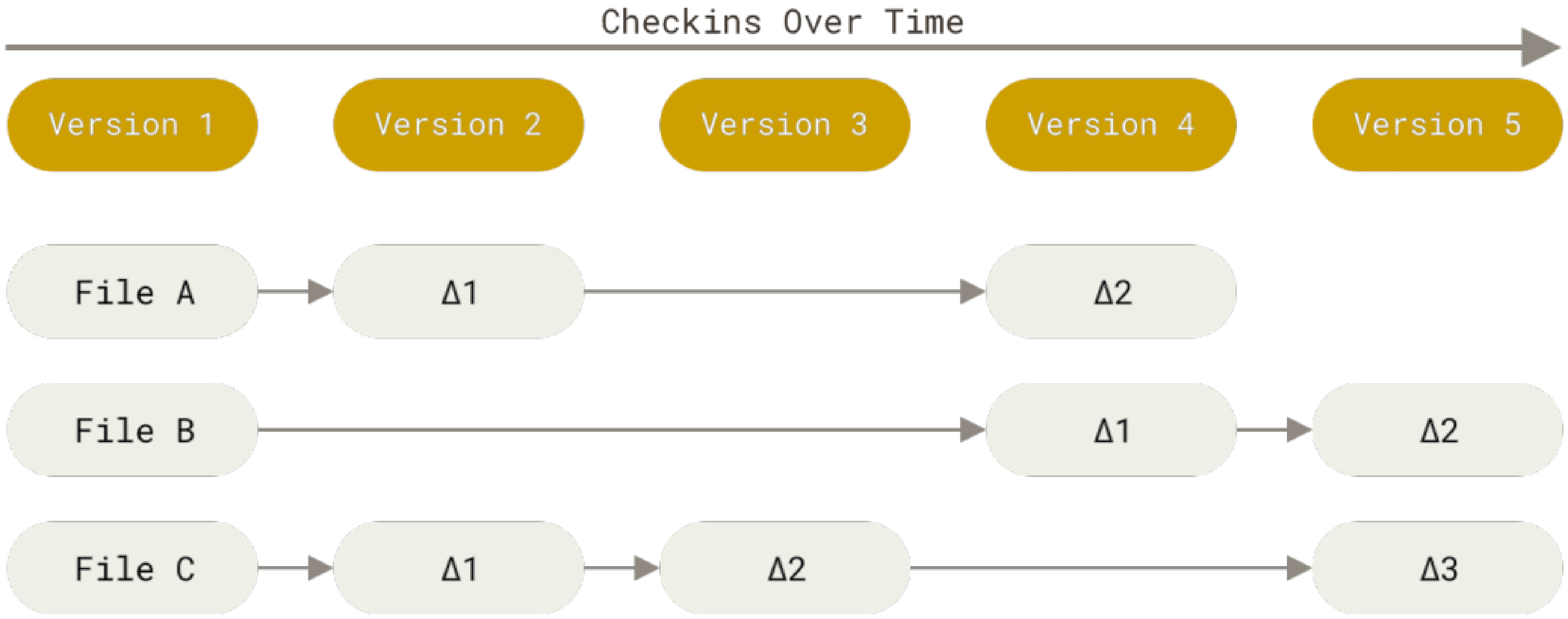
- **It is a free, high-quality distributed version control system suitable for tracking modifications in source code in software development**
- Git is the tool created by Linux Torvalds to do version control and to support collaboration, originally for Linux source code when it was published and many people joined to contribute to it
- Now Git is used widely in companies; when you work for companies in the future, it’s likely that you will have to use Git

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.

Delta-based version control

- Other systems (CVS, Subversion, Perforce) store information as a list of file-based changes.
  - Store as a set of files and the changes made to each file over time (delta-based version control)

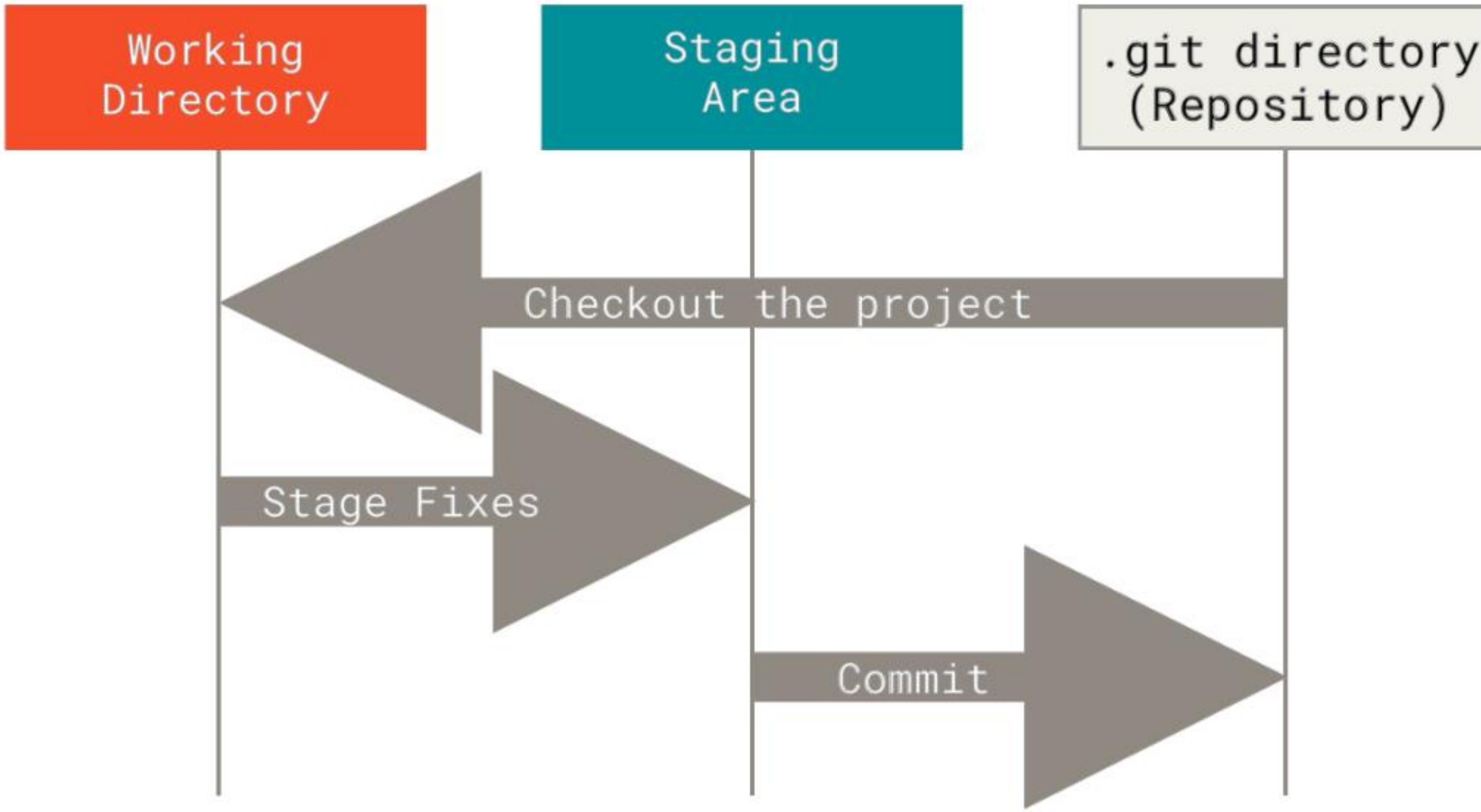


Snapshot

- Git models the history of a collection of files and folders within some top-level directory as a series of snapshots.
- A file is called a “blob”
- A directory is called a “tree”, and it maps names to blobs or trees

```
<root> (tree)
|
+- foo (tree)
|  |
|  + bar.txt (blob, contents = "hello world")
|
+- baz.txt (blob, contents = "git is wonderful")
```

Basic Git: Three stage



Basic Git: Three stage

- Git has three main states that your files can reside in: modified, staged, and committed:
- **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.
- **Committed** means that the data is safely stored in your local database.
- Three main sections of a Git project: the **working tree**, the **staging area**, and the **Git directory**

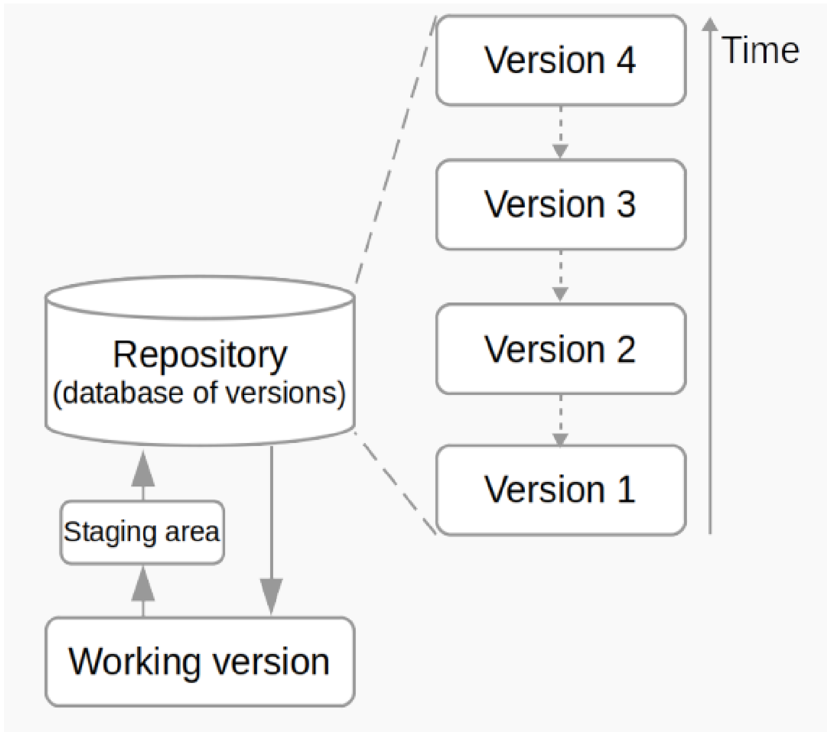
Basic Git: Three stage

- 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.
- The **staging area** is a file, generally contained in your Git directory, that stores information about what will go into your next commit.
- 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.

Basic Git workflow

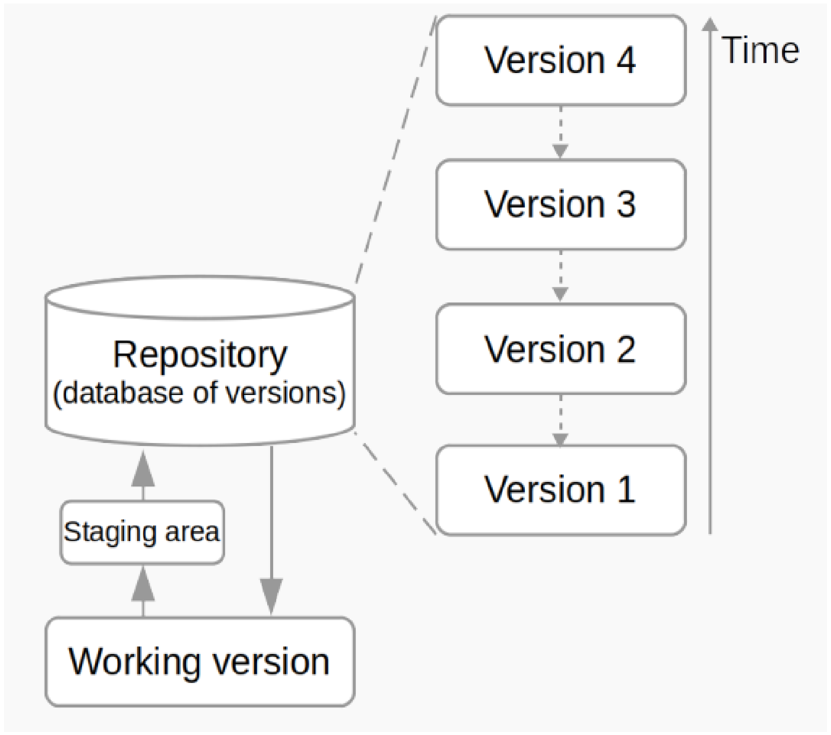
- 1. You modify files in your working tree.
- 2. You selectively stage just those changes you want to be part of your next commit, which adds only those changes to the staging area.
- 3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

How git work



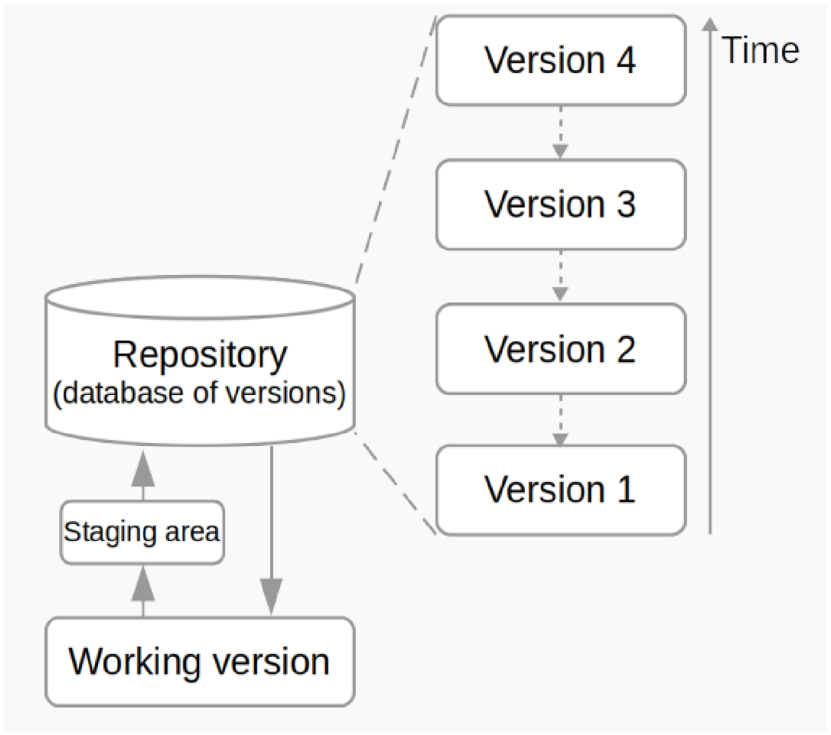
- ☐ Version 4 pointing to version 3 means: version 4 is created by modifying version 3
- ☐ Working version (working directory/tree) is a version we can modify to create a new version

How git work



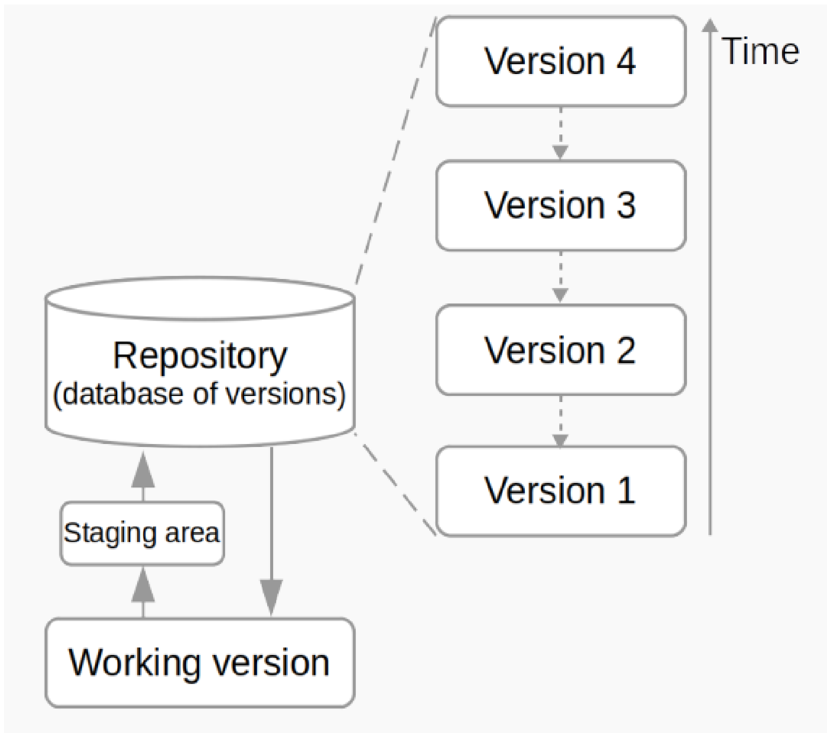
- After modifying the working version, we will save the new version to the repository
- Why do we need to go through the staging area?
  - Because we may make many changes to the working version, but we only want to include some changes to the new version

How git work



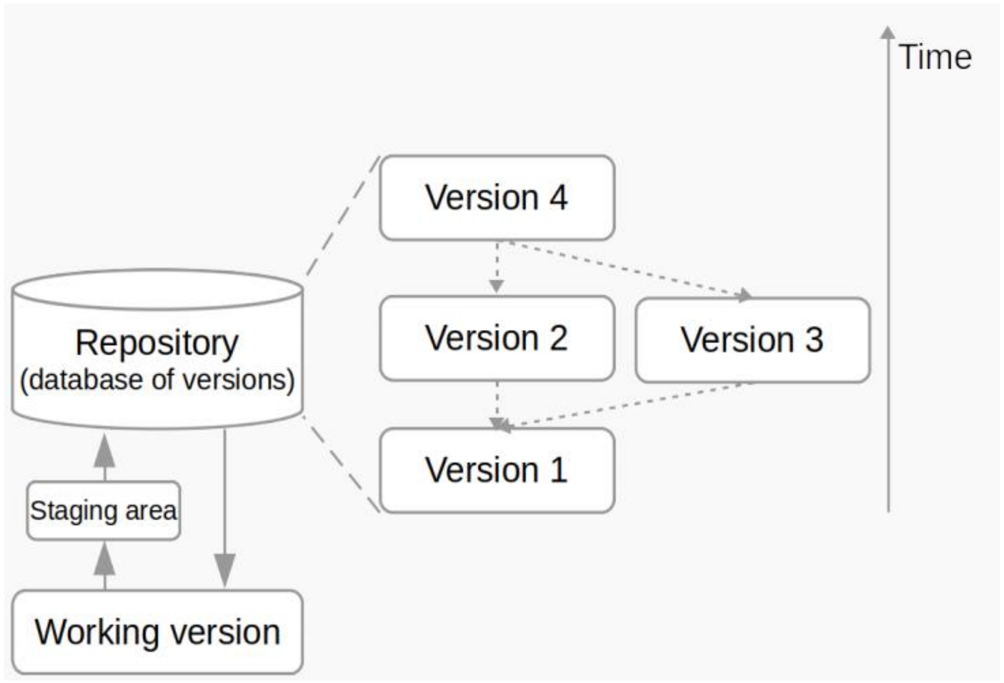
- ☐ For future we can review versions easily, we need to think: new version = old version + what changes? (in general, a new version should include neither too much nor too little changes)

How git work



- Versions can be created in a simple way: version 2 is created from version 1, version 3 is created from version 2, … → there is only one branch

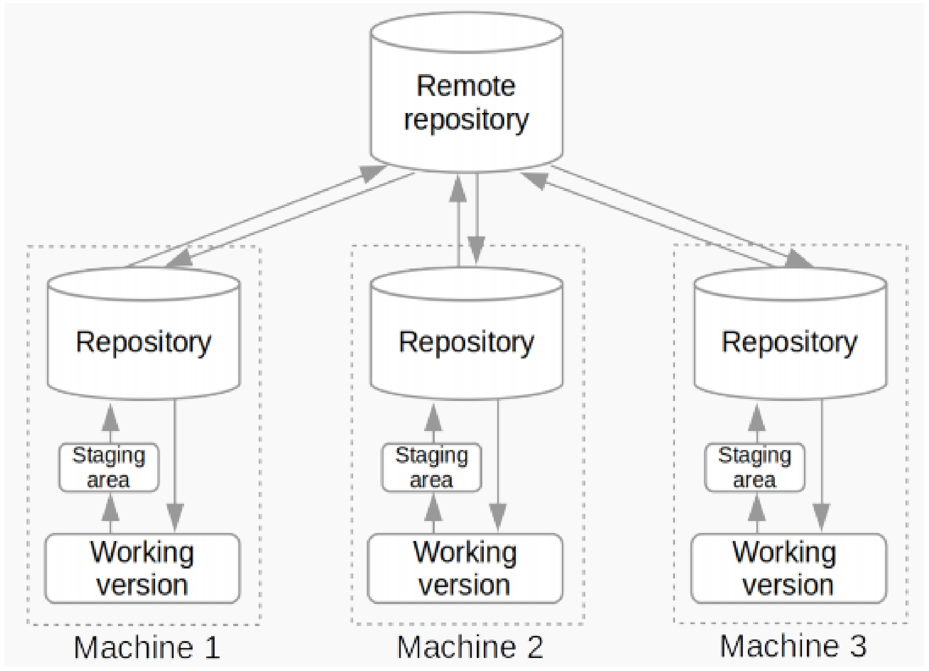
How git work



- ☐ Git allow creating branches from the main branch; e.g., we are coding an app, and we want to code and experiment a feature without interfering with the current code in the main branch

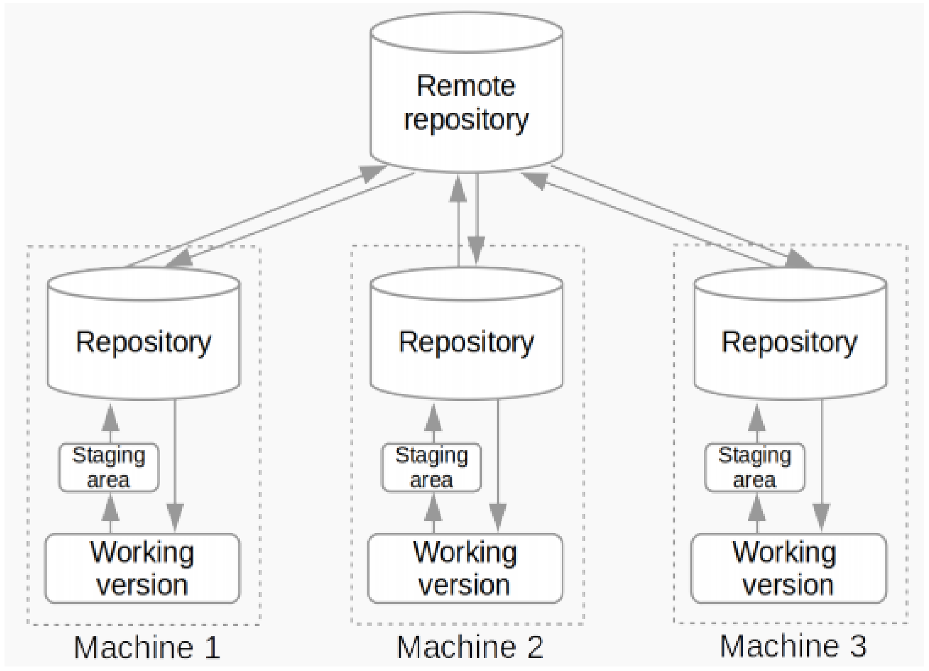
➔Create a second branch to code and experiment; when everything is OK we can merge this branch into the main branch, otherwise we can discard this branch

How git work



- **Git** supports collaboration through a remote repository (**Github** is a service providing remote repositories for Git)
- When working solo, we may still want to use remote repositories to backup data, to access from other computers

How git work



- ☐ Git is a distributed version control tool: each member in a team will have a local repository
- ☐ With central version control tools, data is stored in one place — the remote repository; if data in this place is loss then …

BASIC GIT

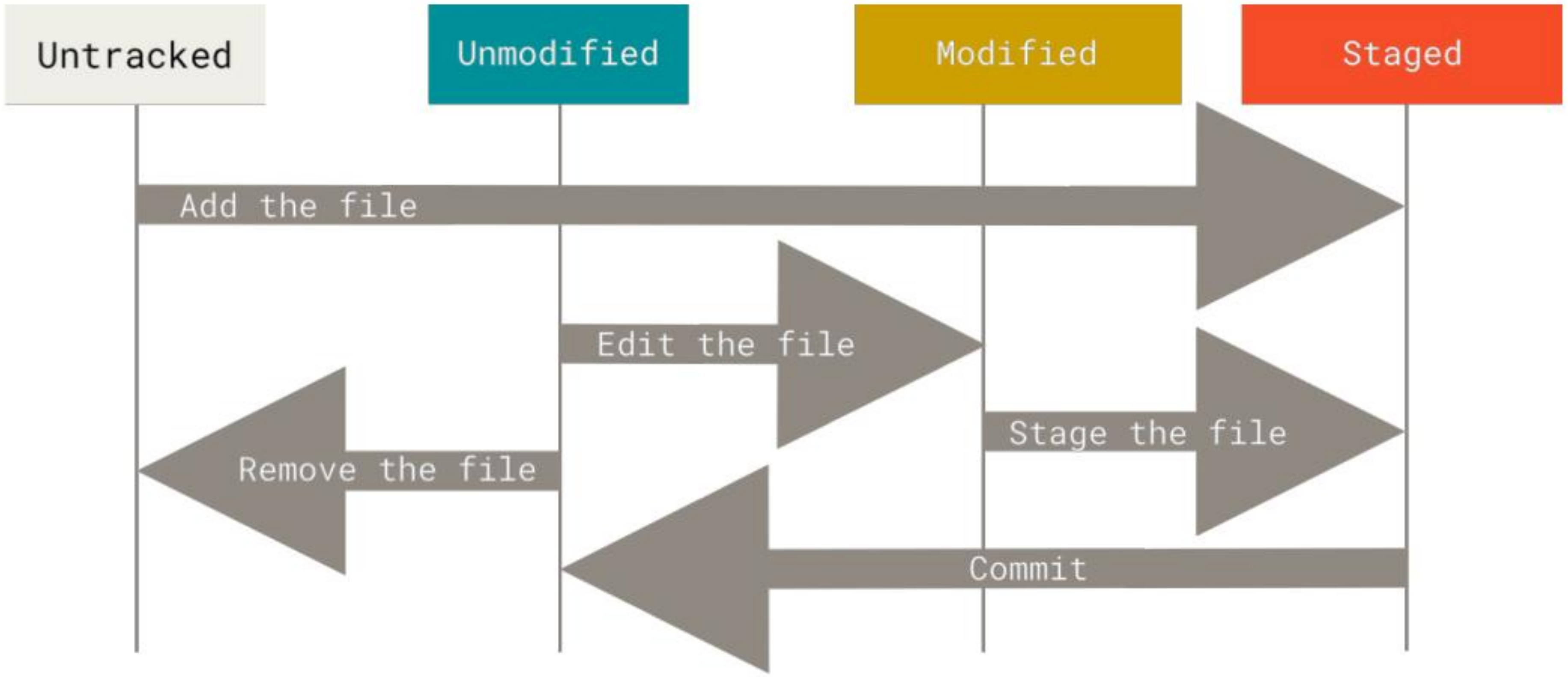
Git Repository

- You typically obtain a Git repository in one of two ways:
- 1. You can take a local directory that is currently not under version control, and turn it into a Git repository, or
- 2. You can clone an existing Git repository from elsewhere.

Initializing a Repository

- Existing Directory: go to that project’s directory:
  - $ git init
- Cloning an Existing Repository:
  - $ git clone https://github.com/libgit2/libgit2

Initializing a Repository



Git basic command

- git help <command>: get help for a git command
- git init: creates a new git repo, with data stored in the .git directory
- git status: tells you what’s going on
- git add <filename>: adds files to staging area
- git commit: creates a new commit
  - Write good commit messages!
  - Even more reasons to write good commit messages!
- git log: shows a flattened log of history
- git log --all --graph: visualizes history as a DAG
- git diff <filename>: show changes you made relative to the staging area
- git diff <revision> <filename>: shows differences in a file between snapshots
- git checkout <revision>: updates HEAD and current branch

Branching and merging

- git branch: shows branches
- git branch <name>: creates a branch
- git checkout -b <name>: creates a branch and switches to it
- same as git branch <name>; git checkout <name>
- git merge <revision>: merges into current branch
- git mergetool: use a fancy tool to help resolve merge conflicts
- git rebase: rebase set of patches onto a new base

Remotes

- git remote: list remotes
- git remote add <name> <url>: add a remote
- git push <remote> <local branch>:<remote branch>: send objects to remote, and update remote reference
- git branch --set-upstream-to=<remote>/<remote branch>: set up correspondence between local and remote branch
- git fetch: retrieve objects/references from a remote
- git pull: same as git fetch; git merge
- git clone: download repository from remote

Undo

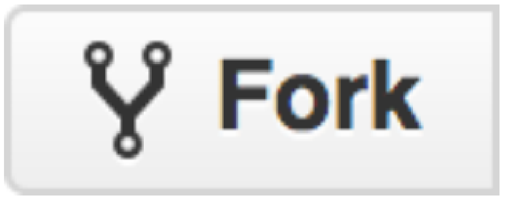
- git commit --amend: edit a commit’s contents/message
- git reset HEAD <file>: unstage a file
- git checkout -- <file>: discard changes

What is Github

- It is a web-based Git repository. This hosting service has cloud-based storage. GitHub offers all distributed version control and source code management functionality of Git while adding its own features. It makes it easier to collaborate using Git.
- Additionally, GitHub repositories are open to the public. Developers worldwide can interact and contribute to one another’s code, modify or improve it, making GitHub a networking site for web professionals. The process of interaction and contribution is also called social coding.

Forking Projects

- If you want to contribute to an existing project to which you don't have push access, you can "fork" the project.
- When you "fork" a project, GitHub will make a copy of the project that is entirely yours; it lives in your namespace, and you can push to it



Forking Projects

- This way, projects don’t have to worry about adding users as collaborators to give them push access.
- People can fork a project, push to it, and contribute their changes back to the original repository by creating what’s called a Pull Request
- This opens up a discussion thread with code review, and the owner and the contributor can then communicate about the change until the owner is happy with it, at which point the owner can merge it in
- To fork a project, visit the project page and click the “Fork” button at the top-right of the page.
- After a few seconds, you’ll be taken to your new project page, with your own writeable copy of the code

The GitHub Flow

1. Fork the project.
2. Create a topic branch from master.
3. Make some commits to improve the project.
4. Push this branch to your GitHub project.
5. Open a Pull Request on GitHub.
6. Discuss, and optionally continue committing.
7. The project owner merges or closes the Pull Request.
8. Sync the updated master back to your fork.

Important reminder

- Use Git & Github to do version control and data backup for your code files
- **Note**: you should be thoughtful about whether you should make repositories in Github public

Reference

- https://missing.csail.mit.edu/2020/version-control/
- https://git-scm.com/book/en/v2