# Getting Started with VIM

## What is VIM

Vim is a text editor that comes pre-installed in **Linux** and **macOS**. It can help you pick up your typing speed and be more productive. It forces you to use the keyboard as much as possible. By doing that, it makes your coding speed faster as well. Its really basic looking interface can be changed by using plugins but that's, of course, optional.

## How to use VIM

There's two ways to do that:

1. You can just type `vi` on the terminal and hit enter.
2. Or you can also use the `vim` command to do the same.

They are both the same but the `vim` command is for the newer version, known as **Vim IMproved**.

If you running the `vim` command gives you error, you have to install it first. You can do that by using the following command:

```
sudo apt install vim
```

## Editing text with VIM

Change directory to your project. Then run the command `vim` followed by the file that you want to edit. If the file exists, it will open it. If it doesn't, it'll first create it in memory and then open it.

VIM has two modes. By default, it will open the **VIM** editor in **command mode**. While the command mode is turned on, you can run commands that sepcific to the VIM editor only. To do that, you have to type `:` and then your command.

The other one mode is **insert mode**. While this mode is on, you can edit your text files.

Let's start with **command mode**! 🤩

### Command Mode

As I said before, this is the default mode VIM opens up in when we first launch it. Let's learn how to use by running some commands and changing how our VIM editor looks 🥳

If you don't see a `:` at the bottom of your VIM, well type it! 😁 Now type `set number` and hit enter. Do it and see the result for yourself 🙃

Run another by typing `colorscheme slate`. This is a personal preference, you can opt from other themes as well. To find out what themes are available, press the **tab** key after typing `colorscheme`. That will suggest you all the themes and you should try all of them one by one 🥰

Then run `set autoindent`. This will indent your cursor when you're inside a code block.

Also run `set tabstop=2`. This is used for setting the size of your **tab**. Setting it 2 means, when you press the **tab** key, it will insert two spaces. This is also how much the auto indent will follow.

## Let's do practical

To help you see this better, let's create a file visually.

```
~$ vim main.cpp
```

This will open our file (if existed) otherwise it will create a temporary file with our specified name (`main.cpp` in this case) and then open it.

```
~
~
~
~
~
~
~
~
"main.cpp" [new]
```

I didn't have the `main.cpp` file so, in my case, this is how it looked like.

And right now, this is in **command mode**. Let's run those commands that we mentioned above. And to save some space, I will only show the bottom of the screen.

I will just type `:` and it will look something like this:

```
~
~
:
```

Let's try the `set number` command.

```
~
:set number
```

This will show the *line number* while writing your code or any text file. And your editor shall now look something like:

```
1. |
~
~
~
~
~
~
~
:set number
```

You can and should try all of the other commands mentioned earlier. I'll show you one more here.

```
~
:set mouse=a
```

This will allow you to select and move around the text using your *mouse* like any other text editor.

**INSERT Mode**

There are multiple ways that you can enter the insert mode depending on where you want to insert. Let's go over them one by one.

🔥By pressing `i` from the keyboard... This is the normal way of entering the **insert mode**. This activates the cursor at the previous position when the **command mode** was activated. By default, it's the beginning of the file.

As soon as you hit `i` from your keyboard when in **command mode**, you will see the things changing.

```
1. |
~
~
~
-- INSERT --
```

The INSERT at the bottom indicates that you are in the insert mode. It will stay there as long as you don't press `esc` key. Pressing the `esc` key, switches the mode to **command mode**.

But let's stay in the insert mode and let's write some code now. `C++` code, to be precise.

```
#include <iostream>

int main() {
  std::cout << "Hello World\n";

}
~
~
~
-- INSERT --
```

Let's now save this file. For that, we need to switch back to the **command mode**. Hit `esc` key and the INSERT text from the bottom will vanish. Type the command to write/save your content. The command is `w`.

```
}
~
~
:w
```

To get out of the **VIM** editor back into terminal, you can use the quit command which is `q`.

```
6. }
~
~
:q
```

Alternatively, you could execute both of these commands at the same time. To do that, type `:wq`. This will write, and quit the file.

You will be brought back to the terminal. Let's compile and run our code using `g++`.

```
~$ |
```

## Compiling the code

Type the command:

```
~$ g++ main.cpp && ./a.out
```

This will compile and run the code. To give you some extra information about the `g++` command, the `g++` in the command above is the name of the compiler. `main.cpp` is the name of the file. These two are the required keywords. Optionally, you can give flags to the command like `-o` which means to what file we want our binaries to be in. That is actually the binary code that will eventually be run by the processor.

Another very useful flag to the `g++` command, is the **standard** for `C++`. If you don't know, C++ has many versions. **C++23** is the latest. They've been releasing a new standard every three years since 2011.

To specify the standard that you want to compile your code with, you can use the `-std` flag. If say, you wanted to compile `main.cpp` with **C++20**, you would specify that by executing the following command.

```
g++ -std=c++2a main.cpp
```

Now, let's get back to the **VIM** editor. Type the same command as before: `vim main.cpp` and it will launch the editor in the command mode.

### Different INSERT Modes

🔥Pressing `o` key from the keyboard will also enter the **insert mode** but it will insert a new line **below** whichever line the cursor was. Consider the following example when the cursor is on the second line and I press `o`.

```
#include <iostream>
|
int main() {
  std::cout << "Hello World\n";

}
~
~
~
"main.cpp" 6L, 62B
```

As soon as I press the `o` key,

```
#include <iostream>

|
int main() {
  std::cout << "Hello World\n";

}
~
~
~
-- INSERT --
```

Notice that it inserted a new line below the position of the cursor and also entered the **insert mode**.

🔥Pressing `a` key from the keyboard will enter the **insert mode** in the **append mode**. What that means is that; unlike the normal insert mode where, when you enter it, the characters are inserted after the cursor, the characters are inserted before the cursor's position. Let's see that in action:

```
#include <iostream>


int main() {
  std::cout << "Hello World\n";

}
~
~
~
-- INSERT --
```

While I'm in **append insert mode**, if I enter a character, it'll be inserted before the cursor. Let's enter something:

```
#include <iostream>

int msain() {
  std::cout << "Hello World\n";

}
~
~
~
-- INSERT --
```

I entered **s** and it inserted it behind the cursor's position.

You can try the **upper cases** of these insert modes as well! What that means is that you can also, when the **command mode** is active, press `I`, `O`, and `A`.

- `I` will enter the **insert mode** and take the cursor to the beginning of the current line.
- `O` will insert a new line above the line where the cursor was and then enter the **insert mode**.
- `A` will take the cursor to the end of the line.

## Selecting text

Let's now talk about how you can select text.

Pressing `v` while in **command mode** will turn on the **visual mode**. Let's have a look.

```
#include <iostream>

int main() {
  std::cout << "Hello World\n";

}
~
~
~
-- VISUAL --
```

While in **visual mode**, press the arrow keys from the keyboard to select text in any direction. Let's select some stuff:

```
#include <iostream>

int main() {
  std::cout << "Hello World\n";

}
~
~
~
-- VISUAL --
```

Pretty much every command can be mixed with numbers. Type in any number before hitting any command and see the result.

Let's practice that on selecting some text. If press `5` and then ➡️, it will select 5 characters to the right side of the cursor. Similarly, you can type any number and then hit **up arrow** or **down arrow** and it will select that number of lines at once.

Similarly, you can try the **upper case** on this as well. `V` will select the entire line the cursor was on.

Another very useful way to select any block entirely, is by first entering the **visual mode** and then pressing `i` followed by the kind of block you're in that you want to select.

What that means is that, let's suppose, you're inside the `main()` function. Its block is enclosed between `{ }` brackets. To select the entire block, I can then hit `i{` while in **visual mode**. The results are:

```
#include <iostream>

int main() {
    std::cout << "Hello World\n";

}
~
~
~
-- VISUAL --
```

Because in this case, we only had one line, that's what got selected. You should try it with bigger blocks.

One thing that I see all the time among learners is that they only do what they're told like robots. Meaning: I've shown an example to select a block that's enclosed in `{ }`, but that's **not all**! You can use `" "`, `[ ]`, `( )` or whatever and this will still work the same.

## Copying & Pasting text

Now, that we know how to select any text, let's also **copy** and paste it.

Copying can be done by pressing `y` in **command mode**. This will copy anything that you'd have copied prior to the execution of the copying command. The **y** means to **yank**.

Another way to do it is, of course, the **upper case** `Y`. This will copy the whole line that your cursor was on.

To **paste**, you can simply take your cursor wherever and hit `p`. (Needless to say, in the **command mode**). This will paste the line that you copied below the line where your cursor is. If you want to paste it on the same line as your cursor, use the **upper case** version of the **paste** command. `P` will paste it on the same line.

As mentioned before, you can mix numbers with your commands. Try typing any number before hitting `p` or `P` and see the magic 🥺

## Deleting text

Let's now see how you can delete certain text.

Deleting is as easy as the other commands so far 😁. Press `d` and then use **arrow keys**. It will delete one character to whichever side you pick. Or it will delete the entire lines if you press the **up** or **down** key.

To delete the entire line at once, use the **upper case** version (i.e., `D` will delete the entire line that the cursor was on). The same thing can be done if you press the **lower case** `d` twice. The difference is that, **lower case** will also remove the line whereas the **upper case** will only delete the contents on that line. Try it yourself to understand it better.

*There's so many more ways of doing this, but this tutorial is only to get you started with using VIM. You can learn the others on your own later, or if you liked this one, I'll make one like this one with much more details.*

## Replacing text

You can also replace a character by pressing the `r` key. It will replace the character that was below the cursor with the one you type after pressing **r**. Immediately after, it will return back to the **command mode** though. If you want to keep replacing, use the **upper case** version. This will keep you in the **replace mode** and replace each character with whatever you type. Press `esc` to get out of the **replace mode**.

## Undoing & redoing

Don't use `ctrl + z` 🥴!

To **undo** your previous action, press `u`. You can also mix it with numbers and it will do that number of undos.

To **redo** your action, use `ctrl + r` keybinding.

### Moving around the text

You can move your cursor or you can just use your keyboard for that too. And I'm not talking about the arrow keys. You can use `w`, `e`, `W`, and `E` to move **forward**. There's not much difference among these. Try them yourself and observe yourself.

To move **backwards**, you can use `b` and `B`.

I'd like to remind you again that mixing numbers with commands is really fun. Try numbers with these commands that you just learned. Also, try mixing numbers with arrow keys. Try `10 ➡️` to move around.

## Saving your settings permanently

The settings that you set at the beginning will work until you don't restart. When you relaunch your VIM again, it'll fall back to its default. Let's fix that!

```
~$ vim ~/.vimrc
```

This will (if existed or create from scratch and then) open it. It will probably be empty. Enter the **insert mode** and start adding all of those commands. I'll show a demo here too:

```
set number
set autoindent
set tabstop=2
set mouse=a
colorscheme slate
~
~
~
~
~
-- INSERT --
```

You can add other settings as if you want. And you can also change this file later anytime. After that, press `esc` to enter the **command mode** and type the command `:wq` to save and close the file.

---