COMPSCI 1XC3

Week 5 — Git & GitHub Basics

> **Goal:** understand Git's *model*, not just memorize commands.

Before We Start: Install Git

Check if Git is installed:

```
git --version
```

If not installed:

- **macOS:** `brew install git` or download from git-scm.com
- **Linux:** `sudo apt-get install git` (Ubuntu/Debian) or `sudo yum install git` (Fedora)
- **Windows:** Download from git-scm.com

Configure Git (first time only):

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

What is Version Control?

- Tracks changes over time (history + rollback)
- Enables collaboration without overwriting work
- Supports branching + merging workflows

  Common systems: SVN, CVS, Perforce, Git

Why Git?

Git is different because:

- **Distributed:** every machine has full repository history
- **Local-first:** most operations are fast and offline
- **Snapshots (not diffs):** commits store full file snapshots
- **Hashes:** commits and files are identified by content (SHA-1)

Git's Mental Model (3 Areas)

Think in **three places**:

1. **Working directory** — your actual files
2. **Staging area** — what will go into the next commit
3. **Repository** (`.git`) — commit history

> A commit is a **snapshot of what's staged**, not what's edited.

File States in Git

Files in your working directory are either:

- **Untracked** — Git doesn't know this file exists
- **Tracked**
  - **Modified** — changed since last commit
  - **Staged** — selected for the next commit
  - **Committed** — stored in repository history

Basic Git Workflow

1. Modify files in the working directory
2. Stage selected changes
3. Commit to the **local repository**
4. Push commits to a **remote repository** (e.g. GitHub)

Local Repo vs Remote Repo

- **Local repository**
  - On your machine
  - Full project history
  - Works offline
- **Remote repository**
  - Hosted online
  - Used for sharing and collaboration

> GitHub is **not** Git — it just hosts Git repositories.

Core Commands (Minimum Set)

- `git status` — inspect file states
- `git add <file>` / `git add .` — stage changes
- `git commit -m "message"` — create a snapshot
- `git log` — view commit history

Clone vs Init

- `git clone <url>`
  - Copies an existing repository
  - Includes full commit history
- `git init`
  - Creates a new empty repository in a folder

Branching (Concept)

- A **branch** is a separate line of development
- Default branch is usually `main`
- Branches let you work safely without breaking `main`

Branching (Commands)

- `git branch` — list branches
- `git checkout -b feature` — create and switch
- `git merge feature` — merge into current branch

Merging (What it Means)

To merge `feature` into `main`:

1. `git checkout main`
2. `git merge feature`

> Git combines commit histories
>
> Conflicts happen if the same lines changed

TA Check-Off — Task 1

GitHub Setup

1. Create a GitHub account at github.com (if you don't already have one)
2. Show your profile page (username visible)

TA Check-Off — Task 2

Clone + Inspect a Repo

Clone this repository:

```
git clone https://github.com/CaamlMcMaster/Example.git
```

Then demonstrate:

1. `git status` — what does it show?
2. `git log` — how many commits exist?
3. `git branch` — what branch are you on?

TA Check-Off — Task 3

Branch, Modify, and Merge

Starting from the cloned repo:

1. Create a new branch called `update-readme`
2. Modify `README.md` — add a line with your name and student number
3. Stage and commit your change
4. Switch back to `main` and merge your branch
5. Verify with `git log` — your commit should now be in `main`