Version Control: Git and GitHub

Lecture 2 Notes

School of Computing Communication and Media Studies

Masoud Hamad

CS2113 – Software Development Project
Academic Year 2025

Contents

1 Introduction to Version Control

Version Control is a system for storing code that enables:

- Storing backups of current and older program versions
- Enabling code/project sharing for collaboration
- Marking specific project states for recovery
- Tracking developer contributions and changes
- Facilitating bug identification

Key Insight

Version control enables "using and developing others' code, even without ever meeting in person." It is one of the most vital skills required in professional software development.

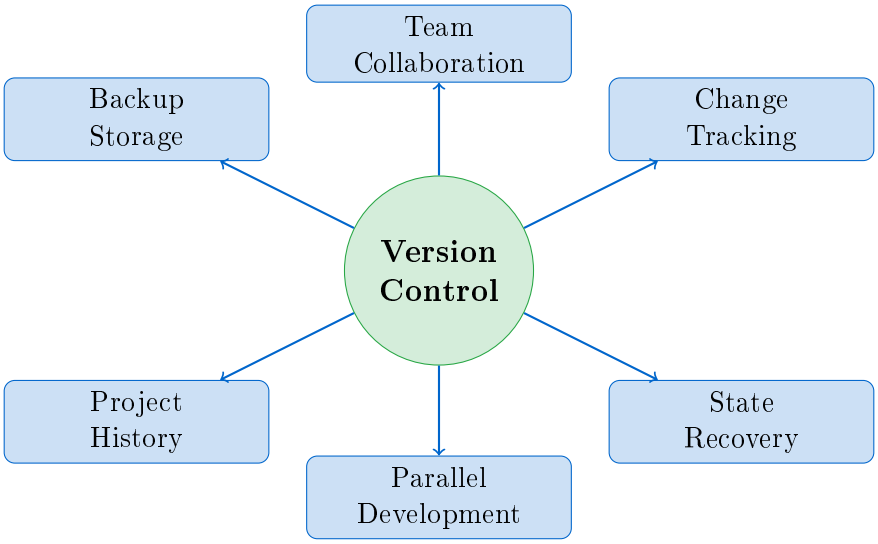


Figure 1: Benefits of Version Control Systems

2 About Git

2.1 History and Origin

Git was created by **Linus Torvalds**, the developer of the Linux kernel. Torvalds created it to manage Linux kernel code versions and share them with collaborators.

2.2 Related Platforms

- **GitHub:** Service for storing and publishing projects (most popular)

- **GitLab:** Similar alternative platform with CI/CD features
- **Bitbucket:** Atlassian's Git hosting service

Tip

Git and GitHub are used in solo and collaborative projects across the industry. Applications extend beyond code—many people backup their course materials, documentation, and other files using Git.

3 Installation and Setup

3.1 Installing Git

- **Windows:** Download Git for Windows from `gitforwindows.org`
- **macOS:** Install via Homebrew: `brew install git`
- **Linux:** Use package manager: `sudo apt install git`

3.2 Initial Configuration

After installation, configure Git with your identity:

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

Warning

Use the same email address as your GitHub account. If you don't want your email to be public, GitHub offers a specific noreply email address.

4 Git Fundamentals

4.1 Starting a Git Project

A project is simply a folder containing files. Initialize with:

```
git init
```

This creates a `.git` subfolder storing all project metadata and history.

4.2 Understanding Commits

A **commit** is a bundle of changes made to files inside the project. In practice, these changes are often adding or removing text from files.

Commits function as "steps towards a finished project." Every commit adds some changes to the previous commit.

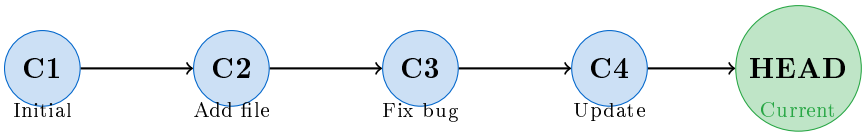


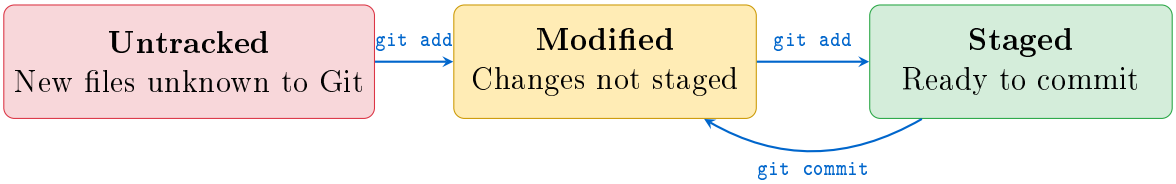
Figure 2: Git Commit History as a Linear Chain

4.3 The Commit Process

1. Make changes to files
2. **Stage** changes using `git add`
3. **Create commit** using `git commit`
4. Check status with `git status`

5 File States in Git

The `git status` command reveals three possible states:



Green = Staged (Changes to be committed)
Red = Not staged or untracked

Figure 3: Git File States and Transitions

5.1 State Descriptions

1. **Changes to be committed (Staging Area):**
   - Changes ready for the next commit
   - Displayed in green
   - Add files via `git add`
   - Remove via `git rm -cached`

2. **Changes not staged for commit:**
   - Modified tracked files not yet staged
   - Undo via `git restore filename`
3. **Untracked files:**
   - New files unknown to Git
   - Must be added with `git add`

Key Insight

Git tracks *changes*. When you create a file, the "change" is the file creation. When you modify it, the "change" is the modification. Each change must be staged before committing.

6 Essential Git Commands

6.1 Adding Changes

```
# Add a single file
git add filename.txt

# Add all files in a folder
git add folder_name/

# Add all changes in the project
git add .

# Interactive selection (tracked files only)
git add -p
```

6.2 Creating Commits

```
# Quick commit with message
git commit -m "Descriptive commit message"

# Open editor for detailed message
git commit
```

Tip

Write clear, descriptive commit messages that explain *why* the change was made, not just *what* changed.

6.3 Viewing History

```
# View commit history
git log

# Compact one-line format
git log --oneline

# View changes in a specific commit
git show commit_id
```

7 Branches

Branches allow separating commits from each other. A new branch can be developed independently from the main branch without affecting it.

7.1 Branch Terminology

- **main** (modern standard): The primary branch
- **master** (traditional): Legacy name for primary branch
- **HEAD**: Points to the current commit/branch

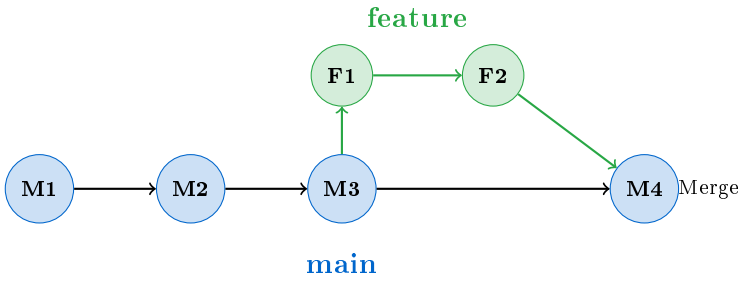


Figure 4: Git Branching and Merging

7.2 Branch Commands

```
# Create a new branch
git branch feature-name

# Switch to a branch
git checkout feature-name

# Create and switch in one command
git checkout -b feature-name

```

```
# List all branches
git branch

# Delete a branch
git branch -d feature-name
```

8 Working with GitHub (Remote Repositories)

8.1 Remote Repository Concept

A **remote repository** is a version of your project hosted on a server (like GitHub). It enables sharing code with others and serves as a backup.

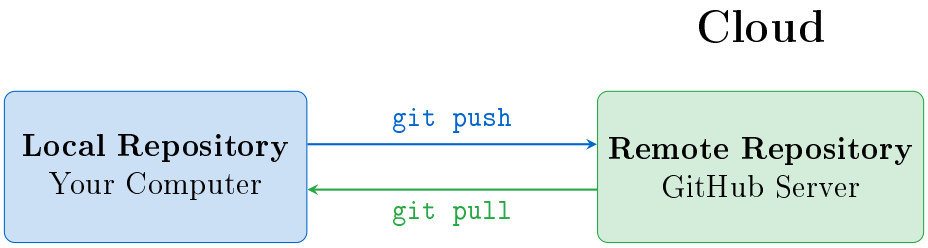


Figure 5: Synchronization Between Local and Remote Repositories

8.2 Adding a Remote

```
# Add a remote repository
git remote add origin https://github.com/username/repo.git

# View configured remotes
git remote -v
```

8.3 Push and Pull

```
# Push changes to remote (first time)
git push -u origin main

# Push subsequent changes
git push

# Pull changes from remote
git pull

# Fetch without merging
```

```
git fetch
```

Warning

Always pull before starting development to ensure you have the latest changes from your team members!

9 Cloning Repositories

```
# Clone a repository
git clone https://github.com/username/repo.git

# Clone into a specific folder
git clone https://github.com/username/repo.git folder-name
```

Key Insight

In order to push commits to a cloned project, the project's owner must add you as a **collaborator**. Otherwise, `git push` will fail due to insufficient permissions.

10 Handling Conflicts

10.1 What Causes Conflicts?

Conflicts occur when two developers make changes to the same lines in a file. Git cannot automatically determine which change to keep.

10.2 Conflict Markers

When a conflict occurs, Git marks the conflicting section:

```
<<<<<<< HEAD
Your local changes
=======
Changes from remote
>>>>>>> abc123def456
```

10.3 Resolving Conflicts

1. Open the conflicted file
2. Find and edit the conflict markers
3. Choose which changes to keep (or combine them)

4. Remove the conflict markers («, ====, »)
5. Stage the resolved file: `git add filename`
6. Commit the merge: `git commit`

Tip

The easiest way to avoid conflicts is by always pulling before starting development and communicating with your team about who is working on which files.

11 Using Stash

Git Stash temporarily hides local changes so you can pull updates from remote, then restore your changes afterward.

```
# Stash changes
git stash

# Stash including untracked files
git stash -u

# Restore stashed changes
git stash pop

# List all stashes
git stash list
```

12 Best Practices

Git Best Practices Checklist

1. Pull before starting any development work
2. Make small, focused commits with clear messages
3. Push frequently to backup your work
4. Use branches for new features
5. Never commit sensitive data (passwords, API keys)
6. Communicate with team about file ownership
7. Review `git status` before committing

Figure 6: Git Workflow Best Practices

Warning

Never push anything secret to the remote repository: no passwords, personal API keys, student numbers, or anything you wouldn't want to share with the whole Internet. Once pushed, it's in the history forever!

13 Git Command Reference

Command	Description
`git init`	Initialize a new repository
`git clone <url>`	Clone a remote repository
`git status`	Check current status
`git add <file>`	Stage changes
`git commit -m "msg"`	Create a commit
`git push`	Push to remote
`git pull`	Pull from remote
`git branch`	List branches
`git checkout <branch>`	Switch branches
`git merge <branch>`	Merge a branch
`git log`	View commit history
`git diff`	View changes
`git stash`	Temporarily hide changes
`git remote -v`	View remotes

Table 1: Essential Git Commands Reference

14 Exercises

Submission Requirements

Submit exercises to Moodle as a single PDF containing the information from Exercises 7, 15, and 16. Deadline: Five days from today.

14.1 Exercise 1: GitHub Account

Create a GitHub account at `github.com` with a professional, CV-appropriate username.

14.2 Exercise 2: Install Git

Install Git on your computer following the installation instructions for your operating system.

14.3 Exercise 3: Configure Git

Configure Git with your name and email using the commands shown in Section 3.2.

14.4 Exercise 4: Create Commits

Create a project with:

- `story.txt` (a lengthy text file)
- `shopping_list.txt` (multiple rows)
- `school/` folder with `school_file.txt`

Create three commits (one per item) with descriptive messages. Verify with `git log`.

14.5 Exercise 5: Staging and Diff

1. Remove staged changes back to modified state
2. Add items to shopping list
3. Inspect changes with `git diff`
4. Stage changes without committing
5. Unstage changes
6. Remove changes entirely so file reverts

14.6 Exercise 6: Credential Storage

Configure Git credential helper for your operating system.

14.7 Exercise 7: GitHub Repository

Create a public GitHub repository for your project and add it as a remote. Submit the repository link to Moodle.

14.8 Exercise 8: Push Commits

Push your commits to the remote repository. Verify all changes are visible on GitHub.

14.9 Exercise 9: Pull and Sync

Create a file via GitHub's web interface, then fetch the changes to your local repository.

14.10 Exercise 10: Stashing

1. Make changes to a tracked file
2. Stash the changes
3. Edit a file via GitHub and commit
4. Edit the same file locally
5. Use stash to fetch changes
6. Push all changes to GitHub

14.11 Exercise 11: Non-Conflicting Merge

Create two non-conflicting commits (one remote, one local) and merge them using `git pull`.

14.12 Exercise 12: Resolve Conflicts

Intentionally create a merge conflict and resolve it manually. Push the resolved result to GitHub.

14.13 Exercise 13: History Exploration

1. Create `secret.txt` with content and commit
2. Delete `secret.txt` and commit the deletion
3. Push to GitHub
4. Find the secret using commit history (both GitHub and command line)

14.14 Exercise 14: Clone Repository

Clone your repository to a different folder, make changes, and push them back.

14.15 Exercise 15: Team Formation

Meet your team members and document:

- Team member names and GitHub usernames
- Team name
- Team strengths and weaknesses
- Communication platform choice

14.16 Exercise 16: Project Repository Setup

1. Create a GitHub organization
2. Create a public repository in the organization
3. Initialize a Spring Boot project with required dependencies
4. Push to GitHub
5. Have all team members clone and verify

14.17 Exercise 17: README Creation

Create a comprehensive `README.md` with:

- Project title as heading
- Project description
- Team members with GitHub profile links
