Lecture XX: Introduction to Git

Course: EECS 291
Format: Live demo + guided practice

Today: Build a safe Git workflow you will use for labs and projects.

Outline (40 minutes)

- Why version control and what Git is
- The Git mental model (three areas)
- Install + configure (quick)
- Start a repo and track files
- Stage, commit, and inspect history
- Safe undo and help
- Remotes at a glance
- Wrap-up and next steps

Goals

By the end, you can:

Goals

By the end, you can:

- Explain why version control exists

Goals

By the end, you can:

- Explain why version control exists
- Create a local Git repo and make clean commits

Goals

By the end, you can:

- Explain why version control exists
- Create a local Git repo and make clean commits
- Read history and fix small mistakes safely

Goals

By the end, you can:

- Explain why version control exists
- Create a local Git repo and make clean commits
- Read history and fix small mistakes safely
- Know where to look for help

Why version control?

Why version control?

- Avoid the 'final_final_really_final' file mess

Why version control?

- Avoid the 'final_final_really_final' file mess
- Recover from mistakes (time travel)

Why version control?

- Avoid the 'final_final_really_final' file mess
- Recover from mistakes (time travel)
- Collaborate without overwriting each other

Why version control?

- Avoid the 'final_final_really_final' file mess
- Recover from mistakes (time travel)
- Collaborate without overwriting each other
- Trace who changed what and why

Git vs GitHub

Git vs GitHub

- **Git** = tool on your machine (tracks changes)

Git vs GitHub

- **Git** = tool on your machine (tracks changes)
- **GitHub/GitLab/Bitbucket** = hosted remotes

Git vs GitHub

- **Git** = tool on your machine (tracks changes)
- **GitHub/GitLab/Bitbucket** = hosted remotes
- You can use Git without any hosting service

What is Git?

What is Git?

- A **distributed** version control system

What is Git?

- A **distributed** version control system
- Every copy has the full history

What is Git?

- A **distributed** version control system
- Every copy has the full history
- Fast, works offline, industry standard

The Git mental model

The Git mental model

- **Working directory:** your files right now

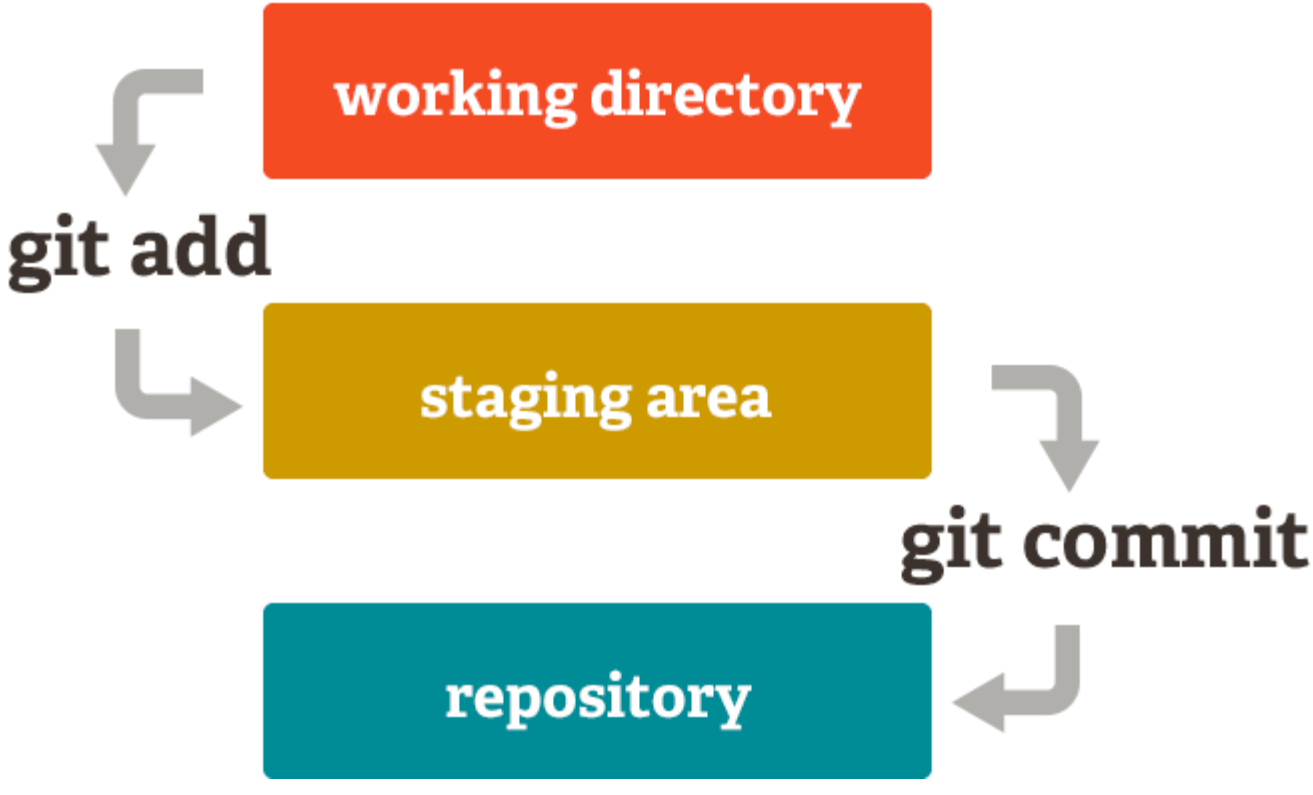
The Git mental model

- **Working directory:** your files right now
- **Staging area (index):** what you *intend* to commit

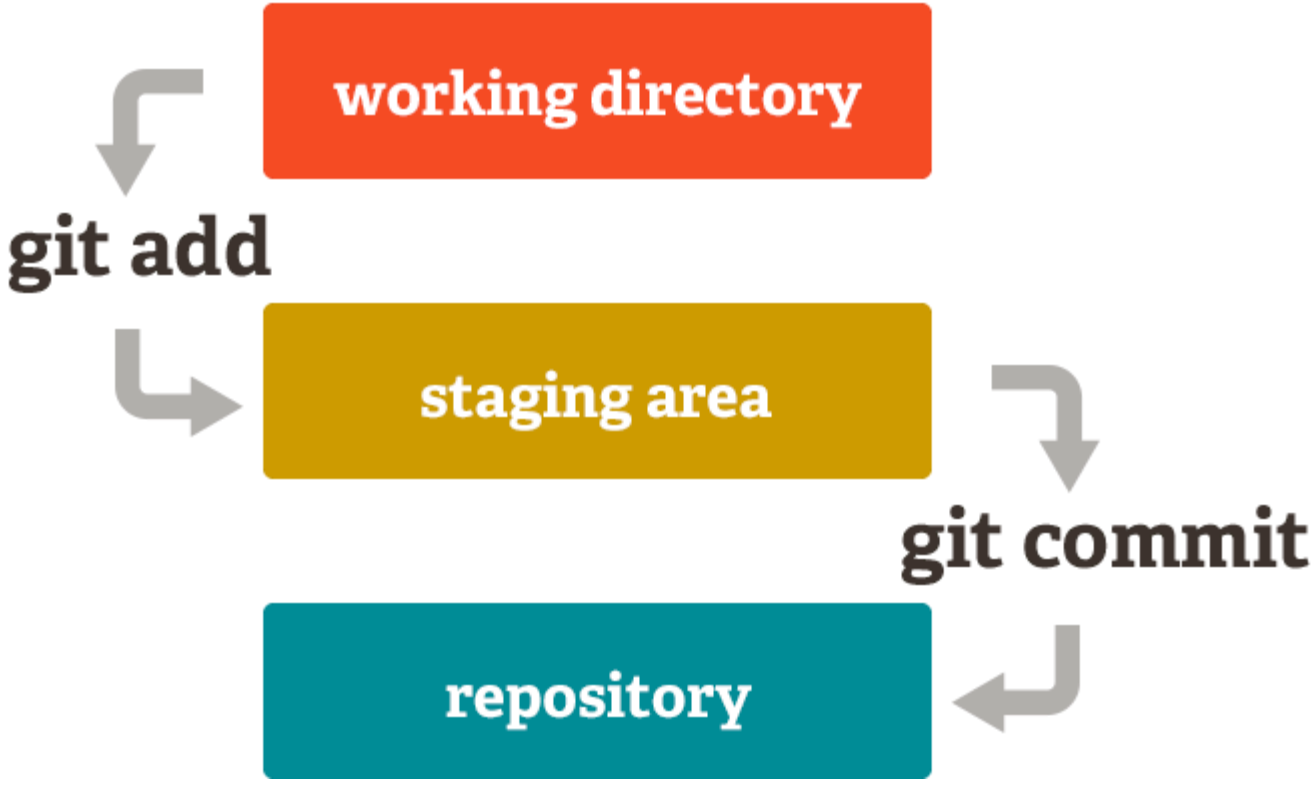
The Git mental model

- **Working directory:** your files right now
- **Staging area (index):** what you *intend* to commit
- **Repository (HEAD):** committed history

Where your changes live



Where your changes live




```
Working Directory  --git add-->  Staging Area  --git commit-->  Repository
        ^                                |
        |                                v
   edit files                    git restore --staged
```

Quick command map

Quick command map

Goal	Command
Check status	`git status`
Stage changes	`git add <file>`
Commit	`git commit -m "message"`
View history	`git log --oneline`
Undo staging	`git restore --staged <file>`
Discard local edit	`git restore <file>`

Install + verify

Install + verify

- Windows: Git Bash (includes Git + Unix-like shell)
- macOS: Terminal + `brew install git` (if needed)
- Linux: `apt install git` / `dnf install git` (if needed)
- Install Git and verify:

```
git --version
```

Configure (quick)

Configure (quick)

- Set identity once:

```
git config --global user.name "Ada Lovelace"
git config --global user.email "ada@example.com"
```

Configure (quick)

- Set identity once:

```
git config --global user.name "Ada Lovelace"
git config --global user.email "ada@example.com"
```

- Optional quality-of-life:

```
git config --global core.editor "code --wait"
git config --global init.defaultBranch main
```

Demo: start a repo

Demo: start a repo

```
mkdir demo-git
cd demo-git
git init
git status
```

- Git created a hidden `.git/` folder
- That folder is the history database
- The rest of your files are normal files

Add a README.md

Add a README.md

Create a file and check status:

```
echo "# Demo Git" > README.md
git status
```

Add a README.md

Create a file and check status:

```
echo "# Demo Git" > README.md
git status
```

Short status view:

```
git status -sb
```

Add a README.md

Create a file and check status:

```
echo "# Demo Git" > README.md
git status
```

Short status view:

```
git status -sb
```

- **Untracked** means Git sees it but is not tracking it yet

Add Notes

Add Notes

```
echo "hello" > notes.txt
git status
```

Add Notes

```
echo "hello" > notes.txt
git status
```

Sample output:

```
On branch main
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        notes.txt
```

Staging: choose what goes into a commit

Staging: choose what goes into a commit

```
git add notes.txt
git status
```

Staging: choose what goes into a commit

```
git add notes.txt
git status
```

Sample output:

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   notes.txt
```

Commit: capture a clean snapshot

Commit: capture a clean snapshot

```
git commit -m "Add notes file"
```

Commit: capture a clean snapshot

```
git commit -m "Add notes file"
```

Commit message tips:

- Use short, descriptive verbs
    - Read as "Accepting this commit will <X>"
- One change per commit when possible

Edit, stage, commit again

Edit, stage, commit again

```
echo "\nmore notes" >> notes.txt
git status
```

Edit, stage, commit again

```
echo "\nmore notes" >> notes.txt
git status
```

```
git add notes.txt
git commit -m "Add more notes"
```

History: what changed?

History: what changed?

```
git log --oneline
```

History: what changed?

```
git log --oneline
```

Example output:

```
a1b2c3d Add more notes
d4e5f6g Add notes file
```

History: what changed?

```
git log --oneline
```

Example output:

```
a1b2c3d Add more notes
d4e5f6g Add notes file
```

```
git log --oneline --graph --decorate
```

Safe undo (local only)

Safe undo (local only)

Unstage a file (keep edits):

```
git restore --staged notes.txt
```

Safe undo (local only)

Unstage a file (keep edits):

```
git restore --staged notes.txt
```

Discard an edit (careful):

```
git restore notes.txt
```

Safe undo (local only)

Unstage a file (keep edits):

```
git restore --staged notes.txt
```

Discard an edit (careful):

```
git restore notes.txt
```

Avoid destructive history rewrites in this course:

- `git reset --hard`
- `git push --force`

Add a .gitignore (recommended)

Add a .gitignore (recommended)

Keep junk out of commits:

```
__pycache__/
.ipynb_checkpoints/
.DS_Store
```

Add a .gitignore (recommended)

Keep junk out of commits:

```
__pycache__/
.ipynb_checkpoints/
.DS_Store
```

Create it once:

```
echo "__pycache__/\n.ipynb_checkpoints/\n.DS_Store\n" > .gitignore
git status
```

Add a .gitignore (recommended)

Keep junk out of commits:

```
__pycache__/
.ipynb_checkpoints/
.DS_Store
```

Create it once:

```
echo "__pycache__/\n.ipynb_checkpoints/\n.DS_Store\n" > .gitignore
git status
```

Tip: add data files to `.gitignore` unless the assignment says otherwise.

Add + commit ignore file

Add + commit ignore file

```
git add .gitignore
git commit -m 'Add gitignore to repo'
```

Remotes at a glance

Remotes at a glance

Clone once to start from a remote:

```
git clone https://github.com/org/repo.git
```

Remotes at a glance

Clone once to start from a remote:

```
git clone https://github.com/org/repo.git
```

Pull updates, push your work:

```
git pull  #(git fetch + git merge)
git push
```

Remotes at a glance

Clone once to start from a remote:

```
git clone https://github.com/org/repo.git
```

Pull updates, push your work:

```
git pull  #(git fetch + git merge)
git push
```

Remember: most commands are local until you push

Getting help fast

Getting help fast

```
git help <command>
git <command> --help
git help -g
```

Practice (5 minutes)

Practice (5 minutes)

Try this loop:

1. Create a repo
2. Add a file
3. Commit
4. Edit, stage, commit
5. View history

Wrap-up

Wrap-up

Workflow loop = edit -> stage -> commit -> push

Wrap-up

Workflow loop = edit -> stage -> commit -> push

Next topics:

- Remotes
- Branching and Merging (and conflict resolution)
- Time Travel (visiting and/or changing the past)

Wrap-up

Workflow loop = edit -> stage -> commit -> push

Next topics:

- Remotes
- Branching and Merging (and conflict resolution)
- Time Travel (visiting and/or changing the past)

Resources:

- https://learngitbranching.js.org
- https://www.w3schools.com/git/