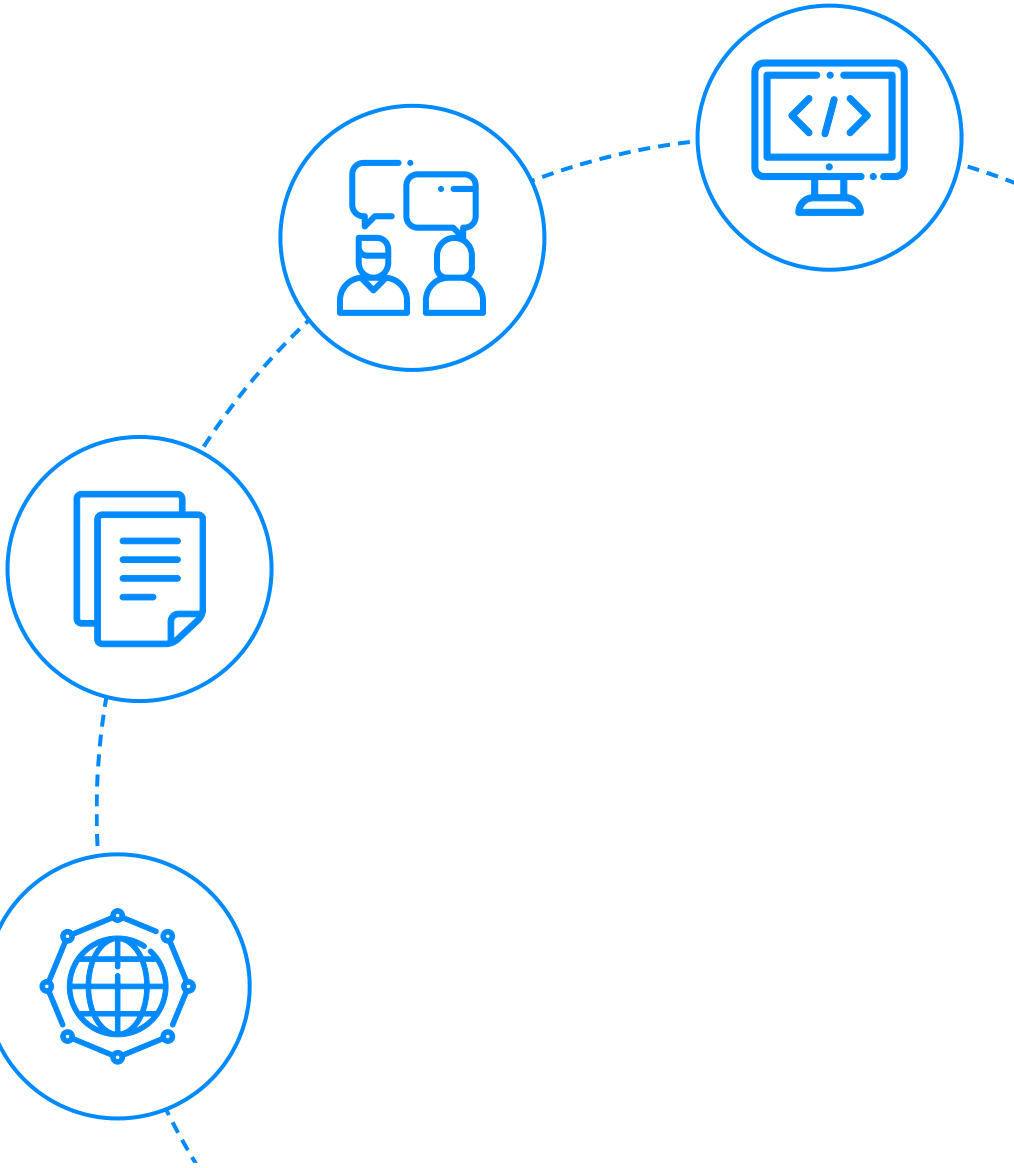
InterviewBit

Vim Cheat Sheet



To view the live version of the page, click here.

Contents

Vim Tutorial: Basics to Advanced Concepts

Let's get Started

What is Vim?

Vim (Vi IMproved) is an open-source text editor for Unix-based Operating Systems. It is a clone of the Vi editor and was created by Bram Moolenaar. It was first released in 1991. After the first release, cross-platform development enabled Vim to be available in many other Operating Systems. Vim traditionally doesn't have GUI and uses the terminal as its GUI but now there is a separate installed called gVim which provides GUI. Vim also has a built-in tutorial for beginners called **vimtutor** which is usually installed along with Vim although it exists as a separate executable.

Vim is a command-centric text editor and hence we can do everything we can do with modern text editors like VS Code, Sublime Text, etc with Vim just using the keyboard commands. Vim is a quite popular text editor because of its many features which save a lot of time for its users. The memory footprint is also very low. It also supports multiple tabs and windows which allows working on multiple files at the same time. Vim is highly configurable and extensible, making it an attractive tool for users who demand a large amount of control and flexibility over their text editing environment. Vim should be available by default on Linux based machines, if not we can download vim here.

Vim Tutorial: Basics to Advanced Concepts

1. Modes

Vim splits its functionality into different modes with each mode having a specific purpose. There are 7 different modes in Vim.

1. Normal Mode: By default, Vim starts in Normal mode. This mode can be thought of as an edit mode. Programmers spend most of their time editing than writing. Hence a text editor should be optimized for editing than writing. Hence this is the default mode. Mostly this mode is used for navigating and editing. To get to normal mode from any other mode, we press **Esc**.

2. Insert Mode: This mode is where the inserting of text happens. This mode can be thought of as a write mode. You can write content to the file using this mode. To enter into insert mode, we press i. When we enter this mode, we can see the status change at bottom of the screen as shown below.

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

3. Command Mode: This mode is used for doing complex operations on a file like a search, replace, etc. To enter into command mode from normal mode, we press : .

4. Visual Mode: This mode is used to make selections of text, similar to how clicking and dragging with a mouse behaves. Any commands used in visual mode apply to only selected text.
There are 3 different variants in Visual Mode.

- **Visual Mode:** To enter this mode, we press v. In this mode, any text selection happens from the start cursor to the end cursor. When we enter this mode, we can see the status change at bottom of the screen as shown below.

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

- **Visual Line Mode:** To enter this mode, we press Shift + v. In this mode, any text selection happens from starting line to the end line. When we enter this mode, we can see the status change at bottom of the screen as shown below.

```
~
~
--VISUAL LINE--
```

- **Visual Block Mode:** To enter this mode, we press Ctrl + v. In this mode, any text selection happens from the start cursor to the end cursor in rectangular blocks. When we enter this mode, we can see the status change at bottom of the screen as shown below.

```
~
~
--VISUAL BLOCK--
```

5. Select Mode: This is very similar to Visual Mode but it more looks like the MS-Windows selection mode. To enter this mode, we press **gh**. This also has 3 different sub-modes like select line mode, select block mode and select mode. When we enter this mode, we can see the status change at bottom of the screen as shown below.

```
~
~
--SELECT--
```

6. Ex Mode: This mode is more like a command mode, but after entering a command you remain in Ex mode. Very limited editing of the command line.

7. Terminal Job Mode: Interacting with a job in a terminal window. Typed keys go to the job and the job output is displayed in the terminal window.

Out of these 7 modes, only 4 modes are used mostly. Normal mode, Insert mode, Visual mode and Command mode.

From Mode	To Mode	Command	Explanation
Any Mode	Normal Mode	Esc	Move from any mode to normal mode.
Normal Mode	Insert Mode	i	Place the cursor at the current position.
Normal Mode	Visual Mode	v	Place the cursor at the current position.
		Shift + v	Move from normal mode to visual line mode.
		Ctrl + v	Move from normal mode to visual block mode.
Normal Mode	Command Mode	:	Move from normal mode to command mode.

2. Open, Save, Exit Files

To work on a file, we should know how to open a file, save the changes to the file and how to exit from vim.

Command	Explanation
vim <file name>	Opens the file in the vim editor.
:e <file name>	Open a file when the vim editor is already open.
:w	Save content to the current file.
:w <file name>	Save content by creating a new file <file name> if the file is not present. If present fails to write.
:q	Exit current vim editor if no changes are present.
:q!	Exit current vim editor ignoring changes.
:wq	Save vim content and then exit vim editor. :wq always writes the buffer to the file and updates the file modification time.
:x	Same as :wq but :x only writes the buffer to the file only if there are unsaved changes.

3. Editing

We do editing in normal mode and most of the time editing needs adding/replacing/deleting characters/words/lines, we need to be in insert mode for this. So most of the below command also implicitly move from normal mode to insert mode.

> Note: The yellow **highlighter** is the cursor. Text in **italic** implies, vim in insert mode else vim in normal mode. The statement before => is statement before applying command and statement after => is statement after applying command.

Below is the Vim Editing Cheat Sheet:

Command	Explanation	Example
a	Move the cursor to the next character.	The quick brown fox jumps over the lazy dog => *The quick brown fox jumps over the lazy dog*
A	Move the cursor to the end of the line.	The quick brown fox jumps over the lazy dog => *The quick brown fox jumps over the lazy dog*
o	Inserts a new line below the current line and place the cursor at the start of a new line.	The quick brown fox jumps over the lazy dog => *The quick brown fox jumps over the lazy dog*
O	Inserts a new line above the current line and place the cursor at the start of a new line.	The quick brown fox jumps over the lazy dog => *The quick brown fox jumps over the lazy dog*

4. Navigating

Navigating through a file can be considered one of the most important features a text editor can provide. Vim provides a lot of commands to not only navigate the file but to do the navigation efficiently. While navigating, the commands do not make vim enter insert mode, unlike editing.

Below is the vim navigation cheat sheet:

Command	Explanation	Example
h (or) left arrow	Move cursor left.	The quick brown fox jumps over the lazy dog. => The quick brown fox jumps over the lazy dog.
l (or) right arrow	Move cursor right.	The quick brown fox jumps over the lazy dog. => The quick brown fox jumps over the lazy dog.
j (or) down arrow	Move cursor down.	The quick brown fox jumps over the lazy dog. This statement is a pangram. => The quick brown fox jumps over the lazy dog. This statement is a pangram.

5. Search and Replace

Vim provides commands for searching content in a file and also can help in replacing content. Here are some of the most used commands for search and replace in vim. vim used sed Unix command like syntax to replace content in a file.

Command	Explanation
/pattern	Search for the pattern in the file from the cursor position to the end of the file (forward search).
?pattern	Search for the pattern in the file from the cursor position to the start of the file (backward search).
n	If multiple patterns are matched, n is used to move the cursor to the next pattern match.
N	If multiple patterns are matched, N is used to move the cursor to the previous pattern match.
:%s/old/new/g	Replaces all the *old* content with *new* content throughout the file.
:%s/old/new/gc	Replaces all the *old* content with *new* content throughout the file but with confirmation for each replacement.
*****	Search forwards for a word the same as a current word under the cursor.
#	Search backwards for a word the same as a current word under the cursor.

6. Cut, Copy, Paste

Cut, Copy, and Paste can be considered one of the most commonly performed tasks when working with text files. Keep this in mind, Vim provides several vim commands to do these tasks efficiently.

In the Vim world, Copy is called Yank, Cut is called Cut, and Paste is called Put. All the operations that we do have the effect only in Vim. For example, copying a line in vim copies the content to its internal buffer and hence can only be pasted in vim editor only. We can't paste the content somewhere else.

Operation	Command	Explanation	Example
Yank (Copy)	y	Copy a single character.	The quick brown fox jumps over the lazy dog. (copies the character w to vim buffer).
	yy	Copy the current line.	The quick brown fox jumps over the lazy dog. (copies the whole line to vim buffer).
	3yy	Copy three lines starting from the current line.	The quick brown fox jumps over the lazy dog. (copies lines The quick, brown, and fox jumps)

It would be much easier to cut, and copy in visual mode than in the normal mode. We can select the exact text we want to copy/cut in visual mode and use y (for copy), d (for cut).
To know more about copy, cut, and paste, visit **here**.

7. Working with Multiple Files

Vim also provides functionality to work with more than 1 file at the same time. To do this, Vim has 3 concepts.

1. Buffer
2. Window
3. Tab

A buffer is the in-memory text of a file. Vim creates a new buffer when we open a new file. Windows and Tabs are introduced in the subsequent section.

Command	Explanation	Example
vim <file1> <file2> <file3>	Opens vim editing session with 3 files with an initial focus on file1. Each opened file is considered a buffer.	vim test1.txt test2.txt test3.txt
:ls (or) :buffers	Shows a list of opened files.	1. test1.txt 2. test2.txt 3. test3.txt
:n (or) :bn	Switch to the next file. > Note: :bn means **n**ext **b**uffer	If we are at test1.txt currently, then :n moves focus to test2.txt.
:N (or) :bp	Switch to the previous file. > Note: :bp means the **p**revious **b**uffer	If we are at test2.txt currently, then :N moves focus to test1.txt.

8. Windows

The other feature that Vim provides for working with multiple files in Windows. A window is a viewport on a buffer. We can have multiple windows opened each for a different buffer. Windows make it possible to see more than 1 file on the same screen.

Command	Explanation	Example
vim -o <file 1> <file 2>	Opens 2 new buffers in 2 horizontally separated windows.	vim -o test1.txt test2.txt test1.txt is in the top window and test2.txt is in the bottom window.
vim -O <file 1> <file 2>	Opens 2 new buffers in 2 vertically separated windows.	vim -O test1.txt test2.txt test1.txt is in the left window and test2.txt is in the right window.
:sp <file>	If <file> is specified, opens the <file> is new horizontal window. If not, open the current buffer in a new horizontal window.	:sp test3.txt Opens another horizontal window for test3.txt buffer.

9. Tabs

The last feature that is provided by vim for making working with multiple files effectively is tabs.

A tab page is a page with one or more windows in it with a label at the top.

Command	Explanation	Example
vim -p <file 1> <file 2>	Opens 2 new buffers in 2 tabs.	vim -p test1.txt test2.txt opens 2 buffers such that test1.txt is in the first tab and test2.txt in the second tab.
vim -O <file 1> <file 2>	Open specified file in a new tab.	:tabe test3.txt opens a new tab for a new file test3.txt.
:sp <file>	Lists all the tabs opened in the current editing session.	:tabs gives 3 tabs as we opened test1.txt, test2.txt and test3.txt.
:vsp <file>	Closes the current tab.	If we are currently in tab that has file test3.txt, :tabc closes this tab.
Ctrl + w , j	Close the specified tab number.	:tabc 1 closes the first tab.
Ctrl + w, k	Go to next tab.	NA
Ctrl + w, l	Go to previous tab.	NA
Ctrl + w, h	Go to the tab at position	2gt moves to the 2nd tab

10. Miscellaneous

Vim also offers a lot of other features that can also be used for effectively doing various miscellaneous stuff.

Command	Explanation	Example
:! <cmd>	Execute <cmd> in the shell and press Enter to get back to where you are in vim.	:! ls will execute the command ls and show the list of files and then when we press enter, we get back to where we are in vim. This greatly helps in quickly getting back out of vim and executing some commands.
Ctrl + n	Autocomplete the word matching the current prefix.	If a file has the content "A quick brown fox jumps over the lazy dog" and we try to write the word brown again, instead of typing the whole brown, we can type b and then use Ctrl + N to auto-complete the word. If there is more than one-word matching prefix, then vim shows the list of possibilities to choose from.
Ctrl + x + l	Autocomplete the whole line matching the	Same as above but auto-complete the current prefix with

Conclusion

With all the features that Vim has, It can easily help its users to do work efficiently and quickly without having to remove their hands from the keyboard. Vim uses different modes to do different operations. We can quickly open files, save changes, navigate around the file, do the editing, do basic operations like a copy (yank), and cut and paste (put) easily with all the vim commands that vim provides. With the introduction of features like buffers, windows, and tabs, vim makes the users work on multiple files at the same time. Although it takes a bit of time to learn vim and get used to vim editor, once we get used to it, we can before very efficient and quick in writing code/ content.

Often in a developer's life, we get the need to ssh to other machines to do some tasks. With ssh, we will not have the ability to use modern text editors to edit/insert the code. Mastering Vim becomes very handy in that case to not only make the changes but do it quickly and efficiently with the help of all these vim shortcuts.

Useful Resources

- Technical Interview Questions
- Coding Interview Questions
- InterviewBit Blog
- Mock Interview

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions