Programming Techniques 2024-2025

Lecture 2: Basics of Git and GitHub

Hannu Parviainen

Universidad de la Laguna

September 30, 2024

What is Git?

Distributed Version Control System (VCS)

- Tracks changes in the source code and helps coordinate work among programmers.
- Designed to handle everything from small to very large projects quickly and efficiently.
- Allows multiple developers to work on a project simultaneously without overwriting each other's changes.

Distributed?

- Each user has a full copy of the repository.
- Allows for a more flexible workflow than centralised VCSs.
- Branching is fast and does not require a connection to the VCS server.

Why Use Git?

- **Collaboration:** Streamlines teamwork on projects.
- **Backup and Restore:** Safeguards code with version history.
- **Track History:** Allows you to see what changes were made and when.
- **Branching and Merging:** Supports multiple development lines.

Installing Git

Windows

Install via Git for Windows.

Mac

Install via Homebrew using ‘brew install git‘ or install Xcode.

Linux

Install via package manager using ‘sudo apt-get install git‘ or equivalent.

Configuring Git

Set your username and email for Git commits:

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

- These settings are used to attribute commits to you.
- The ‘–global‘ flag applies settings for all repositories.

Initializing a Repository

Initialize a new Git repository in the current directory:

```
$ git init
```

- Creates a new ‘.git‘ subdirectory.
- Starts tracking versions for your project.

Checking Status

Check the status of your working directory and staging area:

```
$ git status
```

- Shows tracked and untracked files.
- Indicates changes that are staged for commit.

How Git Tracking Works

- **Working Directory:** Your local filesystem where you modify files.
- **Staging Area (Index):** A temporary area where you add changes to prepare for a commit.
- **Repository (History):** The database where commits are stored.
- **Tracking Changes:**
  - Git monitors changes in tracked files.
  - Untracked files are not monitored until added.
- **Lifecycle of a File:**
  1. Modify files in the working directory.
  2. Stage changes using ‘git add‘.
  3. Commit changes to the repository with ‘git commit‘.

Staging vs. Committing Changes

- **Staging Changes (`git add`):**
  - Prepares selected changes for the next commit.
  - Allows you to review and group changes.
- **Committing Changes (`git commit`):**
  - Records the staged changes into the repository history.
  - Creates a new commit object with a unique ID.
  - Includes a commit message describing the changes.
- **Key Differences:**
  - **Staging:** Prepares changes, but does not save them to history.
  - **Committing:** Saves the staged changes permanently in the repository.

Staging Changes with `git add`

Add files to the staging area:

```
$ git add filename
$ git add .
```

- `git add filename`: Stages a specific file.
- `git add .`: Stages all changes in the current directory.
- **Staging:** Prepares changes to be included in the next commit.

Committing Changes with `git commit`

Commit the staged changes to the repository:

```
$ git commit -m "Commit message"
```

- Records a snapshot of the staging area.
- **Commit Message:** Describes the changes made.

Understanding Branches

- **What is a Branch?**
  - A parallel version of the repository.
  - Allows you to work on different features independently.
- **Why Use Branches?**
  - Isolate development work without affecting the main codebase.
  - Facilitate collaboration by allowing multiple features to be developed simultaneously.
- **Default Branch:** Typically named ‘main‘ or ‘master‘.

Creating and Switching Branches

Create a new branch and switch to it:

```
$ git branch new-feature
$ git checkout new-feature
```

Or combine both steps:

```
$ git checkout -b new-feature
```

- `git branch new-feature`: Creates a new branch named ‘new-feature‘.
- `git checkout new-feature`: Switches to the ‘new-feature‘ branch.
- `git checkout -b new-feature`: Creates and switches to ‘new-feature‘ in one command.

Merging Branches

Merge changes from ‘new-feature‘ branch into ‘main‘:

```
$ git checkout main
$ git merge new-feature
```

- `git checkout main`: Switches to the ‘main‘ branch.
- `git merge new-feature`: Merges ‘new-feature‘ into ‘main‘.
- **Merging:** Combines changes from one branch into another.

What is GitHub?

- **Code Hosting Platform:** Hosts Git repositories online.
- **Facilitates Collaboration:** Tools for team communication and coordination.
- **Additional Features:**
  - Issue tracking.
  - Pull requests for code reviews.
  - Wiki pages and documentation.

Setting Up GitHub

- **Create an Account:** Sign up at github.com.
- **Set Up SSH Keys:**
  - Generate an SSH key pair on your local machine.
  - Add the public key to your GitHub account.
- **Configure Git to Use SSH:**
  - Ensures secure communication with GitHub.

Connecting a Local Repository to GitHub

Add a remote repository and push changes:

```
$ git remote add origin git@github.com:username/repo.
    git
$ git push -u origin main
```

- `git remote add origin`: Links your local repo to GitHub.
- `git push -u origin main`: Pushes commits to the ‘main‘ branch on GitHub.
- **The `-u` Flag:** Sets ‘origin main‘ as the default upstream branch.

Collaborating on GitHub

- **Forking Repositories:** Create a personal copy of someone else's project.
- **Cloning Repositories:** Download a repository to your local machine.
- **Pull Requests:** Propose changes to a repository.
- **Code Reviews:** Collaboratively review code changes before merging.
- **Issue Tracking:** Report bugs or request features.

Best Practices

- **Write Meaningful Commit Messages:** Clearly describe what changes were made.
- **Keep Commits Small and Focused:** Easier to review and understand.
- **Regularly Pull Updates:** Keep your local repository up-to-date with the remote.
- **Use Branches for New Features:** Isolate development work.
- **Avoid Committing Sensitive Information:** Don't commit passwords or API keys.