Full Stack Development Coaching

Module 1: The Developer's Toolkit
Bash, Git, and GitHub

Rezig Hamza

Contents

1. Introduction to the Shell

1.1 What is the Shell?

The shell is a powerful program that provides a text-based interface to your computer's operating system. Instead of clicking on icons and menus, you interact with your computer by typing commands. For developers, proficiency with the shell is not optional—it is a fundamental skill.

- **Efficiency:** Performing complex tasks like searching for files, installing software, or managing servers is often much faster in the shell.
- **Automation:** You can write scripts to automate repetitive tasks, saving you countless hours.
- **Power:** The shell gives you direct, unfiltered access to your system's capabilities, which is essential for development, deployment, and debugging.

On Unix-based systems (like Linux and macOS), the most common shell is **Bash** (Bourne Again SHell). On Windows, common options include PowerShell, Git Bash (which we recommend for this course), or the Windows Subsystem for Linux (WSL).

1.2 Basic Shell Commands

These are the foundational commands you will use every single day. Let's explore how to navigate and manipulate the filesystem.

1.2.1 pwd - Print Working Directory

Tells you exactly where you are in the filesystem hierarchy. Think of it as the "You are here" marker on a map.

dev@machine:

```
$ pwd
/home/dev
```

1.2.2 ls - List Files and Directories

Lists the contents of the current directory. By default, it hides files and directories that start with a dot (`.`). To see everything, we use the `-a` (all) flag.

dev@machine:

```
$ ls
Desktop  Documents  Downloads  Projects
```

dev@machine:

```
$ ls -a
.  ..  .bashrc  .profile  Desktop  Documents  Downloads  Projects
```

Notice the new entries `.`, `..`, and `.bashrc`. These are hidden by default.

1.2.3 cd - Change Directory

Allows you to navigate to a different directory.

dev@machine:

```
$ cd Projects
$ pwd
/home/dev/Projects
```

Understanding the File Structure The filesystem on Linux and macOS is a tree-like hierarchy, starting from the "root" directory (`/`). Every file and folder on your system exists somewhere inside this tree.

Example File Tree

```
/ (root)
|-- home/
|   `-- dev/  (Your Home Directory, also known as ~)
|       |-- .bashrc (a hidden configuration file)
|       |-- Documents/
|       |   `-- report.pdf
|       `-- Projects/
|           `-- my-webapp/
```

With this structure in mind, let's clarify the special directories you saw with `ls -a`:

- `.` refers to your **current** directory.
- `..` refers to the **parent** directory (one level up the tree).

So, if you are currently inside `/home/dev/Projects`, running `cd ..` will take you up one level to `/home/dev`.

dev@machine: /Projects

```
$ pwd
/home/dev/Projects
$ cd ..
$ pwd
/home/dev
```

1.2.4 mkdir - Make Directory

Creates a new directory. Let's create a folder for our first project.

dev@machine: /Projects

```
$ mkdir my-webapp
$ ls
my-webapp
```

1.2.5 touch - Create a File

Creates a new, empty file.

dev@machine: /Projects

```
$ cd my-webapp
$ touch index.html
$ ls
index.html
```

2. Version Control with Git and GitHub

2.1 What is a Version Control System (VCS)?

A Version Control System is a tool that tracks and manages changes to your code. Think of it as a "save game" system for your projects. It allows you to:

- **Track your changes:** See exactly who changed what, when, and why.
- **Manage your code better:** Keep a clean history of your project's evolution.
- **Work in parallel:** Safely experiment with new features without breaking the main codebase.
- **Collaborate effectively:** It is the backbone of modern team-based software development.

Git is the world's most popular distributed version control system. **GitHub** is a web-based platform that hosts your Git repositories and provides tools for collaboration.

2.2 The Core Git Workflow: The Three Areas

To master Git, you must understand its three-stage process for saving work. Every file in your project is in one of three states:

1. **Working Directory:** Your actual project folder, containing the files you are currently editing.
2. **Staging Area (or "Index"):** A "drafting" area. You add snapshots of your files here to prepare them for the next permanent save. This allows you to craft precise, meaningful saves (commits).
3. **Repository (`.git` directory):** The permanent, unchangeable history of your project. The ‘commit‘ command takes what's in your staging area and saves it forever in the repository.

2.3 Local Operations: Your First Repository

2.3.1 git init: Initializing a Repo

This command turns a regular directory into a Git repository. It creates a hidden `.git` folder where Git stores all its history and metadata. You only run this once per project.

dev@machine: /Projects/my-webapp

```
$ git init
Initialized empty Git repository in /home/dev/Projects/my-webapp/.git/
```

2.3.2 git status: Checking the Status

This is your most-used command. It tells you the current state of your working directory and staging area. Let's see the status of our new repo.

dev@machine: /Projects/my-webapp(*main*)

```
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

nothing added to commit but untracked files are present (use "git add" to track)
```

Git sees our `index.html` but tells us it's "untracked."

2.3.3 git add: Staging Files

This command moves changes from the working directory to the staging area. You can stage a specific file or all files.

To stage our `index.html` file:

dev@machine: /Projects/my-webapp(*main*)

```
$ git add index.html
```

Now let's check the status again.

dev@machine: /Projects/my-webapp(*main*)

```
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html
```

Notice the file is now green and listed under "Changes to be committed." It's ready to be saved permanently.

2.3.4 git commit: Committing Files

This command takes everything in the staging area and saves it as a permanent snapshot in the repository. Each commit has a unique ID and a message describing the changes. **Good commit messages are crucial!**

dev@machine: /Projects/my-webapp(*main*)

```
$ git commit -m "feat: Add initial index.html page"
[main (root-commit) a1b2c3d] feat: Add initial index.html page
 1 file changed, 1 insertion(+)
 create mode 100644 index.html
```

The `-m` flag lets you provide the message inline. Now our changes are safely saved.

2.4 Branching and Merging: Parallel Development

A branch is a movable pointer to a commit. It allows you to create a separate line of development to work on a feature or a bug fix without affecting the main codebase (the `main` branch).

2.4.1 git checkout: Creating and Switching Branches

Let's create a new branch to add a CSS file. The best practice is to use the `-b` flag, which creates the new branch and immediately switches to it.

dev@machine: /Projects/my-webapp(*main*)

```
$ git checkout -b feat/add-styling
Switched to a new branch 'feat/add-styling'
```

2.4.2 git merge: Combining Branches

Once the feature is complete, we merge it back into our main branch. First, switch back to the target branch (`main`), then run the merge command.

dev@machine: /Projects/my-webapp$(feat/add - styling)$

```
$ # Imagine we've added and committed a style.css file on this branch
$ git checkout main
Switched to branch 'main'
$
$ git merge feat/add-styling
Updating a1b2c3d..2f8e1a9
Fast-forward
 style.css | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 style.css
```

2.5 Working with Remotes: GitHub Workflow

So far, everything has been on our local machine. To collaborate or back up our code, we use a "remote"—a version of our repository hosted on a server like GitHub.

2.5.1 git push: Sending Changes to a Remote

After making and committing changes locally, you use `git push` to upload your commits to the remote repository.

dev@machine: /Projects/my-webapp(*main*)

```
$ # First, we need to add our GitHub repo as a remote.
$ git remote add origin https://github.com/your-username/my-webapp.git
$
$ # Now, we can push our main branch to origin.
$ git push -u origin main
Enumerating objects: 6, done.
...
To https://github.com/your-username/my-webapp.git
 * [new branch]      main -> main
```

2.5.2 git pull: Fetching and Merging from a Remote

If a teammate has pushed changes to the remote, you need to get those changes onto your local machine. `git pull` does this by fetching the changes and immediately merging them into your current branch.

dev@machine: /Projects/my-webapp(*main*)

```
$ git pull origin main
From https://github.com/your-username/my-webapp
 * branch            main       -> FETCH_HEAD
Already up to date.
```