2.1.3 Create a Development Environment

The first thing we need to do is set up our development environment. Because the current codebase is live, we can't make changes to it without disturbing Run Buddy's users. This is where Git comes into play.

Git allows developers to create copies of the codebase, known as **branches**. Different branches are used to represent different environments. Here's how this applies to Run Buddy:

- The Run Buddy site runs from the `master` branch—this is known as the production environment. The **production environment** is where the "live" website or application is housed. This is the finished product that people are using in real-time.

- The branch we'll create to work on will be a copy of the `master` branch—this will be our development environment. The **development environment** is where developers make changes to the software. Using a separate environment and branch allows changes to safely be made without affecting the website or application that people are currently using.

As a rule, production environments should never be worked on directly because any bugs or errors that are introduced will be immediately

apparent to users. Git branching lets us isolate versions of the code for development and testing. This is known as **working in isolation**.

HIDE PRO TIP

Branch names are just labels that can be changed. The `master` branch does not always have to be the production environment, but in our scenario this will be the case.

For Run Buddy, we'll use a `develop` branch as our development environment. It will be where we develop and test new features. When we're satisfied with the result and have a stable working version, we'll **merge** it into the `master` branch and update the live website.

Important Git Commands

Before we create our development environment, let's learn some new Git commands:

- `git branch` displays a list of all the branches in the repo. It also highlights the branch you're in, which is called the **active branch** or **working directory**.
- `git branch <branch-name>` creates a new branch (with your `<branch-name>`) that is a version of the current active branch. After you commit changes to the new branch, the branches will no longer be in sync with each other because their commit histories will diverge.
- `git checkout <branch-name>` lets you switch into a different branch, which is then the new active branch. This is how to navigate between different versions of the codebase.

- `git checkout -b <branch-name>` is two commands in one. It creates a new branch (`git branch <branch-name>`) and moves (`git checkout <branch-name>`) into that branch, making it the active branch.

Now let's create that `develop` branch. First, go to the command prompt and `cd` into your working root directory.

ON THE JOB

In most workplaces, these branches will have already been configured by the IT department or development operations (DevOps) team.

In the root directory, type the following commands to create a new `develop` branch from the `master` branch, pressing Enter after each one:

```
git branch develop
git checkout develop
```

Another way to do this is to use the following command, which combines these two commands into one:

```
git checkout -b develop
```

Now you'll list all the branches in the repo to verify that you created the branch:

```
git branch
```

Your terminal should now look like this:

The green font color and asterisk indicate the active branch.

`git branch` could be one of the most commonly used Git commands because it's important to know which branch you're in when undertaking any Git workflow operation. Get in the habit of using it often!

Don't worry if you're having a hard time memorizing these Git commands. A helpful tip is to add a bookmark in your browser for a Git command cheatsheet such as **Atlassian's Git cheatsheet (https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet)** .

Let's move on to another invaluable function of branching: creating feature branches.