T03-01. Git

Require

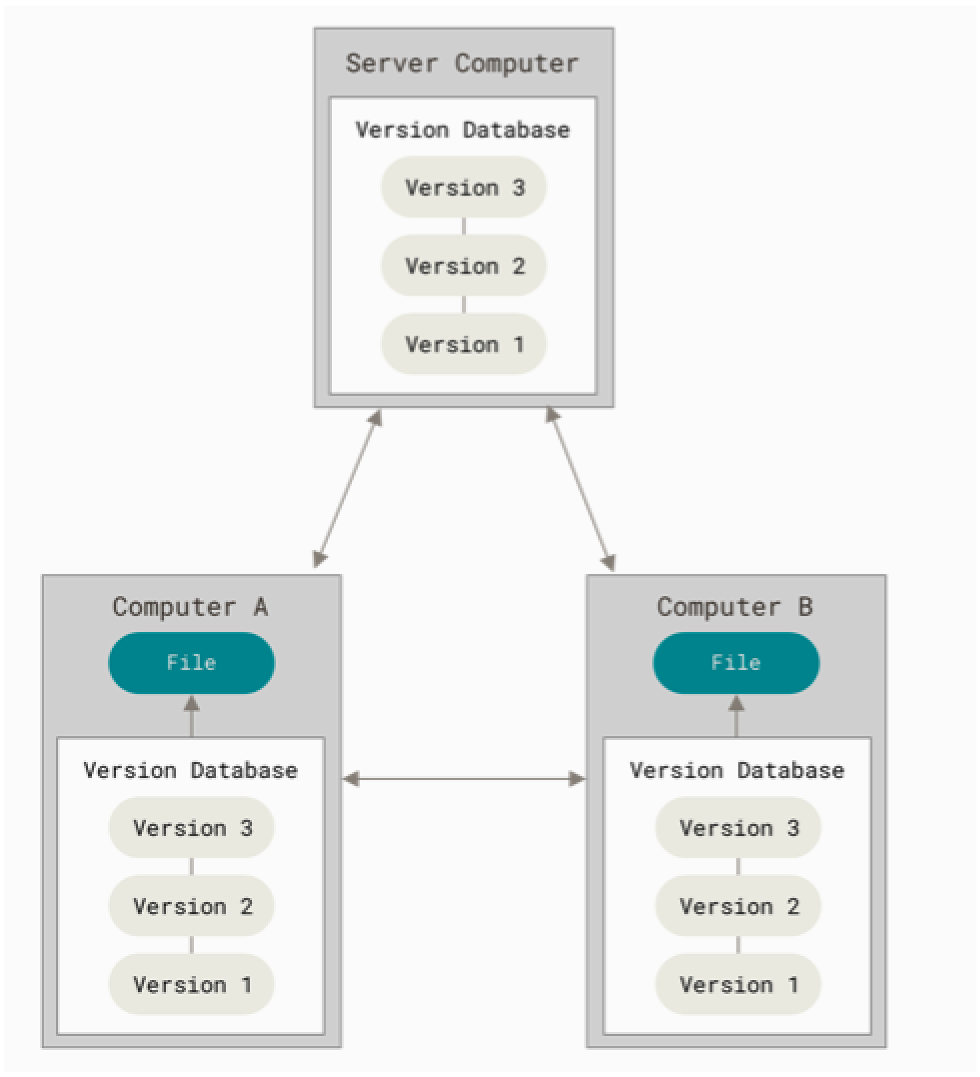
- Estimate time: 1 day
- Raise any questions when problem occurs

Concept

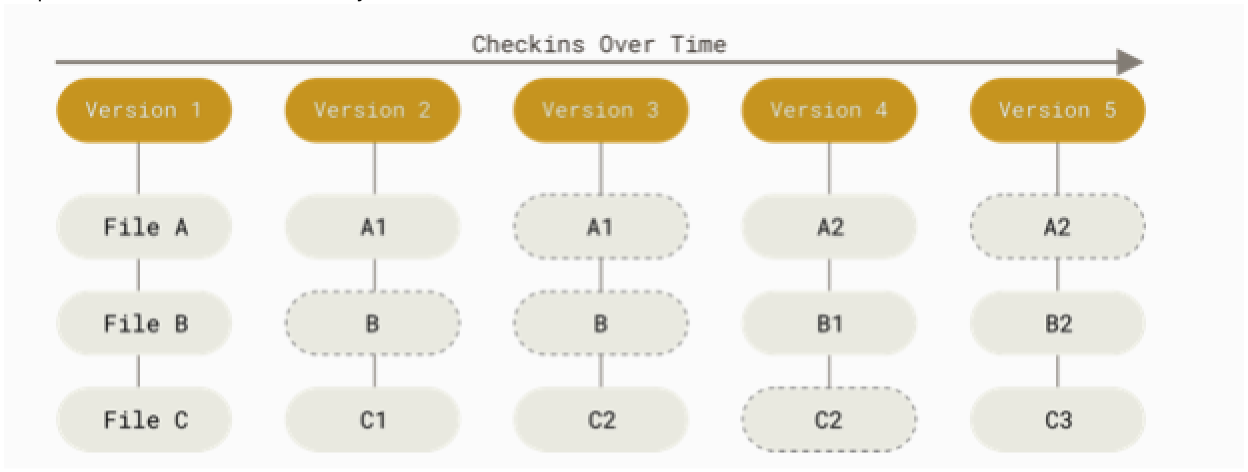
Git

What is Git?

- Git is a tool used for source code management. It is a free and open-source version control system used to handle projects efficiently. Git is used to tracking changes in the source code, enabling multiple developer to work together.
  - 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.

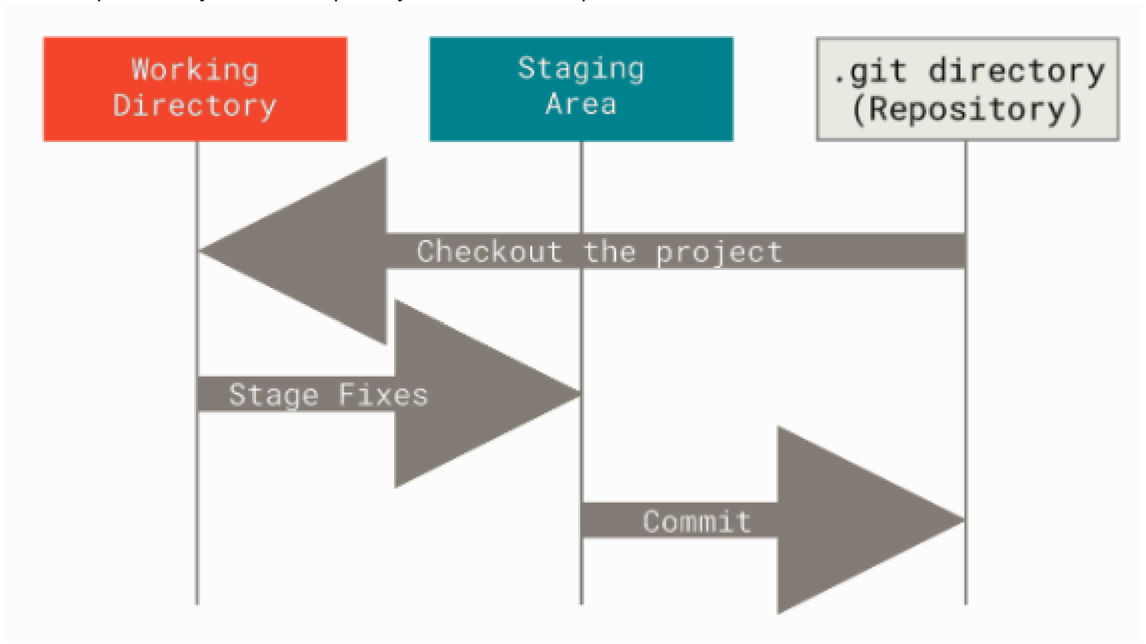


- How to Git working?
  - With Git, every time you commit or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored.

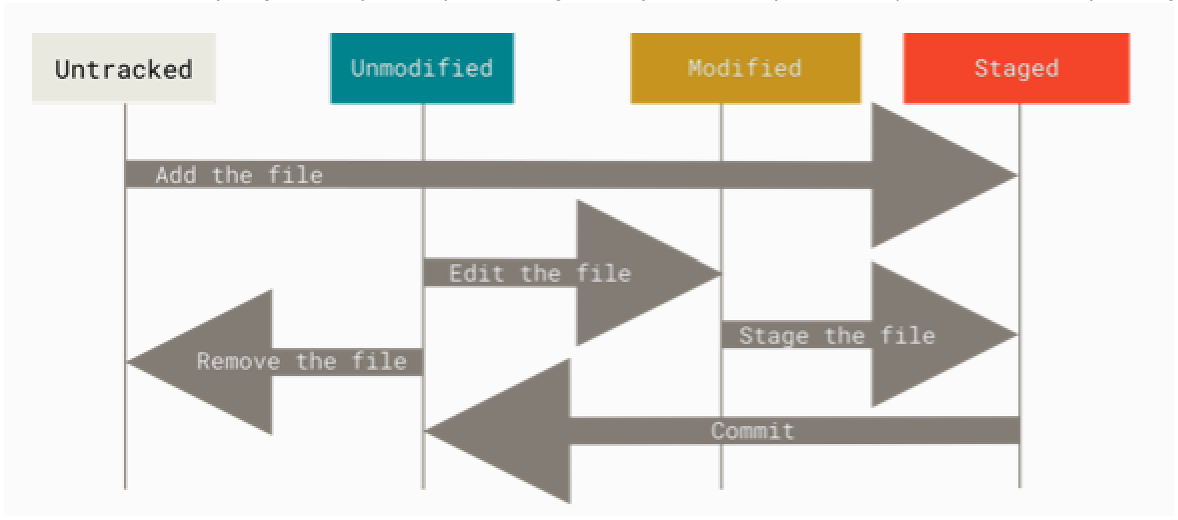


- The three states of Git
  - Git has three main states that your files can reside in: **modified, staged** and **committed:**
    - Modified: you have changed the file but have not committed it to your database yet.
    - Staged: you have marked a modified file in its current version to go into your next commit snapshot.
    - Committed: the data is safely stored in your local database
  - From three states, we have three main sections of a Git project: **the Working Tree, the Staging Area, the Git Directory.**

        - The working tree: a single checkout of one version of the project. These files are pulled out of the compressed database in Git directory and placed on disk for you to use or modify
        - The staging area (index) 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.



- File status:
    - Remember that each file in your working directory can be in one of two states: **tracked** or **untracked**.
        - Tracked files are files that were in the last snapshot, as as any newly staged files; they can be unmodified, modified or staged. *"Tracked files are files that Git knows about"*
        - Untracked file are everything else - any files in your working directory that were in your last snapshot and are not in your staging area.



- The 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.

Setup

- Install Git on MacOS, Linux, Window:
    - Reference: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
    - Cheat sheet: Git - Cheat sheet
- Configuration
    - Identity:

```
• git config --global user.name "Huu Phan"
  git config --global user.email "huu.phan@gotecq.com"
```

- Editor: example using vim editor

```
• git config --global core.editor vim
```

- Checking configuration

```
• git config --list
```

- Github/Gitlab account setup:
  - SSH Access
  - Two Factor Authentication
- Tool or extension: Git Graph, Git Lens,…

Git basic command

Reference: Git Basic Command

Git Commit convention

- Purpose: commit message can adequately communicate why a change was made, and understanding that makes development and collaboration more efficient.
- Convention:
  - Struture:

```
<type>[optional scope]: <description>

[optional body]

[optional footer(s)]
```

  - Convention:
    - **fix:** a commit of the *type* `fix` patches a bug in your codebase (this correlates with `PATCH` in Semantic Versioning).
    - **feat:** a commit of the *type* `feat` introduces a new feature to the codebase (this correlates with `MINOR` in Semantic Versioning).
    - **BREAKING CHANGE:** a commit that has a footer `BREAKING CHANGE:`, or appends a `!` after the type/scope, introduces a breaking API change (correlating with `MAJOR` in Semantic Versioning). A BREAKING CHANGE can be part of commits of any *type*.
    - *types* other than `fix:` and `feat:` are allowed, for example @commitlint/config-conventional (based on the Angular convention) recommends `build:`, `chore:`, `ci:`, `docs:`, `style:`, `refactor:`, `perf:`, `test:`, and others.
    - *footers* other than `BREAKING CHANGE: <description>` may be provided and follow a convention similar to git trailer format.
- Reference: https://www.conventionalcommits.org/en/v1.0.0/

Git Flow

Document: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Exercise

- Git research
- Git practice

Reference:

- Book: https://git-scm.com/book/en/v2

- How Git truly works