Session 1: Git Fundamentals & Local Workflow

Minor in AI, Batch 4

Study Notes

November 25, 2025

Contents

1 Summary

This foundational lecture introduces Git as a distributed version control system and covers essential workflows for managing code repositories. Key topics include understanding version control problems, Git's three-area architecture (Working Directory, Staging Area, Repository), basic commands, configuration setup, and best practices for commit messages. The session emphasizes the distinction between Git (local tool) and GitHub (cloud hosting service) while demonstrating practical workflows for initializing repositories, tracking changes, and pushing code to remote repositories.

2 Concepts Covered

2.1 Introduction to Version Control and Problems Without Git

Why Version Control Matters:

- Multiple people working on the same project need change tracking
- Manual versioning creates chaos (e.g., `project_final_v2_updated.doc`)

Problems Git Solves:

- **Tracking Changes**: Know exactly what changed between versions
- **Author Attribution**: Identify who made which changes
- **Code Recovery**: Restore deleted code from history
- **Concurrent Work**: Enable simultaneous work without conflicts

Analogy

Git is like a "time machine" or "smart photo album" that takes snapshots (commits) of your project, allowing you to travel back to any previous state.

2.2 What is Git?

Git is a **distributed version control system** (DVCS):

- Software tool installed locally on your machine
- Tracks changes and maintains complete project history
- Works **offline** - every developer has a full copy of history
- Enables branching, merging, and collaborative development

2.3 Difference Between Git and GitHub

Feature	Git	GitHub
Type	Version control software	Cloud hosting service
Location	Local machine	Online/Remote
Internet	Works offline	Requires internet
Purpose	Track changes & history	Share & collaborate
Alternatives	-	GitLab, Bitbucket, Azure DevOps

Table 1: Git vs GitHub Comparison

Key Takeaway: Git is the tool; GitHub is where you store and share your Git repositories.

2.4 Installing and Configuring Git

Configuration Commands:

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

Why Configure?

- Links your commits to your identity
- Email should match your GitHub account for proper attribution
- Required before making your first commit

2.5 Git Configuration Levels

Git has **three configuration levels**:

1. **System** (`--system`): Applies to all users on the machine
2. **Global** (`--global`): Applies to current user across all projects
3. **Local** (`--local`): Applies to specific repository only

Priority Order: Local > Global > System (lower levels override higher ones)

2.6 Git Architecture: Working Directory, Staging Area, Repository

Git manages your project through **three main areas**:

```
 Working Directory → Staging Area → Repository
(Edit)               (git add)      (git commit)
```

1. **Working Directory**: Where you create and modify files
2. **Staging Area**: Preparation zone for changes you want to commit
3. **Repository**: Permanent storage of committed snapshots

File States:

- **Untracked**: New files not yet added to Git
- **Staged**: Files added via `git add`, ready to commit
- **Committed**: Snapshot saved in repository
- **Modified**: Changes made after last commit

2.7 Creating Git Repositories: Cloning vs Initializing

Two Ways to Start:

Option 1: Clone Existing Repository

```
git clone <repository-URL>
```

- Downloads complete repository from GitHub/remote
- Automatically sets up remote connection

Option 2: Initialize New Repository

```
git init
```

- Creates new Git repository in current folder
- Used for starting fresh projects locally

2.8 Hidden .git Directory and Its Role

What is `.git`?

- Hidden folder created by `git init` or `git clone`
- Contains all Git metadata and version history
- Stores: branches, commit history, configuration, hooks, pointers

Important

Deleting `.git` removes all Git tracking and remote connections. Don't delete it unless you want to completely remove version control!

2.9 Basic Git Workflow: add, commit, push

Complete Workflow:

```
# 1. Check status
git status

# 2. Stage changes
git add <filename>
git add .  # Stage all changes

# 3. Commit with message
git commit -m "Descriptive message"

# 4. Push to remote
git push origin <branch-name>
```

Process Flow:

1. Make changes in working directory
2. Stage files you want to include
3. Commit staged changes locally
4. Push commits to remote repository

2.10 Understanding Git Status

`git status` shows:

- Current branch name
- Files in staging area (ready to commit)
- Modified files not yet staged

- Untracked files (new files)
- Whether local branch is ahead/behind remote

Use it frequently to understand your repository state!

2.11 Difference Between `git add` and `git commit`

Command	Purpose	Effect
`git add`	Stages changes	Moves files to staging area (preparation)
`git commit`	Saves snapshot	Creates permanent record in repository

Table 2: git add vs git commit

Analogy

- `git add`: Putting items in a shopping cart
- `git commit`: Completing the purchase

You can add multiple files before committing them all together in one snapshot.

2.12 Unstaging Files

Remove files from staging area:

```
git restore --staged <filename>
```

This keeps your changes in the working directory but removes them from the staging area (undoes `git add`).

2.13 Committing with Good Commit Messages

Every commit records:

- Changed files
- Author name and email
- Timestamp
- Commit message
- Unique commit ID (SHA hash)

Best Practices for Commit Messages:

DO:

Keep subject line to 50 characters

Capitalize first letter

Use imperative mood (add, fix, update)

Be clear and descriptive

Explain **what** and **why**, not how

DON'T:

× Write vague messages ("fixed stuff", "changes")

× Make it too long

× Add period at end of subject

× Blame others or be non-descriptive

Good Example:

```
feat: add user authentication feature

Introduced JWT-based authentication; users can now register,
login, and maintain sessions.
```

Bad Examples:

- "fixed bug"
- "updated files"
- "asdfghj"

2.14 Git Push and Remote Repositories

Connecting Local to Remote:

```
git remote add origin <repository-URL>
```

Pushing Changes:

```
git push origin <branch-name>
```

What happens:

- Uploads local commits to GitHub/remote repository
- Makes your changes accessible to collaborators
- Backs up your work in the cloud

2.15 Branches and Default Branch

What are Branches?

- Separate lines of development
- Allow multiple developers to work on different features simultaneously
- Prevent conflicts during parallel work

Default Branch:

- Usually called `main` (previously `master`)
- Primary branch where stable code lives

Note: Branching and merging strategies will be covered in detail in future sessions.

3 Key Takeaways

1. **Always check status**: Use `git status` frequently to understand your repository state
2. **Stage intentionally**: Only add files you want to commit
3. **Write meaningful commits**: Future you (and your team) will thank you
4. **Git $\neq$ GitHub**: Git is local; GitHub is remote hosting
5. **Configuration matters**: Set up your identity before first commit
6. **Don't initialize twice**: Avoid running `git init` in the same directory multiple times
7. **Understand the flow**: Working Directory → Staging → Repository → Remote

4 Essential Commands Reference

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

# Repository Setup
git init                          # Initialize new repository
git clone <URL>                   # Clone existing repository
git remote add origin <URL>       # Connect to remote

# Daily Workflow
git status                        # Check repository state
git add <file>                    # Stage specific file
git add .                         # Stage all changes
git commit -m "message"           # Commit with message
git push origin <branch>          # Push to remote

# Unstaging
git restore --staged <file>       # Unstage file
```

This lecture establishes the foundation for using Git effectively in local development and collaborative workflows with GitHub.