My Git and GitHub Course Summary

Authored by: Fouenang Miguel Bruce

Introduction

This document is a personal summary of the concepts and practices I learned in the Git and GitHub version control course on Udemy. It includes technical instructions, personal insights, and tips for mastering Git and GitHub workflows.

Git and GitHub Overview

Git is a cross-platform version control system used to track changes in source code, while GitHub is a cloud-based hosting service for Git repositories. Together, they facilitate collaboration, change tracking, and versioning in development projects.

Installing Git

On macOS:

- Via Homebrew:

  ```
  brew install git
  git --version
  ```

- Via Xcode Command Line Tools:

  ```
  xcode-select --install
  git --version
  ```

On Linux (Ubuntu/Debian):

```
sudo apt update
sudo apt install git
git --version
```

Setting Up GitHub Account

1. Go to github.com
2. Sign up and follow the instructions
3. Configure Git locally:

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

Creating a GitHub Repository

```
git init
git add .
git commit -m "Initial commit"
```

- The `.git` folder is created to start tracking.
- `git add` stages changes.
- `git commit` saves the snapshot.

Understanding Git Areas

- **Working Directory**: The local directory where files are edited.
- **Staging Area**: Files added with `git add` are staged here.
- **Repository**: After `git commit`, changes are saved to the repository.

Git saves snapshots of changes, not full file versions, which allows fast performance.

Branching

```
git branch feature-xyz  # Create a branch
git switch feature-xyz  # Switch to the branch
```

Each branch enables isolated development. Changes on one branch won't affect others. Merging:

```
git merge feature-xyz
```

If a merge conflict occurs, Git will prompt you to resolve it manually.

Connecting to GitHub & Pushing Changes

Create a repository on GitHub and then:

```
git remote add origin <repo-url>
git branch -M main
git push -u origin main
```

Forking and Cloning

Fork to copy someone else's repo:

```
git clone <repo-url>
cd cloned-repo
```

Make changes in a new branch and push.

Merge Conflicts

Conflict Example:

```
CONFLICT (content): Merge conflict in file.txt
```

Fix:

- Edit file and resolve the conflict between `<<<<<<<`, `=======`, `>>>>>>>`
- Stage and commit:

```
git add file.txt
git commit -m "Resolve merge conflict"
```

Common Git Errors & Fixes

1. **Detached HEAD**:

```
git checkout main
# Or to keep changes:
git switch -c new-branch-name
```

1. **Push Rejected**:

```
git pull origin main
# If conflict:
git pull --rebase origin main
```

1. **Wrong Commit Branch**:

```
git switch correct-branch
git cherry-pick <commit-id>
git switch wrong-branch
git reset --hard HEAD~1
```

1. **Forgot to Add File**:

```
git add missing-file.txt
git commit --amend
```

Pro Tip

Use `git status` often to stay informed.

Writing Effective Commit Messages

Structure:

1. **Title** (max 50 characters)
2. **Body** (optional detailed explanation)
3. **Footer** (references to issues/PRs)

Examples:

```
feat: add login validation
fix: correct typo in README
docs: update SSH key instructions
```

Bad Examples:

```
update
more changes
fix stuff
```

Use present tense and conventional tags (`feat`, `fix`, `docs`, etc.).

Using Vim for Git Commit

If `git commit` opens Vim:

1. Press `i` to insert
2. Write message
3. Press `Esc`, then type `:wq`

Or use:

```
git commit -m "Your message"
```

GitHub Actions (Automation)

GitHub Actions allow automating workflows:

- Create `.github/workflows/main.yml`
- Define tasks (CI/CD, tests, deployment, etc.)

Example:

```
name: CI
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - run: echo "Running CI"
```

Git Rebase (Advanced History Rewriting)

`git rebase` moves or combines a sequence of commits to a new base commit.

Example:

```
git switch feature-branch
git rebase main
```

This integrates the latest changes from `main` into `feature-branch` linearly.

Benefits:

- Cleaner project history
- Avoids merge commits

If conflicts occur, Git will ask for manual resolution.

To abort:

```
git rebase --abort
```

To continue:

```
git rebase --continue
```

Git Squash (Combine Commits)

Squashing allows you to merge multiple commits into one.

Interactive example:

```
git rebase -i HEAD~3
```

You'll see a list like:

```
pick abc123 Add login
pick def456 Fix login bug
pick ghi789 Improve UI
```

Change the second and third to `squash`:

```
pick abc123 Add login
squash def456 Fix login bug
squash ghi789 Improve UI
```

This combines all into one commit.

Use it to clean up your history before merging a PR.

Git Stash (Temporary Save)

`git stash` saves changes in your working directory without committing them.

Example:

```
git stash save "work in progress"
git stash list
git stash apply # to reapply
```

Useful when you want to switch branches but keep your uncommitted changes.

You can also drop or pop stashes:

```
git stash pop
```

Final Word This document was created by *Fouenang Miguel Bruce* as a complete reference to support Git/GitHub learning and development practices. Keep building, collaborating, and committing wisely!