JOINT INSTITUTE
交大密西根学院

Git Workshop

TechJI

University of Michigan - Shanghai Jiaotong University

Joint Institute

2025.05

Table of Contents

Table of Contents

What is a Shell?

A **shell** is a text-based interface for interacting with the operating system.

It lets you run programs, navigate files, and automate tasks by typing commands.

Common shells:

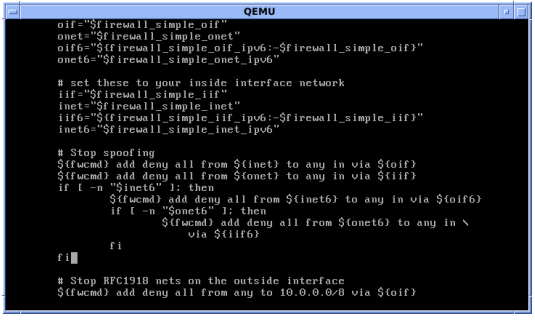
- Zsh — default on macOS
- Bash — default on most Linux systems
- PowerShell — default on Windows

What You Can Do With the Shell

Basic commands (see cheat sheet):

- Navigation: `cd`, `ls`, `pwd`
- Files: `touch`, `mkdir`, `mv`, `rm`, `cp`
- Viewing: `cat`, `less`, `head`, `tail`

Shell has its own scripting language:



Exercise

Tasks:

1. Go to your Desktop directory
2. Create a folder called `test`
3. Inside that folder, create a new file named `ex1.c`

Hint: Use: `ls`, `cd`, `touch`

What is Git?

Git is a distributed version control system. It keeps a full history of your project and allows you to:

- Track changes over time
- Go back to any previous state
- Work in parallel via branches
- Merge contributions from others

Unlike services like Feishu Docs, Git gives you full control and works offline.

Why Use Git?

- Prevent messy filenames like `report-final-fixed-v3-real.c`
- Add different features simultaneously without breaking the main project
- Collaborate with others without conflicts
- Manage not only code, but also papers, configs, and notes

Git isn't just for teams — it's a useful tool even if you work alone.

How Do We Use Git?

Git is installed locally — see the `installation_git` guide in the repo.

Once installed, you can use Git in different ways:

- In the terminal (CLI) — direct and powerful
- With tools like `lazygit` (TUI) — text-based UI in the terminal
- In graphical interfaces (GUI) — like GitHub Desktop or inside VS Code

Git also works with remote platforms like GitHub and Gitea — both are based on Git. They let you back up projects online, share with others, and collaborate more easily.

Table of Contents

Git Setup: init and clone

Initialize a repository

- Creates a Git project in the current folder.
- Adds a hidden `.git` directory to track changes.

```
git init
```

Clone an existing repository

- Copies all files and full version history from a remote project.
- You can now contribute locally.

```
git clone <repository-url>
```

Saving Changes: add and commit

Stage changes

- Choose which changes you want to include in your next commit.

```
git add <file>
```

Create a snapshot

- Save staged changes with a message.
- This becomes part of the project's version history.

```
git commit -m "your message"
```

Tip: Commit frequently, with clear messages.

Remote Sync: push and pull

Upload local changes

- Send your commits to a remote repository (e.g., GitHub, Gitea).

```
git push
```

Download new changes

- Fetch and merge new commits from the remote repository into your current branch.

```
git pull
```

Tip: Always pull before pushing to avoid conflicts.

Switching Branch: checkout

Switch to another branch

- Move to a different line of development.

```
git checkout <branch-name>
```

Restore a file

- Undo changes to a file and bring it back to the last committed version.

```
git checkout -- <file>
```

Note: Git 2.23+ supports `git switch` and `git restore` as clearer alternatives.

Merge Conflicts

What is a merge conflict?

- Happens when Git can't auto-combine changes from two sources.
- Often caused by editing the same lines in different commits.

When does it happen?

- `git merge` or `git pull` introduces conflicting edits
- Someone else pushed to the same branch and you didn't pull first

Git pauses and asks you to fix the conflict before continuing.

Resolving Conflicts: Manual Editing

Step 1: Open the conflicted file Git marks the conflict like this:

```
<<<<<<< HEAD
your version
=======
their version
>>>>>>> branch-name
```

Step 2: Manually edit

- Choose which code to keep (or combine)
- Delete conflict markers

Step 3: Save and confirm `git add <file>`

```
git commit
```

Resolving Conflicts: Git Commands (Optional)

These tools can help, but use with care:

`git diff` — See differences between conflicting versions

`git merge --abort` — Cancel merge and return to previous state

`git checkout -- <file>` — Discard changes and restore last committed version

`git reset --mixed` — Unstage and reset to last commit

Start with manual fixes. Use commands if you're more confident.

Never Use `--force`

`git push --force` can permanently delete others' work.

Why it's dangerous:

- Overwrites history on shared branches
- Causes teammate data loss
- Breaks collaboration workflows

AI tools (e.g. GitHub Copilot, ChatGPT) may suggest it blindly.
Unless you're working solo and fully understand it, don't use `--force`.

Developer: git push origin master --force
Developer: Sorry, wrong window
Every other developer in the chat channel:

Table of Contents