Introduction to Git

Linghui Wu

BFI

References:

- https://git-scm.com/
- Rahaman MPCS-51042

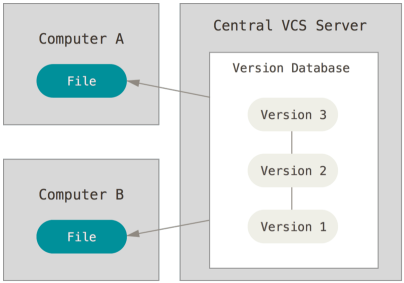
Contents

What is version control system (VCS)?

- A version control system (VCS) records changes to a set of files
  - All the recorded changes are called history
  - The VCS can retrieve specific, previous versions of files
- Why is VCS?
  - If a bug is detected after deployment, you can easily revert to a working version
  - Can help discover which specific changes caused the bug
  - Can discover who was responsible for that change

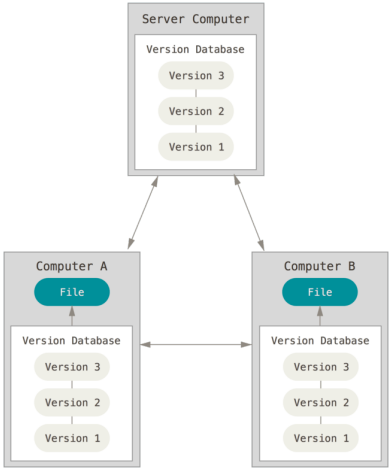
Centralized VCS

- Pros
  - Individuals can see everyone's activity.
  - Admin has fine-grained control over database permissions.
- Cons
  - Developers has restricted access to the repository and cannot update if their connection is ofline.
  - Hopefully database is backed-up elsewhere if central system is corrupted.



Distributed VCS - Git

- Everyone has entire database
  - Arbitrarily-many remote servers
  - Arbitrarily-many local clients
- Pros
  - Any server or client can recover the database.
  - Users can work offline.
  - Support non-linear workflows
- Cons
  - Learning curve



Getting and setting-up Git

- Many choices of clients
  - Highly-recommended command line tools (https://git-scm.com/downloads)
  - GUI clients (https://git-scm.com/downloads/guis)
  - IDE-integrated clients
- First-time set-up
  - Check version

    ```
    $ git --version
    ```

  - Set config variables: your CNetID and `@uchicago.edu` email.

    ```
    $ git config --global user.name "Linghui Wu"
    $ git config --global user.email "linghuiwu@uchicago.edu"
    $ git config --list
    ```

  - Need help?

    ```
    $ git help <verb>
    $ git <verb> --help
    ```

Creating or downloading project

- Initialize a repository from existing code

```
$ mkdir git-demo
$ cd git-demo
$ git init
```

- Downloading an existing repository using SSH

```
$ git clone git@github.com:linghui-wu/RP-orientation-2022.git
```

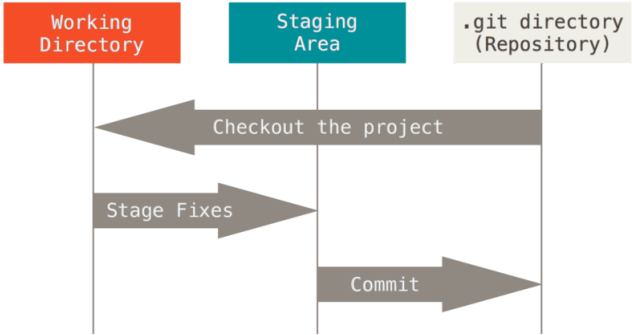
Before first commit

- Check file status

  ```
  $ git status .
  ```

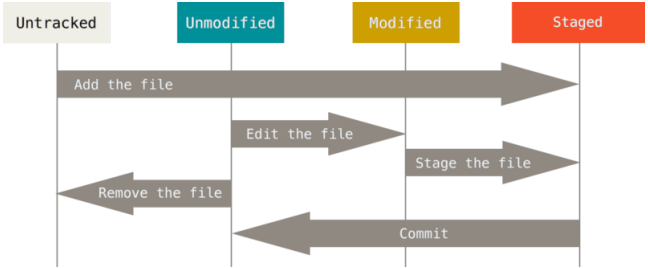
- Create `.gitignore` file to intentionally ignore untracked/modifeid files

Three file states



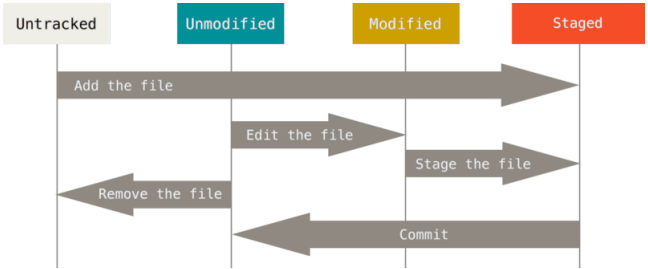
- Working directory stores untracked and modified files
- Staging area organizes changes we want to commit to the repository

Adding local changes



- For new files
  - Untracked → tracked: `git add <file_name>`
- For existing files
  - Unmodified → modified: any text editor
  - Modified → staged: `git add <file_name>`
  - Staged → unmodified: `git commit -m "<commit message>"`

Undoing local changes



- Staged → modified: `git reset -- <file_name>`
- Modified unmodified:
  - Lose changes forever: `git checkout -- <file_name>`
  - Keep changes for later: `git stash`
- Unmodified untracked: `git rm --cached <file_name>`

Showing and adding remote databases

- Every remote that is known to the local repository has a short name and a URL
- To show existing remote repositories

```
$ git remote -v
```

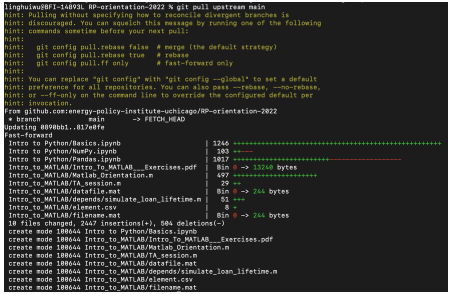
- To add a new remote

```
$ git remote add <remote_name> <url>
```

Getting commits to and from remotes

- Get the latest changes from a remote

  `$ git pull <remote_name> <branch_name>`



- Send committed local changes to a remote

  `$ git push <remote_name> <branch_name>`

- ALWAYS pull before you push

Branching

- Create a branch for desired feature

  ```
  $ git checkout -b <branch_name>
  ```

  It is a shorthand for

  ```
  $ git branch <branch_name>
  $ git checkout <branch_name>
  ```

- After committing the changes, push the branch to remote repo

  ```
  $ git push <remote_name> <branch_name>:<branch_name>
  ```

- Create a pull request and merge your branch into `main` branch
- Delete the branch

  ```
  $ git branch -d <branch_name>
  ```