Vim — The Programmer's Editor

AiCE Software Engineering Foundations — The Developer Environment

Every file you create in the terminal eventually needs to be written or edited from inside the terminal too — and that requires a text editor that lives there. This lesson introduces you to Vim, one of the most powerful text editors ever built, and the reason every professional developer eventually learns to use it.

1. Why Learn Terminal Text Editors?

When you deploy a web application to a server, or provision a cloud instance, the machine you connect to will almost certainly have no graphical interface at all. There will be no mouse, no File Explorer, no double-clickable icons — only a terminal, and a text editor running inside it.

This is not a rare edge case. It is the daily reality of backend development, server administration, and cloud engineering. Any developer who works beyond their own local machine needs to be comfortable editing files entirely from the command line.

> **Two Editors, One Requirement**
> *The two most widely available terminal text editors are Vim and Nano. Vim is more powerful but has a steeper learning curve; Nano is simpler and friendlier for beginners. Every professional developer is expected to know both — this document covers Vim, and the next covers Nano alongside a direct comparison of the two.*

2. Checking Installation

Before using Vim, confirm that it's installed on your system:

```
vim --version
```

If this produces output showing a version number, Vim is installed and ready to use. If you instead see a “command not found” error, install it with:

```
sudo apt install vim
```

sudo and apt

sudo temporarily grants you administrator (root) privileges. apt is Ubuntu's package manager — think of it as an app store for the terminal. You will use both of these tools often as your skills grow.

3. Vim's Modal Design

Vim (Vi Improved) is available on virtually every Unix-based system, and many of the world's most productive developers write code exclusively inside it. Its power comes from a modal design: rather than always being ready to accept typed text, Vim has distinct modes for distinct tasks. This feels unusual at first, but becomes second nature with practice.

Vim operates in two primary modes:

Mode	Purpose	How to Activate
Command Mode	Navigate, save, quit, run Vim commands	Default when Vim opens (press Esc to return)
Insert Mode	Type and edit text freely	Press i in Command Mode

The bottom of the Vim interface always tells you which mode you're in. In Insert Mode, you'll see -- INSERT -- displayed there. In Command Mode, that indicator disappears.

4. Opening, Creating & Inserting Text

To open an existing file, or create a new one, use:

```
vim filename.txt
```

If the file already exists, Vim opens it. If it doesn't, Vim creates a new empty buffer under that name, which becomes an actual file only once you save it.

When Vim opens, you start in Command Mode — you cannot type text yet. Press i to switch into Insert Mode:

```
i
```

The -- INSERT -- indicator appears at the bottom, and you can now type freely, just as you would in any ordinary text editor.

5. Saving and Quitting

When you're done typing, press Escape to return to Command Mode — the -- INSERT -- indicator disappears. From Command Mode, typing a colon (:) followed by a letter runs a Vim command.

To save the file:

```
:w
```

Press Enter, and the file name appears at the bottom, confirming the save. To quit Vim afterward:

```
:q
```

To save and quit in a single step:

```
:wq
```

And to quit without saving — discarding any changes entirely:

```
:q!
```

> **The Exclamation Mark**
> *The ! in :q! forces Vim to exit even with unsaved changes present. Use it deliberately — anything not saved beforehand is lost permanently the moment you confirm.*

Command	Action
i	Enter Insert Mode (start typing)
Esc	Return to Command Mode
:w	Save the file
:q	Quit Vim (only works if the file is saved)
:wq	Save and quit simultaneously
:q!	Force quit without saving

6. A Complete Vim Workflow

Put together, a full round trip of creating and writing a file in Vim looks like this:

```
vim africa_report.txt        # 1. open (or create) the file
i                             # 2. enter Insert Mode
                              # 3. type your content freely
<Esc>                         # 4. return to Command Mode
:wq                           # 5. save and quit
cat africa_report.txt         # 6. confirm the file contents
```

This same shape — open, insert, escape, save, confirm — covers the overwhelming majority of what you will do with Vim day to day.

7. Reading a File Without Editing It

Opening a file with Vim purely to read it, without making any changes, is perfectly normal and works the same way:

```
vim africa_report.txt
```

Read the content, then type :q to quit. Since no changes were made, Vim exits immediately without asking you to save anything.

> **If You Type by Accident**
> *If you accidentally press i and type something while reading, press Escape immediately, then :q! to discard the unintended change rather than saving it.*

8. What Happens When You Forget to Save

Suppose you make changes to a file and then try to quit without saving them. Vim refuses, and shows an error along these lines:

```
E37: No write since last change (add ! to override)
```

This is Vim protecting you from losing work by accident. At that point you have exactly two options: type :wq to save and quit, or type :q! to force quit and discard the changes deliberately.

Key Takeaways

Vim's entire workflow rests on the switch between Command Mode and Insert Mode — press i to type, Escape to stop, and a colon command to save or quit. It feels unusual for the first few files, but the same five-step pattern (open, insert, escape, save, confirm) repeats so often that it becomes automatic. The next document introduces Nano, a far gentler alternative, and lays out exactly when each editor is the right tool for the job.