Git and GitHub - Complete Teaching Notes

Table of Contents

Introduction to Version Control

What is Version Control?

Version control is a system that tracks changes to files over time. It allows you to:

- Keep track of every change made to your code
- Revert to previous versions when needed
- Collaborate with others without conflicts
- Maintain multiple versions of your project

Why Do We Need Version Control?

- **Backup**: Never lose your work
- **History**: See what changed and when
- **Collaboration**: Multiple people can work on the same project
- **Branching**: Work on different features simultaneously
- **Recovery**: Undo mistakes easily

What is Git?

Git is a **distributed version control system** created by Linus Torvalds in 2005.

Key Features:

- **Distributed**: Every user has a complete copy of the project history
- **Fast**: Operations are performed locally
- **Secure**: Uses SHA-1 hashing for data integrity
- **Flexible**: Supports various workflows
- **Free and Open Source**

Git vs Other Version Control Systems:

- **Centralized systems** (SVN, CVS): Single central repository
- **Git (Distributed)**: Every clone is a full backup

What is GitHub?

GitHub is a **cloud-based hosting service** for Git repositories.

GitHub Features:

- **Remote repositories** hosting
- **Web-based interface** for Git
- **Issue tracking** and project management
- **Pull requests** for code review
- **Actions** for CI/CD
- **Pages** for hosting websites
- **Social coding** features

GitHub vs Git:

- **Git**: The version control tool
- **GitHub**: A service that hosts Git repositories online

Installing Git

Windows:

1. Download from git-scm.com
2. Run the installer with default settings

3. Use Git Bash, Command Prompt, or PowerShell

Mac:

```bash
# Using Homebrew
brew install git

# Or download from git-scm.com
```

Linux (Ubuntu/Debian):

```bash
sudo apt update
sudo apt install git
```

Verify Installation:

```bash
git --version
```

Basic Git Concepts

Repository (Repo)

A directory containing your project files and Git's tracking information.

Working Directory

The current state of your project files that you can see and edit.

Staging Area (Index)

A intermediate area where you prepare changes before committing them.

Commit

A snapshot of your project at a specific point in time.

Branch

An independent line of development in your repository.

Remote

A version of your repository hosted elsewhere (like on GitHub).

The Three States of Git:

1. **Modified**: Files changed but not staged
2. **Staged**: Files marked to go into the next commit
3. **Committed**: Files safely stored in the Git database

Essential Git Commands

Initial Setup

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

# Check configuration
git config --list
```

Creating a Repository

```bash
# Initialize a new repository
git init

# Clone an existing repository
git clone https://github.com/username/repository.git
```

Basic Workflow Commands

```bash
```

```bash
# Check status of files
git status

# Add files to staging area
git add filename.txt        # Add specific file
git add .                   # Add all files
git add *.js                # Add all JavaScript files

# Commit changes
git commit -m "Commit message"
git commit -am "Add and commit in one step"

# View commit history
git log
git log --oneline           # Condensed view
git log --graph             # Visual representation
```

Working with Changes

bash

```bash
# See differences
git diff                    # Changes not staged
git diff --staged           # Changes staged for commit
git diff HEAD               # All changes since last commit

# Undo changes
git checkout -- filename    # Discard changes in working directory
git reset HEAD filename     # Unstage a file
git reset --soft HEAD~1     # Undo last commit, keep changes staged
git reset --hard HEAD~1     # Undo last commit, discard changes
```

Remote Repository Commands

bash

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

# View remotes
git remote -v

# Push to remote
git push origin main
git push -u origin main      # Set upstream branch

# Pull from remote
git pull origin main

# Fetch from remote (without merging)
git fetch origin
```

Working with GitHub

Creating a GitHub Repository

1. Log into GitHub
2. Click "New repository"
3. Fill in repository name and description
4. Choose public or private
5. Initialize with README (optional)
6. Click "Create repository"

Connecting Local Repository to GitHub

bash

```bash
# Method 1: Start with local repository
git init
git add .
git commit -m "Initial commit"
git remote add origin https://github.com/username/repo.git
git push -u origin main

# Method 2: Start with GitHub repository
git clone https://github.com/username/repo.git
cd repo
# Make changes, then add, commit, and push
```

GitHub Authentication

- **HTTPS**: Use personal access tokens
- **SSH**: Set up SSH keys for secure access

Setting up SSH:

```bash
# Generate SSH key
ssh-keygen -t ed25519 -C "your.email@example.com"

# Add to SSH agent
ssh-add ~/.ssh/id_ed25519

# Copy public key to GitHub settings
cat ~/.ssh/id_ed25519.pub
```

Branching and Merging

Understanding Branches

Branches allow you to work on different features or experiments without affecting the main codebase.

Branch Commands

```bash
```

```
# View branches
git branch              # Local branches
git branch -a           # All branches (local and remote)

# Create and switch to a new branch
git checkout -b feature-branch
# or (newer syntax)
git switch -c feature-branch

# Switch between branches
git checkout main
git switch main

# Merge a branch
git checkout main
git merge feature-branch

# Delete a branch
git branch -d feature-branch  # Safe delete
git branch -D feature-branch  # Force delete
```

Merge vs Rebase

```
bash

# Merge: Creates a merge commit
git merge feature-branch

# Rebase: Replays commits on top of target branch
git rebase main
```

Common Branching Strategies

- **Git Flow**: main, develop, feature, release, hotfix branches
- **GitHub Flow**: main branch with feature branches
- **GitLab Flow**: Similar to GitHub Flow with additional environment branches

---

Collaboration Workflows

Fork and Pull Request Workflow

1. **Fork** the repository on GitHub
2. **Clone** your fork locally

3. **Create** a feature branch
4. **Make** changes and commit
5. **Push** to your fork
6. **Create** a pull request

Pull Requests

Pull requests are a way to propose changes and discuss them before merging.

Creating a Pull Request:

1. Push your branch to GitHub
2. Go to the repository on GitHub
3. Click "New pull request"
4. Select branches to compare
5. Add title and description
6. Click "Create pull request"

Code Review Process

- Review code changes
- Leave comments and suggestions
- Request changes if needed
- Approve when ready
- Merge the pull request

Best Practices

Commit Messages

```bash
# Good commit messages
git commit -m "Add user authentication feature"
git commit -m "Fix navigation bug in mobile view"
git commit -m "Update README with installation instructions"

# Bad commit messages
git commit -m "fix"
git commit -m "changes"
git commit -m "stuff"
```

Commit Message Format:

```
Type: Brief description (50 characters max)

Detailed explanation if needed (wrapped at 72 characters)

- Bullet points for multiple changes
- Reference issues: Fixes #123
```

.gitignore File

Create a `.gitignore` file to exclude files from version control:

```gitignore
# Dependencies
node_modules/
*.log

# Build outputs
dist/
build/

# Environment variables
.env
.env.local

# IDE files
.vscode/
.idea/

# OS files
.DS_Store
Thumbs.db
```

Repository Structure

```
project/
├── README.md          # Project documentation
├── .gitignore         # Files to ignore
├── LICENSE            # License information
├── src/               # Source code
├── docs/              # Documentation
├── tests/             # Test files
```

```
└── .github/          # GitHub-specific files
    └── workflows/    # GitHub Actions
```

Common Issues and Solutions

Merge Conflicts

When Git can't automatically merge changes:

```bash
# After a merge conflict occurs
git status              # See conflicted files

# Edit files to resolve conflicts
# Look for markers: <<<<<<<, =======, >>>>>>>

# After resolving
git add conflicted-file.txt
git commit -m "Resolve merge conflict"
```

Undoing Mistakes

```bash
# Undo last commit but keep changes
git reset --soft HEAD~1

# Undo last commit and discard changes
git reset --hard HEAD~1

# Revert a commit (creates new commit)
git revert commit-hash

# Clean untracked files
git clean -fd
```

Lost Commits

```bash
```

```
# Find lost commits
git reflog

# Restore lost commit
git checkout commit-hash
git checkout -b recovery-branch
```

Practical Exercises

Exercise 1: Basic Git Workflow

1. Create a new directory and initialize Git
2. Create a `README.md` file
3. Add and commit the file
4. Make changes and commit again
5. View the commit history

Exercise 2: Working with Remotes

1. Create a repository on GitHub
2. Connect your local repository
3. Push your commits to GitHub
4. Make changes on GitHub and pull them locally

Exercise 3: Branching

1. Create a new branch for a feature
2. Make changes and commit
3. Switch back to main branch
4. Merge the feature branch
5. Delete the merged branch

Exercise 4: Collaboration

1. Fork a classmate's repository
2. Clone your fork locally
3. Create a feature branch
4. Make improvements and commit
5. Push to your fork

6. Create a pull request

Exercise 5: Handling Conflicts

1. Work with a partner on the same file
2. Both make different changes to the same lines
3. Try to merge and resolve the conflict
4. Complete the merge successfully

Quick Reference Sheet

Most Used Commands

```bash
git status                # Check repository status
git add .                 # Stage all changes
git commit -m "message"   # Commit with message
git push origin main      # Push to remote
git pull origin main      # Pull from remote
git checkout -b branch-name  # Create and switch to branch
git merge branch-name     # Merge branch
git log --oneline         # View commit history
```

Useful Aliases

Add these to your Git config for shortcuts:

```bash
git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.ci commit
git config --global alias.st status
git config --global alias.lg "log --oneline --graph --all"
```

Additional Resources

Documentation

- Official Git Documentation
- GitHub Docs
- Atlassian Git Tutorials

Interactive Learning

- Learn Git Branching
- GitHub Skills
- Git Immersion

GUI Tools

- **GitHub Desktop**: User-friendly interface
- **GitKraken**: Professional Git client
- **SourceTree**: Free Git client by Atlassian
- **VS Code**: Built-in Git integration

Command Line Tools

- **tig**: Text-mode interface for Git
- **lazygit**: Simple terminal UI for Git
- **hub**: Command-line tool for GitHub

Remember: The best way to learn Git and GitHub is through practice. Start with simple projects and gradually work on more complex collaborative scenarios.