Version Control Systems: Git & GitHub

CMP9134: Software Engineering

Dr Francesco Del Duchetto
Lecturer in Robotics and Autonomous Systems

University of Lincoln

6 February 2026

Lab Session's Agenda

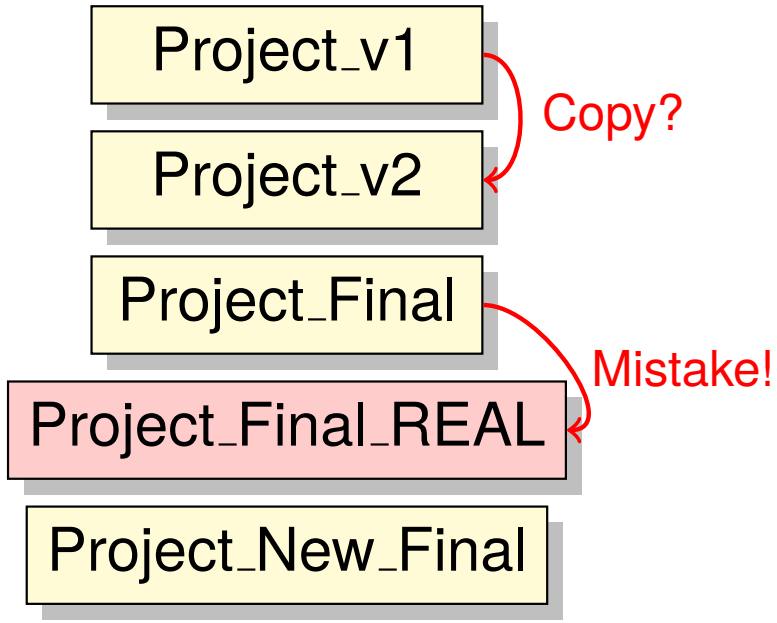
Agenda

The ”Copy-Paste” Method

”I’ll just make a copy of the folder before I try this risky change...”

The Result:

- Hard to find the right file.
- Wastes disk space.
- ”Which ’Final’ is the real one?”



Collaboration

The Scenario: You and a teammate are working on the same codebase.

- You both make changes to the same file.
- You email each other the updated files.
- You accidentally overwrite each other's work.
- Now you have two conflicting versions of the same file.
- Who has the latest changes? How do you merge? How do you keep track of who did what? ...

The Solution: Version Control Systems (VCS)

A VCS is a system that records changes to a file system over time.

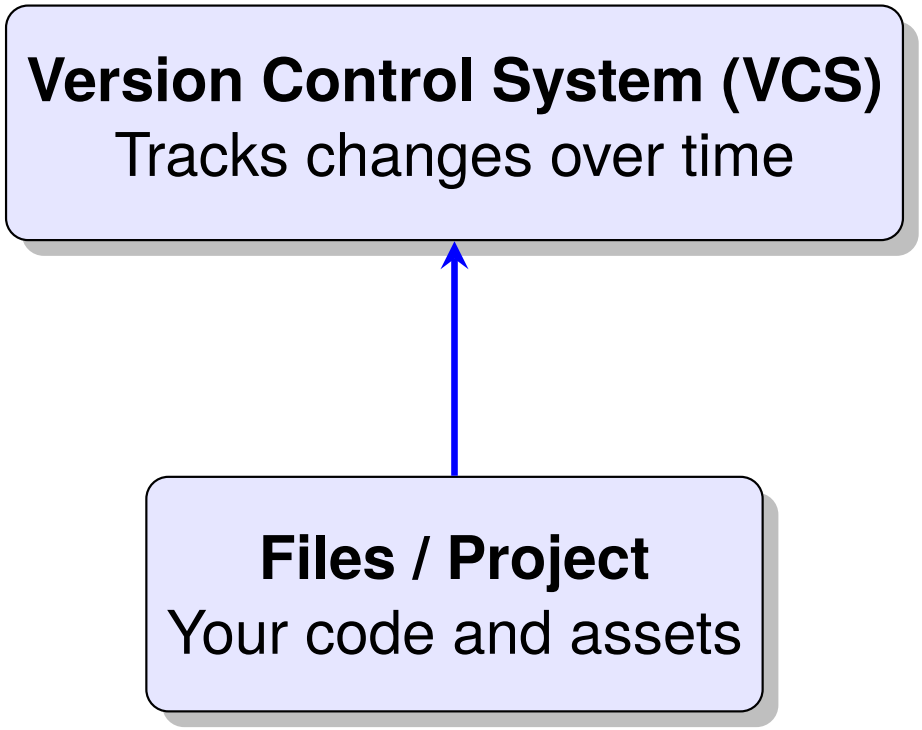
Key Benefits:

1. **History:** You can see exactly *who* changed *what* and *when*.
2. **Revert:** Made a mistake? Go back to yesterday's version instantly.
3. **Collaboration:** Multiple people can work on the same file without overwriting each other.
4. **Backup:** Your code exists on your machine AND the cloud.

Agenda

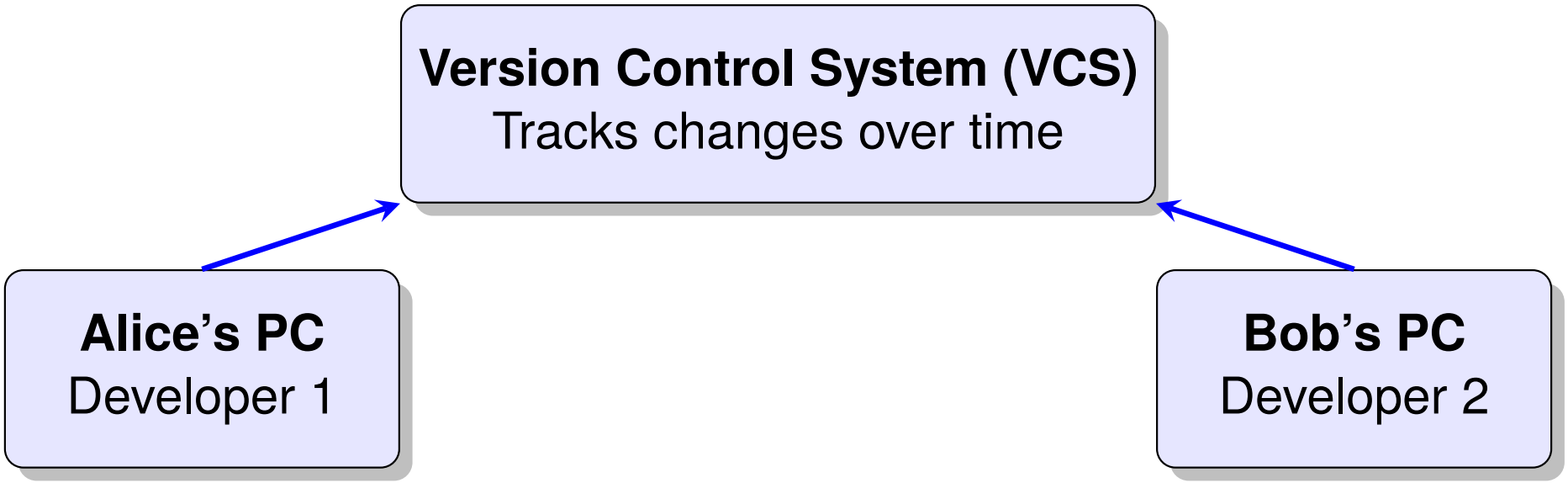
What is a Version Control System (VSC)?

Is a program/set of tools that help you manage changes to files over time.



What is a Version Control System (VSC)?

VSC systems allow several team members to work on the same project simultaneously.



Each developer has a local copy of the project and can make changes independently.

Popular Version Control Systems

- **Git:** The most widely used VCS today. Created by Linus Torvalds (creator of linux) in 2005 for Linux kernel development.
- **Subversion (SVN):** An older centralized VCS, still used in some legacy projects.
- **Mercurial:** Similar to Git, but with a different design philosophy.
- **Perforce:** A commercial VCS often used in large enterprises and game development.

In this lecture, we will focus on **Git** and its cloud counterpart, **GitHub**.

If you are curious about what “git” stands for, have a look at the first definition from the creator: `https://github.com/git/git/blob/e83c5163316f89bfbde7d9ab23ca2e25604af290/README`

Centralized vs Distributed VCS

Centralized VCS:

- A single central server stores all versions of the project.
- Developers check out files from this central place.
- Examples: Subversion (SVN), Perforce.

Distributed VCS:

- Every developer has a full copy of the entire repository (including history).
- Changes can be shared between repositories.
- Examples: Git, Mercurial.

Most modern VCS (including Git) are distributed, offering greater flexibility and offline capabilities.

Git vs GitHub: The Distinction

Git (The Engine)

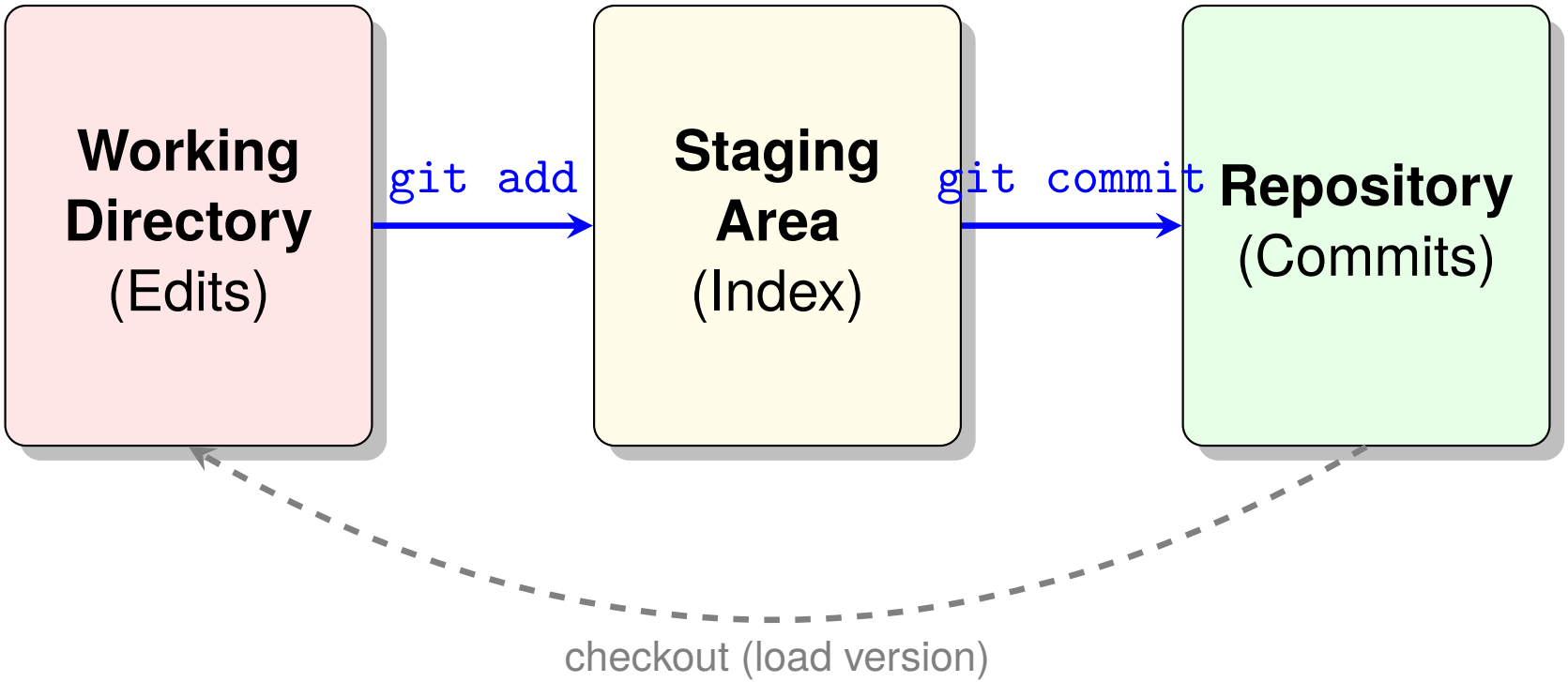
- A command-line tool installed on your laptop.
- Tracks history locally.
- Does not need internet.
- *Analogy: Microsoft Word*

GitHub (The Host)

- A website (cloud service).
- Hosts Git repositories online.
- Adds social features (Teams, Pull Requests).
- *Analogy: Google Drive / Dropbox*

The Three States of Git

Files move through three ”zones”.



- **1. Working:** Where you create and edit files.
- **2. Staging:** Where you select files to be saved.
- **3. Repository:** Where the permanent snapshot is stored.

Agenda

The Terminal: Your New Best Friend

Git is primarily a **Command Line Interface (CLI)** tool.

1. Where am I?

```
pwd   # Print Working Directory
ls    # List Files
```

2. Moving Around

```
cd FolderName   # Enter folder
cd ..           # Go back/up
```

3. Creating

```
mkdir NewProject # Make folder
touch file.txt   # Create file
```

```
user@laptop:~$ pwd
/home/user
user@laptop:~$ ls
Documents Downloads
user@laptop:~$ cd Documents
user@laptop: /Documents$ _
```

Step 1: Check Status

Rule #1 of Git: If you are confused, type `git status`.

```
git status
```

It tells you:

- Red Files: Modified but not staged (Working Directory).
- Green Files: Staged and ready to commit (Staging Area).
- ”Nothing to commit”: Everything is saved.

Step 2: Staging Files (Add)

We move files from ”Working” to ”Staging”.

```
# Add a single file
git add Program.cs

# Add EVERYTHING (The most common command)
git add .
```

Why separate steps? Sometimes you modify 10 files, but you only want to save 5 of them as a specific update (e.g., ”Fix login bug”). You stage only the login files.

Step 3: Committing (Save)

We seal the Staged files into a permanent snapshot.

```
git commit -m "Added the main menu logic"
```

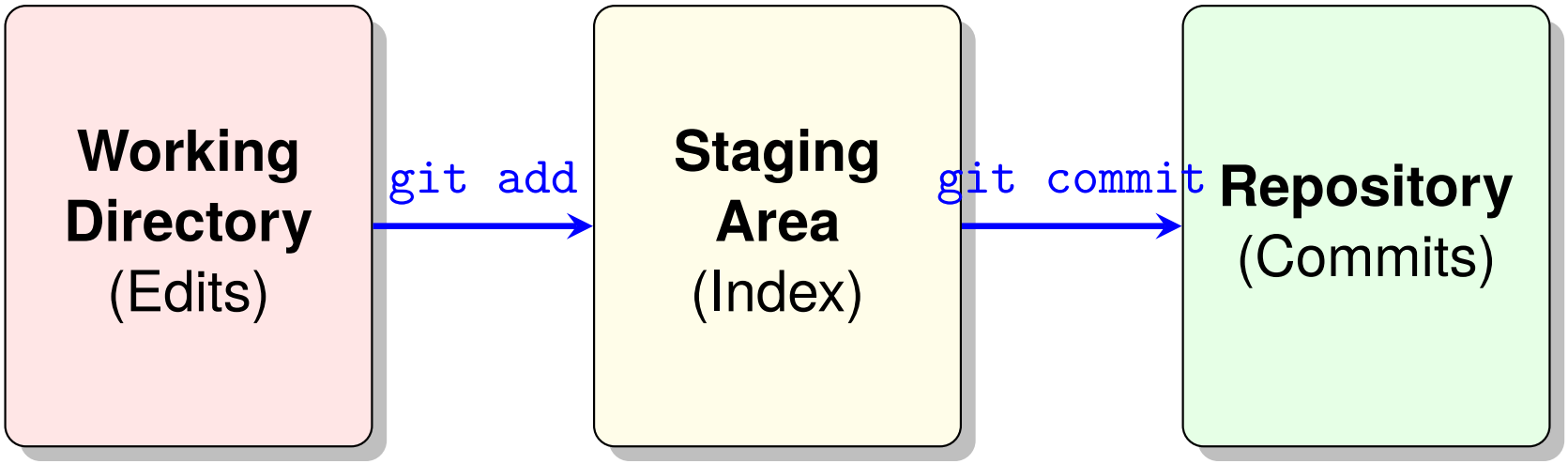
The Commit Message (`-m`):

- Mandatory.
- Be descriptive! ”Fixed bug” is bad. ”Fixed divide-by-zero error” is good.

You can check the history of commits in your repository with:

```
git log
```

The Core Cycle Recap



- **1. Status:** Check what's happening (`git status`).
- **2. Add:** Stage files (`git add .`).
- **3. Commit:** Save snapshot (`git commit -m "..."`).

Agenda

Understanding Commits

Each commit has a unique ID (hash).

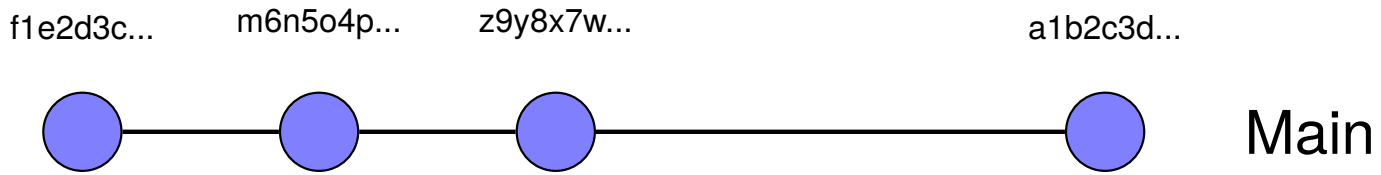
View Commit History:

```
git log
```

Sample Output:

```
commit a1b2c3d4e5f6g7h8i90jklmnopqrstuv
Author: Your Name <email>
Date:   Mon Jan 12 10:00:00 2026 +0000

    Added the main menu logic
```



Branches

Branches: Pointers to specific commits, allowing parallel development.

- `main`: The default branch.
- `feature-x`: A branch for a new feature.
- `...`

You will see something like:

```
commit a1b2c3d4e5f6g7h8i90jklmnopqrstuv (HEAD -> main)
Author: ..
```

Which means you are currently on the `main` branch. `HEAD` points to the latest commit on that branch.

Reverting to Previous Versions

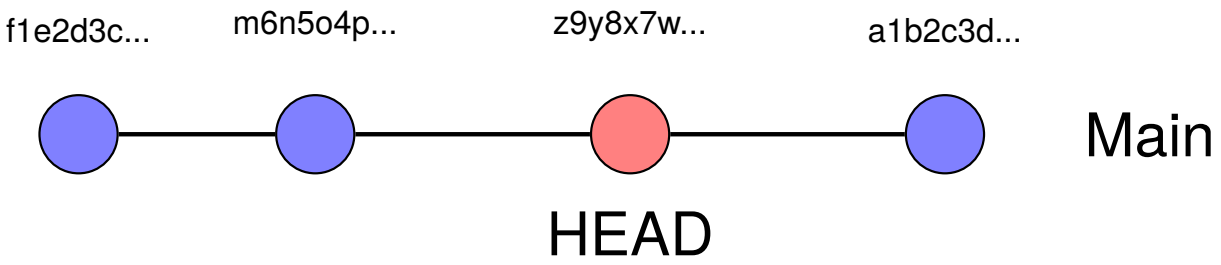
If you made a mistake, you can revert to a previous commit.

Find the Commit ID:

```
git log
```

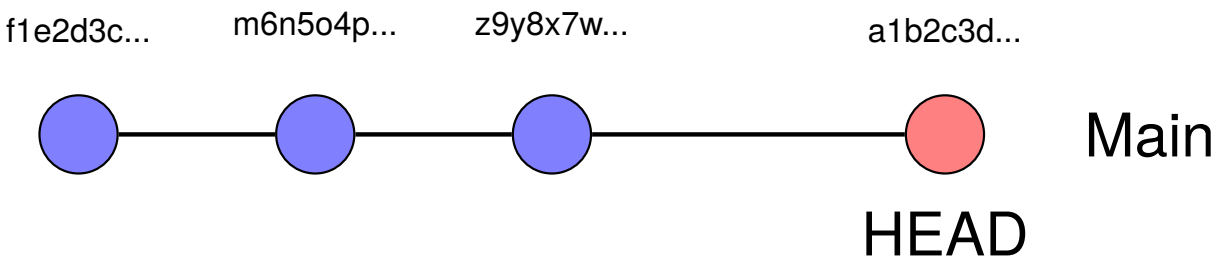
Revert to that Commit:

```
git checkout z9y8x7w...
```

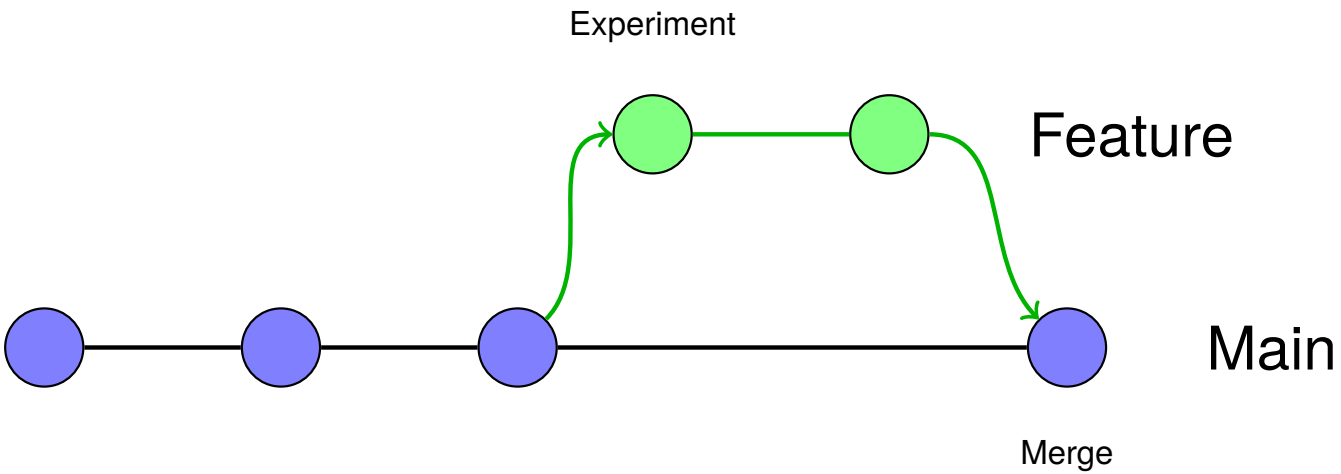


To return to the latest commit on the main branch:

```
git checkout main
```



Branching: Parallel Universes



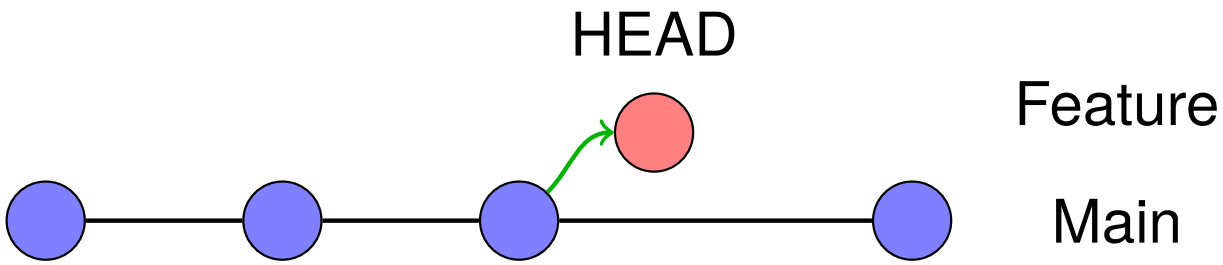
The Scenario: You want to add a feature.

- You work in the **Feature Branch** (Green).
- The **Main Branch** (Blue) stays safe.
- If it works, you **Merge** it back.

Branching Commands

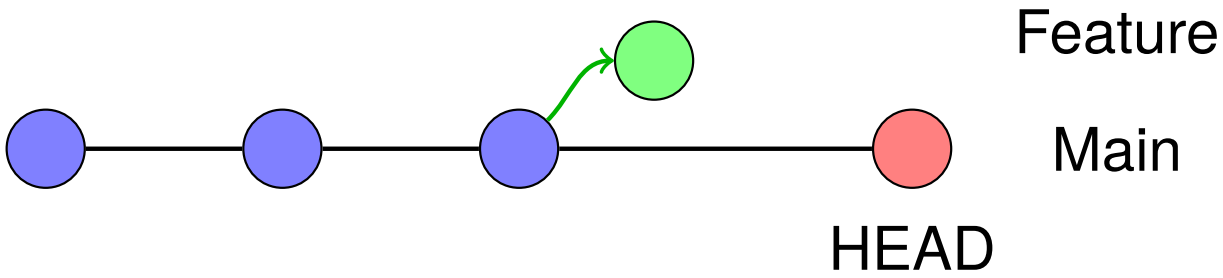
Create and switch to a new branch:

```
# Create and Switch to a new branch
git checkout -b feature
# ... write code, add, commit ...
```



Switch back to an existing branch:

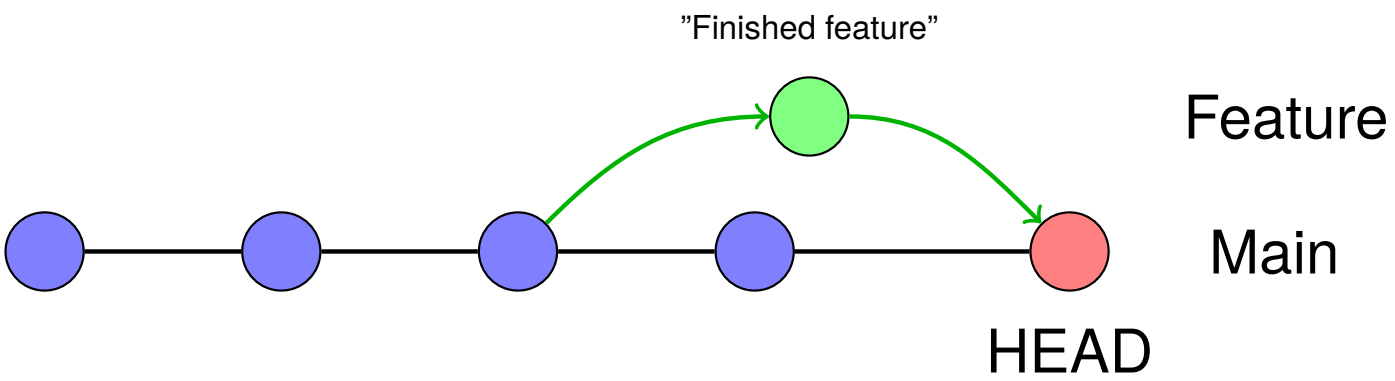
```
git checkout main
```



Merging Branches

Once the feature is complete and tested, merge it back into main.

```
# while on the feature branch, commit your changes
git checkout feature
# ... make changes ...
git add .
git commit -m "Finished feature"
# Switch to main branch
git checkout main
# Merge feature branch into main
git merge feature
```



Reverting with Reset (Advanced)

Warning

This can rewrite history. Use with caution.

Soft Reset: Moves HEAD but keeps changes in Working Directory.

```
git reset --soft <commit-id>
```

Hard Reset: Moves HEAD and discards changes in Working Directory.

```
git reset --hard <commit-id>
```

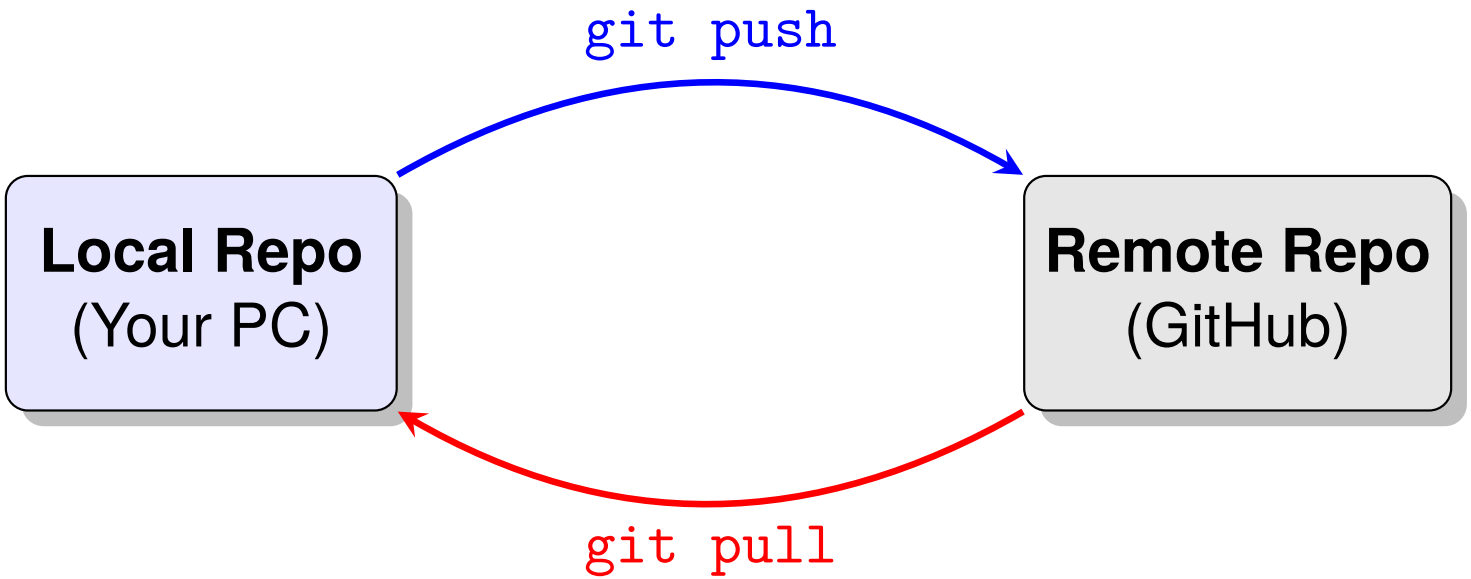
Questions on Git so far?

Let's Try the interactive "Visualizing Git" demo in your browser to explore commits, branches and checkouts:
`https://git-school.github.io/visualizing-git/`

Any questions before we move on to GitHub?

Agenda

Moving to the Cloud



- **Origin:** The default nickname Git gives to your remote server.
- **Push:** Upload changes (Local $\rightarrow$ Remote).
- **Pull:** Download changes (Remote $\rightarrow$ Local).

Create Repo and Connecting to GitHub

If you create a new existing local repo:

```
# 0. Initialize Git (if not done yet)
git init
# 1. Stage and commit your files
git add .
git commit -m "Initial commit"
# 2. Link your local folder to GitHub (need to create
# an empty repo on GitHub first)
git remote add origin https://github.com/User/Repo.git
# 3. Rename branch to 'main' (modern standard)
git branch -M main
# 4. Push your code!
git push -u origin main
```

If you are starting from an existing repo on GitHub:

Clone to download a repo from scratch (e.g., on a new PC):

```
git clone <url>
```

Pulling Changes

If others have pushed changes, you need to pull them first.

Pull the latest changes:

```
git pull origin main
```

Good practice

Always pull before you push to avoid conflicts!

Merging

If there are changes both locally and remotely, Git will try to merge them.

Automatic Merge: Git combines changes automatically.

```
git pull origin main
```

Manual Merge (Conflict): If changes overlap, Git will mark conflicts in the files. You must resolve them manually.

```
# After resolving conflicts in files
git add conflicted-files # replace with actual file names
git commit -m "Resolved merge conflicts"
```

Pull Requests

On GitHub, changes are often proposed via **Pull Requests (PRs)**.

- A way to review and discuss changes before merging them into the main branch.
- Typically used in collaborative projects.
- Enables code review, comments, and approval workflows.

GitHub Features

- **Issues:** Track bugs and feature requests.
- **Actions:** Automate workflows (e.g., testing, deployment).
- **Wiki:** Documentation for your project.
- **Projects:** Kanban boards for task management.

Your Turn Now!

1. Open the Workshop1_tasks.pdf file from blackboard
2. Start working on the git/github exercises!
3. Ask for help if you get stuck!