

Text Editing using

vi Editor

LinuXpert Systems

No.5, Dr. Ambedkar St, Rathinapuram,

Nanganallur, Chennai – 600 061.

Phone: 98841 65649, 90808 01382

E-mail: baskar@linuxpert.in

TABLE OF CONTENTS

1. Introduction to the vi Editor 1
2. History of vi 2
3. Why Learn vi ? 2
4. Starting vi 3
5. Understanding vi Modes 4
6. Command Mode 4
7. Insert Mode 5
8. Last-Line Mode 5
9. Switching Between Modes 6
10. Creating and Opening Files 6
11. Saving and Exiting 7
12. Moving Around a File 8
13. Character-Level Movement 8
14. Word-Level Movement 9
15. Line-Level Movement 9
16. Screen-Level Movement 10
17. Searching in vi 11
18. Advanced Search 11
19. Searching for Whole Words 12
20. Replacing Text 12
21. Replace Throughout the File 13
22. Interactive Replacement 13
23. Inserting Text 13
24. Deleting Text 14
25. Advanced Deletion 15
26. Copying Text 16
27.Pasting Text 16
28. Undo and Redo 17

29. Repeating Commands 17
30. Number Prefixes 18
31. Operators and Motions 18
32. Change Operations 19
33. Text Objects 20
34. Quotes and Parentheses 20
35. Marks 21
36. Jumping Between Locations 22
37. Named Registers 22
38. Advanced Copy and Paste 23
39. Multiple-Line Operations 23
40. Line Numbers 24
41. External Linux Commands 24
42. Running Shell Commands 25
43. Reading External Files 25
44. Filtering Text 26
45. Advanced Substitution 26
46. Global Commands 27
47. Global Search and Replace 27
48. Regular Expressions 28
49. Character Classes 28
50. Advanced Regular Expressions 29
51. Pattern-Based Editing 29
52. Macros 30
53. Recording a Macro 30
54. Repeating Macros 31
55. Indentation 31
56. Joining Lines 32
57. Splitting Lines 32
58. Case Conversion 33
59. Working with Tabs and Spaces 33
60. Useful Vim Settings 34

61. `.vimrc` 35
62. vi for Linux Administration 35
63. Editing Configuration Files 35
64. Editing Shell Scripts 36
65. Editing Log Files 37
66. Working with Large Files 37
67. Editing System Files Safely 38
68. vi for Programmers 38
69. Common Mistakes 39
70. Practical Training Exercises 40
71. Advanced vi Command Reference 43
72. Important Advanced Concepts 46
73. Example: Real Linux Configuration Task 48
74. Example: Bulk Editing 49
75. Example: Commenting Lines 50
76. Example: Removing Blank Lines 50
77. Example: Delete Lines Containing a Word 51
78. Example: Extract Matching Lines 51
79. vi and Linux Philosophy 51
80. vi vs Graphical Editors 52
81. vi vs nano 52
82. vi / vim Learning Strategy 53
83. Important Interview Questions 55
84. Final Practical Assignment 57
85. Final Quick Reference 58

1. Introduction to the vi Editor

The **vi editor** is one of the most important text editors available on Linux and Unix systems.

The name `vi` comes from **visual editor**.

Unlike graphical editors, vi is primarily operated through the keyboard. This makes it extremely useful when working with:

- Linux servers
- Remote systems
- SSH sessions
- Terminal-only environments
- System configuration files
- Shell scripts
- Programming source code
- Log files
- Administrative files

For Linux administrators, knowing vi is particularly important because a graphical desktop may not be available on a server.

For example, suppose you connect to a remote Linux server using:

```
ssh user@server
```

You may need to edit:

```
/etc/hosts
/etc/fstab
/etc/ssh/sshd_config
/etc/crontab
/etc/resolv.conf
```

In such situations, vi can be used directly from the terminal.

2. History of vi

The original vi editor was developed by **Bill Joy** in the late 1970s as part of the Unix operating system environment.

vi evolved from an earlier editor called **ed**.

The basic development path was approximately:

```
ed
 ↓
ex
 ↓
vi
 ↓
vim and other vi-compatible editors
```

Today, many Linux systems provide:

```
vi
```

and often:

```
vim
```

where Vim means:

Vi Improved

Vim provides many additional capabilities while maintaining compatibility with traditional vi commands.

---

3. Why Learn vi?

There are several reasons Linux administrators and programmers should learn vi.

3.1 Available on servers

Minimal Linux installations frequently provide vi or a vi-compatible editor.

3.2 Keyboard driven

You don't need a mouse.

3.3 Very powerful

A small number of commands can perform complex editing operations.

3.4 Excellent for large files

vi can navigate through large text files efficiently.

3.5 Useful over SSH

It works through a terminal connection.

3.6 Powerful search and replacement

Regular expressions allow sophisticated text manipulation.

3.7 Automation

Macros and command combinations can automate repetitive editing.

---

4. Starting vi

The basic syntax is:

```
vi filename
```

Example:

```
vi test.txt
```

If the file exists, vi opens it.

If it doesn't exist, vi creates a new editing buffer and the file is created when you save it.

You can also open multiple files with Vim:

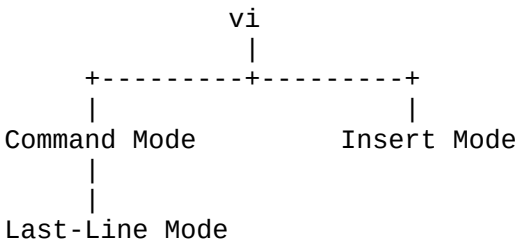
```
vim file1.txt file2.txt
```

5. Understanding vi Modes

The most important concept in vi is **modes**.

Unlike editors such as Notepad, vi does not always interpret keyboard input as text.

The major modes are:



The three important modes are:

1. Command mode
2. Insert mode
3. Last-line mode

---

6. Command Mode

When vi starts, it normally begins in **command mode**.

In command mode, pressing a key performs an editing or navigation operation.

For example:

```
h
j
k
l
```

move the cursor.

Other commands include:

```
x       delete character
dd      delete line
yy      copy line
p       paste
u       undo
```

If you accidentally type a letter and it doesn't appear in the file, you are probably in command mode.

7. Insert Mode

Insert mode is used to enter text.

Common commands that enter insert mode include:

```
i
a
o
O
```

For example:

```
i
```

means:

insert before the cursor

After pressing `i`, type your text.

To return to command mode:

```
Esc
```

This is one of the most important keys in vi.

---

8. Last-Line Mode

Last-line mode is entered using:

```
:
```

The command appears at the bottom of the terminal.

Examples:

```
:w
:q
:wq
:q!
```

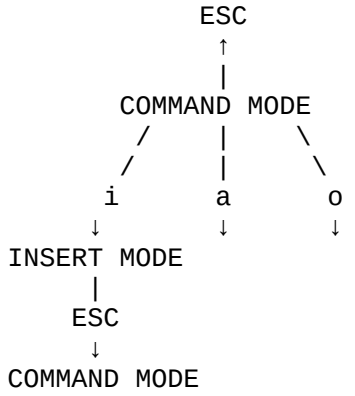
Other commands include:

```
:set number
:set nonumber
:%s/old/new/g
```

Last-line commands are sometimes called **ex commands**.

9. Switching Between Modes

A useful mental model is:



To insert text:

```
i
```

To return:

```
Esc
```

To save:

```
:w
```

To quit:

```
:q
```

---

10. Creating and Opening Files

Create a file:

```
vi example.txt
```

Type:

```
Linux is an operating system.
Linux supports multitasking.
Linux supports networking.
```

Save:

```
:w
```

Quit:

```
:q
```

You can combine both:

```
:wq
```

---

11. Saving and Exiting

Important commands:

Command	Meaning
`:w`	Save
`:q`	Quit
`:wq`	Save and quit
`:x`	Save and quit
`:q!`	Quit without saving
`ZZ`	Save and quit
`ZQ`	Quit without saving

Example:

```
:wq
```

is one of the most commonly used commands.

12. Moving Around a File

The cursor can be moved using:

```
h    left
j    down
k    up
l    right
```

Visual representation:

```
       k
       ↑
h ← cursor → l
       ↓
       j
```

This is extremely important because experienced vi users often don't need arrow keys.

---

13. Character-Level Movement

Useful commands:

```
h
l
```

Move one character.

You can also use:

```
0
```

Move to beginning of line.

```
$
```

Move to end of line.

Example:

```
Linux administration
^                   $
```

`0` moves to `L`.

`$` moves to the final character.

14. Word-Level Movement

Important commands:

`w`

Move forward to the next word.

`b`

Move backward to the previous word.

`e`

Move to the end of the current/next word.

Example:

```
Linux system administration
```

If the cursor is at `Linux`:

`w`

moves toward:

`system`

Another:

`b`

moves backward.

---

15. Line-Level Movement

Useful commands:

`0`

Beginning of line.

`^`

First non-whitespace character.

`$`

End of line.

`gg`

Beginning of file.

`G`

End of file.

For example:

`gg`

is useful when you want to quickly return to the first line.

---

16. Screen-Level Movement

Useful commands include:

`Ctrl+f`

Page forward.

`Ctrl+b`

Page backward.

`Ctrl+d`

Half-page down.

`Ctrl+u`

Half-page up.

`Ctrl+e`

Scroll down one line.

`Ctrl+y`

Scroll up one line.

These commands are particularly useful for large files.

17. Searching in vi

Search forward:

```
/pattern
```

Example:

```
/Linux
```

Press:

```
Enter
```

The cursor moves to the next occurrence.

Search backward:

```
?Linux
```

After searching:

```
n
```

moves to the next matching result.

```
N
```

moves in the opposite direction.

---

18. Advanced Search

Suppose a file contains:

```
Linux
Linux Administration
Linux Server
Linux Networking
Linux Security
```

Search:

```
/Linux
```

Then:

```
n
```

moves through each occurrence.

Case-sensitive searching depends on configuration.

In Vim, you can use:

```
:set ignorecase
```

to ignore case.

To make searches case-sensitive again:

```
:set noignorecase
```

---

19. Searching for Whole Words

Suppose the file contains:

```
server
servers
server-side
```

Searching:

```
/server
```

may find all occurrences.

To search for a whole word:

```
/\<server\>
```

This is useful when you don't want `servers` to match.

---

20. Replacing Text

The substitution command has the general form:

```
:s/old/new/
```

Example:

```
:s/Linux/UNIX/
```

This replaces the first matching occurrence on the current line.

To replace every occurrence on the current line:

```
:s/Linux/UNIX/g
```

21. Replace Throughout the File

The % symbol means the entire file.

Example:

```
:%s/Linux/UNIX/g
```

Meaning:

```
%       entire file
s       substitute
Linux   old text
UNIX    new text
g       all matches on each line
```

This is one of the most powerful basic vi commands.

---

22. Interactive Replacement

Sometimes you don't want to replace every match automatically.

Use:

```
:%s/Linux/UNIX/gc
```

The c means **confirm**.

vi asks for confirmation for each match.

Typical choices include:

```
y   yes
n   no
a   all
q   quit
```

This is safer when editing important configuration files.

23. Inserting Text

The major insertion commands are:

Command	Function
i	Insert before cursor
I	Insert at beginning of line
a	Append after cursor
A	Append at end of line
o	Open new line below
O	Open new line above

Example:

```
Linux
```

Move to the beginning and press:

```
I
```

Then type:

```
Operating System:
```

Result:

```
Operating System: Linux
```

---

24. Deleting Text

Basic deletion:

```
x
```

Delete character under cursor.

```
X
```

Delete character before cursor.

Delete a complete line:

```
dd
```

Delete multiple lines:

```
5dd
```

This means:

```
5 + dd
```

Delete five lines.

---

25. Advanced Deletion

vi uses an important concept:

operator + motion

For example:

```
dw
```

means:

```
d = delete
w = word movement
```

Therefore:

```
dw
```

deletes toward the next word.

Other examples:

```
d$
```

Delete from cursor to end of line.

```
d0
```

Delete from cursor to beginning.

```
dG
```

Delete from cursor to end of file.

26. Copying Text

Copying in vi is called **yanking**.

The basic command:

```
yy
```

copies the current line.

Multiple lines:

```
5yy
```

copies five lines.

The command:

```
y
```

is the yank operator.

---

27. Pasting Text

After copying:

```
p
```

pastes after the cursor.

```
P
```

pastes before the cursor.

Example:

```
yy
p
```

copies the current line and pastes it below.

28. Undo and Redo

Undo:

```
u
```

This reverses the most recent change.

In Vim, redo is commonly:

```
Ctrl+r
```

Example:

```
dd
u
```

The deleted line comes back.

Then:

```
Ctrl+r
```

reapplies the deletion.

---

29. Repeating Commands

The dot command:

```
.
```

repeats the previous change.

Example:

```
dw
.
.
```

If the first `dw` deletes one word, each `.` repeats the operation.

This is extremely useful for repetitive editing.

30. Number Prefixes

Many vi commands accept numbers.

For example:

```
5dd
```

Delete five lines.

```
10j
```

Move down ten lines.

```
3dw
```

Delete three words.

```
4yy
```

Copy four lines.

This makes vi very efficient.

---

31. Operators and Motions

A very important advanced vi concept is:

```
operator + motion
```

Common operators:

```
d   delete
c   change
y   yank
```

Common motions:

```
w
b
e
$
0
G
```

Examples:

```
dw
d$
dG
cw
c$
yw
y$
```

Once you understand this pattern, learning advanced vi becomes much easier.

---

32. Change Operations

The `c` operator means **change**.

Examples:

```
cw
```

Change word.

```
cc
```

Change entire line.

```
c$
```

Change from cursor to end of line.

After changing, vi enters insert mode.

Press:

```
Esc
```

when finished.

33. Text Objects

Vim provides powerful text-object operations.

For example:

```
ciw
```

means:

```
c = change
i = inside
w = word
```

So:

```
ciw
```

changes the entire word under the cursor.

Similarly:

```
diw
```

deletes the word.

```
yiw
```

copies the word.

---

34. Quotes and Parentheses

Text objects are particularly useful with programming.

Examples:

```
ci"
```

Change text inside double quotes.

```
di"
```

Delete text inside double quotes.

```
ci(
```

Change text inside parentheses.

```
di(
```

Delete text inside parentheses.

For example:

```
printf("Hello Linux");
```

Place the cursor inside `"Hello Linux"` and use:

```
ci"
```

Then type new text.

---

35. Marks

Marks allow you to remember specific locations.

Set a mark:

```
ma
```

Here `a` is the mark name.

Return to the mark:

```
'a
```

For example:

```
ma
```

marks the current line.

Later:

```
'a
```

returns to that line.

Marks are extremely useful when editing large files.

36. Jumping Between Locations

Useful navigation:

```
gg
```

Beginning of file.

```
G
```

End of file.

```
:100
```

Go to line 100.

```
100G
```

Go to line 100.

```
:$
```

Go to last line.

---

37. Named Registers

Registers provide storage for copied or deleted text.

The unnamed register is used automatically.

Named registers use:

```
"a
```

through:

```
"z
```

For example:

```
"ayy
```

copies the current line into register a.

Paste it:

```
"ap
```

This allows multiple pieces of text to be stored separately.

---

38. Advanced Copy and Paste

Suppose you want to copy a configuration block.

You can:

```
"ayy
```

Then move elsewhere.

Paste:

```
"ap
```

Another register:

```
"byy
```

Paste:

```
"bp
```

This is useful when working with several separate pieces of text.

---

39. Multiple-Line Operations

You can perform commands on line ranges.

For example:

```
:10,20d
```

deletes lines 10 through 20.

Copy lines:

```
:10,20y
```

Replace lines:

```
:10,20s/old/new/g
```

This allows precise editing.

---

40. Line Numbers

Display line numbers:

```
:set number
```

or:

```
:set nu
```

Hide them:

```
:set nonumber
```

or:

```
:set nonu
```

Line numbers are useful when:

- Debugging scripts
- Reading error messages
- Editing configuration files
- Discussing specific lines
- Navigating large files

---

41. External Linux Commands

One of vi's powerful features is its ability to execute Linux commands.

Use:

```
:!command
```

Example:

```
:!ls
```

This executes:

```
ls
```

without leaving vi.

Another:

```
:!pwd
```

displays the current directory.

---

42. Running Shell Commands

Example:

```
:!date
```

produces the current date and time.

You can run:

```
:!who
```

or:

```
:!df -h
```

or:

```
:!free -h
```

This is particularly useful for Linux administrators.

---

43. Reading External Files

You can insert the contents of another file into the current file using:

```
:r filename
```

Example:

```
:r /etc/hosts
```

This reads `/etc/hosts` into the current editing buffer.

You can also read command output:

```
:r !date
```

The output of `date` is inserted into the file.

---

44. Filtering Text

The `!` operator can send selected text to an external Linux command.

For example:

```
!$
```

works with a motion and external command.

In Vim, you may see operations such as:

```
!}
```

followed by:

```
sort
```

This can sort a selected block.

For example:

```
!}
sort
```

can send the text through the Linux `sort` command.

This creates a powerful connection between vi and standard Linux utilities.

---

45. Advanced Substitution

Basic:

```
:%s/old/new/g
```

Case-insensitive:

```
:%s/old/new/gi
```

Confirm:

```
:%s/old/new/gc
```

Specific lines:

```
:10,50s/old/new/g
```

Entire file:

```
:%s/old/new/g
```

Only current line:

```
:s/old/new/g
```

---

46. Global Commands

The global command is:

```
:g
```

It allows operations on lines matching a pattern.

Example:

```
:g/Linux/p
```

prints all lines containing `Linux`.

Delete lines containing a pattern:

```
:g/Linux/d
```

This is a powerful administrative editing feature.

---

47. Global Search and Replace

Example:

```
:g/Linux/s/Linux/UNIX/g
```

This finds lines containing `Linux` and performs substitution on those lines.

This is useful when you don't want to modify unrelated lines.

48. Regular Expressions

Regular expressions make vi search extremely powerful.

Basic patterns include:

`^`

Beginning of line.

`$`

End of line.

`.`

Any character.

`*`

Zero or more occurrences of the preceding pattern.

Example:

```
/^Linux
```

finds lines beginning with `Linux`.

Search:

```
/Linux$
```

finds lines ending with `Linux`.

---

49. Character Classes

You can use:

`[0-9]`

for a digit.

`[a-z]`

for lowercase letters.

`[A-Z]`

for uppercase letters.

Example:

```
/[0-9][0-9][0-9]
```

searches for three digits.

---

50. Advanced Regular Expressions

Consider:

```
:%s/[0-9][0-9]*/NUMBER/g
```

This can replace sequences of digits.

Another example:

```
:%s/^/# /
```

adds # to the beginning of every line.

This can be useful for commenting configuration or script lines.

To remove a leading #:

```
:%s/^# //
```

---

51. Pattern-Based Editing

Suppose a file contains:

```
ERROR: Disk failure
INFO: Server started
ERROR: Network failure
INFO: User login
```

You can find errors:

```
:g/^ERROR/p
```

Delete all error lines:

```
:g/^ERROR/d
```

Replace text only in error lines:

```
:g/^ERROR/s/failure/problem/g
```

This demonstrates the power of pattern-based editing.

52. Macros

Macros are one of the most advanced vi/Vim features.

A macro records a sequence of commands.

Start recording into register `a`:

```
qa
```

Perform your editing commands.

Stop recording:

```
q
```

Replay:

```
@a
```

Repeat the macro:

```
@@
```

This is extremely useful for repetitive editing.

---

53. Recording a Macro

Suppose you have:

```
apple
banana
orange
mango
```

and want to add:

```
Fruit:
```

before each line.

You can record a macro that:

1. Goes to line beginning.
2. Inserts text.
3. Returns to command mode.
4. Moves down.

Then replay it.

The exact sequence depends on the desired formatting, but the concept is:

```
qa
<commands>
q
```

then:

```
@a
```

---

54. Repeating Macros

If the macro is stored in register `a`:

```
@a
```

runs it once.

Run it multiple times:

```
10@a
```

This executes the macro ten times.

This can save a huge amount of time.

---

55. Indentation

Indentation is important for programming.

In Vim, common commands include:

```
>>
```

indent current line.

```
<<
```

unindent current line.

Multiple lines can be indented using:

```
5>>
```

or through visual selection.

Automatic indentation can be configured with:

```
:set autoindent
```

---

56. Joining Lines

The command:

```
J
```

joins the current line with the next line.

Example:

```
Linux is an
operating system.
```

After:

```
J
```

becomes approximately:

```
Linux is an operating system.
```

This is useful for fixing wrapped text.

---

57. Splitting Lines

Place the cursor where you want the split and press:

```
i
```

Then press:

```
Enter
```

This creates a new line.

Alternatively, in Vim, advanced editing commands can be combined with motions and operators.

58. Case Conversion

In Vim, case conversion commands include:

~

Toggle case of a character.

For example:

```
linux
```

can become:

```
Linux
```

Repeated use of ~ toggles the character case.

Advanced operations include:

```
gU
```

uppercase operation.

```
gu
```

lowercase operation.

For example:

```
guw
```

converts a word to lowercase.

---

59. Working with Tabs and Spaces

Tabs can be configured using:

```
:set tabstop=4
```

This controls tab display width.

Use:

```
:set shiftwidth=4
```

for indentation width.

Use:

```
:set expandtab
```

to insert spaces instead of tab characters.

Disable:

```
:set noexpandtab
```

These settings are especially useful when programming.

---

60. Useful Vim Settings

Some useful settings:

```
:set number
:set autoindent
:set ruler
:set showmode
:set showcmd
```

Meaning:

Setting	Purpose
`number`	Display line numbers
`autoindent`	Automatic indentation
`ruler`	Display cursor position
`showmode`	Show current mode
`showcmd`	Show command being entered

61. `.vimrc`

Vim configuration is commonly stored in:

```
~/.vimrc
```

You can edit it:

```
vi ~/.vimrc
```

For example:

```
set number
set autoindent
set tabstop=4
set shiftwidth=4
set expandtab
```

After saving, Vim can use these settings.

---

62. vi for Linux Administration

vi is extremely important for system administration.

Examples:

```
sudo vi /etc/hosts
```

```
sudo vi /etc/fstab
```

```
sudo vi /etc/ssh/sshd_config
```

```
sudo vi /etc/crontab
```

Administrators often work with these files from a terminal.

---

63. Editing Configuration Files

Suppose `/etc/hosts` contains:

```
127.0.0.1 localhost
```

You can open it:

```
sudo vi /etc/hosts
```

Search:

```
/localhost
```

Edit:

```
ciw
```

and enter the new value.

Save:

```
:w
```

---

64. Editing Shell Scripts

vi is commonly used for shell scripts.

Example:

```
vi backup.sh
```

Enter:

```
#!/bin/bash

echo "Starting backup"
date
```

Save and exit.

Make executable:

```
chmod +x backup.sh
```

Run:

```
./backup.sh
```

65. Editing Log Files

vi can be useful for inspecting logs, although read-only tools such as `less` are usually better for very large logs.

Example:

```
vi /var/log/syslog
```

Search:

```
/ERROR
```

or:

```
/failed
```

Move between matches:

```
n
```

---

66. Working with Large Files

For large files, navigation commands become particularly important.

Useful commands:

```
gg
G
Ctrl+f
Ctrl+b
/pattern
n
N
:1000
```

You can jump directly to a line:

```
:50000
```

This is much faster than scrolling manually.

67. Editing System Files Safely

System configuration files should be edited carefully.

Before modifying an important file, create a backup.

For example:

```
sudo cp /etc/hosts /etc/hosts.backup
```

Then edit:

```
sudo vi /etc/hosts
```

If something goes wrong, the backup can be restored.

For important configuration files, this is a good administrative practice.

---

68. vi for Programmers

vi/Vim is widely used for:

- C
- C++
- Python
- Shell scripting
- Perl
- Java
- JavaScript
- HTML
- CSS
- YAML
- JSON
- SQL

Useful features include:

```
search
replace
indentation
macros
copy/paste
navigation
case conversion
text objects
```

---

69. Common Mistakes

Mistake 1: Typing text in command mode

Solution:

```
i
```

then type.

Mistake 2: Can't exit

Press:

```
Esc
```

then:

```
:q!
```

Mistake 3: Accidentally deleted something

Use:

```
u
```

Mistake 4: Don't know the current mode

Press:

```
Esc
```

Then start again.

Mistake 5: Accidentally changed many lines

Use:

```
u
```

or carefully use undo/redo in Vim.

---

70. Practical Training Exercises

Exercise 1 - Create a file

```
vi training.txt
```

Enter ten lines of text.

Save:

```
:wq
```

Exercise 2 - Navigation

Practice:

```
h
j
k
l
w
b
e
0
$
gg
G
```

Exercise 3 - Deletion

Practice:

```
x
dd
3dd
dw
d$
```

Exercise 4 - Copy and Paste

Practice:

```
yy
p
P
3yy
```

Exercise 5 - Search

Search for:

```
/Linux
```

Then practice:

```
n
N
```

Exercise 6 - Replace

Replace:

```
Linux
```

with:

```
UNIX
```

using:

```
:%s/Linux/UNIX/g
```

Exercise 7 - Confirm Replacement

Use:

```
:%s/Linux/UNIX/gc
```

Answer each prompt with:

```
y
n
a
q
```

Exercise 8 - Line Ranges

Delete lines 10 through 20:

```
:10,20d
```

Exercise 9 - Global Command

Delete lines containing `ERROR`:

```
:g/ERROR/d
```

Exercise 10 - Macro

Record a simple macro:

```
qa
```

Perform several editing operations.

Stop:

```
q
```

Replay:

```
@a
```

71. Advanced vi Command Reference

Navigation

Command	Function
h	Left
j	Down
k	Up
l	Right
w	Next word
b	Previous word
e	End of word
0	Beginning of line
^	First non-space
$	End of line
gg	Beginning of file
G	End of file
Ctrl+f	Page down
Ctrl+b	Page up
Ctrl+d	Half page down
Ctrl+u	Half page up

Insert Commands

Command	Function
i	Insert
I	Beginning of line
a	Append
A	End of line
o	New line below
O	New line above

Delete Commands

Command	Function
x	Character
X	Previous character
dd	Line
dw	Word
d$	To end
d0	To beginning
dG	To end of file

Copy Commands

Command	Function
yy	Copy line
yw	Copy word
y$	Copy to end
4yy	Copy four lines

Paste

Command	Function
p	Paste after
P	Paste before

Undo/Redo

`u`

Undo.

`Ctrl+r`

Redo in Vim.

`.`

Repeat last change.

Search

`/pattern`

Forward search.

`?pattern`

Backward search.

`n`

Next match.

`N`

Previous/opposite-direction match.

Substitution

`:s/old/new/`

Current line, first match.

`:s/old/new/g`

Current line, all matches.

`:%s/old/new/g`

Entire file.

```
:%s/old/new/gc
```

Entire file with confirmation.

File Commands

```
:w
```

Save.

```
:q
```

Quit.

```
:wq
```

Save and quit.

```
:q!
```

Quit without saving.

```
:x
```

Save and quit.

---

72. Important Advanced Concepts

For Linux training, the following concepts should be mastered:

Beginner

```
i
Esc
:w
:q
:wq
:q!
```

Navigation

```
h j k l
w b e
0 ^
$
gg
G
```

Editing

```
x
dd
dw
cw
yy
p
u
.
```

Search

```
/pattern
?pattern
n
N
```

Replacement

```
:s/old/new/
:%s/old/new/g
:%s/old/new/gc
```

Advanced

```
:g
:!
:r
registers
marks
macros
text objects
regular expressions
```

---

73. Example: Real Linux Configuration Task

Suppose you need to modify a configuration file:

```
sudo vi /etc/example.conf
```

First enable line numbers:

```
:set number
```

Search for:

```
/Port
```

Suppose you find:

```
Port 22
```

Move to `22`.

Use:

```
ciw
```

Type:

```
2222
```

Press:

```
Esc
```

Save:

```
:w
```

Then exit:

```
:q
```

This demonstrates several vi concepts:

```
search
text object
change
insert
command mode
save
exit
```

---

74. Example: Bulk Editing

Suppose you have:

```
user1
user2
user3
user4
user5
```

and want:

```
User: user1
User: user2
User: user3
User: user4
User: user5
```

A substitution can add the prefix:

```
:%s/^/User: /
```

The expression:

```
^
```

means beginning of each line.

Therefore:

```
:%s/^/User: /
```

adds `User:` to every line.

75. Example: Commenting Lines

Suppose:

```
server
port 8080
debug yes
```

You want:

```
# server
# port 8080
# debug yes
```

Use:

```
:%s/^/# /
```

This adds # at the beginning of every line.

To remove it:

```
:%s/^# //
```

This demonstrates why regular expressions are so useful.

---

76. Example: Removing Blank Lines

In Vim, a commonly used global command is:

```
:g/^$/d
```

Explanation:

```
^     beginning
$     end
```

Therefore:

```
^$
```

matches an empty line.

Then:

```
d
```

deletes it.

77. Example: Delete Lines Containing a Word

Suppose:

```
Linux server
Windows server
Linux desktop
Linux networking
Windows desktop
```

To delete lines containing `Windows`:

```
:g/Windows/d
```

Result:

```
Linux server
Linux desktop
Linux networking
```

---

78. Example: Extract Matching Lines

To display lines containing `ERROR`:

```
:g/ERROR/p
```

This is useful when analyzing configuration or text data.

For large logs, however, commands such as:

```
grep ERROR logfile
```

are often more appropriate.

The important concept is that vi can interact with the same Linux command-line tools.

---

79. vi and Linux Philosophy

One of the most important lessons of Linux is:

> Small tools can be combined to perform powerful tasks.

vi follows this philosophy.

You can combine:

```
vi
grep
sed
awk
sort
uniq
cut
head
tail
```

For example, you can search a file using vi and then use Linux commands to process external data.

This makes command-line editing extremely powerful.

---

80. vi vs Graphical Editors

Feature	vi	Graphical Editor
Terminal required	Yes	Usually GUI
Mouse	Not required	Usually
Remote SSH	Excellent	Difficult without GUI
Server use	Excellent	Limited
Keyboard efficiency	Very high	Moderate
Automation	Excellent	Variable
Learning curve	High	Low
Large-file navigation	Good	Depends
Linux administration	Excellent	Depends

---

81. vi vs nano

`nano` is generally easier for beginners.

Example:

```
nano file.txt
```

However, vi provides more powerful editing operations.

For example:

```
:%s/old/new/g
:g/pattern/d
qa
@a
ciw
```

These operations can make large editing tasks much faster.

A good Linux administrator should ideally know at least:

```
vi/vim
nano
```

---

82. vi / vim Learning Strategy

Don't try to memorize every command immediately.

Learn in stages.

Stage 1

Learn:

```
i
Esc
:w
:q
:wq
:q!
```

Stage 2

Learn:

```
h j k l
w b
0 $
gg G
```

Stage 3

Learn:

```
dd
yy
p
u
x
```

Stage 4

Learn:

```
/
n
N
:%s
```

Stage 5

Learn:

```
:g
:r
:!
marks
registers
macros
```

Stage 6

Learn:

```
regular expressions
text objects
advanced Vim commands
```

83. Important Interview Questions

Q1. What is vi?

vi is a command-line visual text editor commonly available on Unix/Linux systems.

Q2. What are the major modes?

The major modes are:

```
Command mode
Insert mode
Last-line/Ex mode
```

Q3. How do you enter insert mode?

Using commands such as:

```
i
a
o
```

Q4. How do you exit insert mode?

Press:

```
Esc
```

Q5. How do you save?

```
:w
```

Q6. How do you quit without saving?

```
:q!
```

Q7. How do you delete a line?

```
dd
```

Q8. How do you copy a line?

```
yy
```

Q9. How do you paste?

```
p
```

Q10. How do you search?

```
/pattern
```

Q11. How do you replace text throughout a file?

```
:%s/old/new/g
```

Q12. What does `gg` do?

Moves to the beginning of the file.

Q13. What does `G` do?

Moves to the end of the file.

Q14. What does `u` do?

Undo.

Q15. What does `.` do?

Repeats the previous change.

84. Final Practical Assignment

Create:

```
vi linux_training.txt
```

Enter at least 20 lines describing Linux.

Then perform all of the following:

1. Display line numbers.
2. Move to the beginning.
3. Move to the end.
4. Search for `Linux`.
5. Move through search results.
6. Copy five lines.
7. Paste them.
8. Delete three lines.
9. Undo the deletion.
10. Replace one word.
11. Replace that word throughout the file.
12. Confirm replacements.
13. Add text at the beginning of every line.
14. Delete all blank lines.
15. Use a macro.
16. Use a named register.
17. Set a mark.
18. Jump to the mark.
19. Execute a Linux command from vi.
20. Save and exit.

This exercise covers most of the skills expected from a Linux trainee.

85. Final Quick Reference

The most important commands to memorize are:

```
ESC                 Command mode
i                   Insert
a                   Append
o                   New line below
O                   New line above

h                   Left
j                   Down
k                   Up
l                   Right

w                   Next word
b                   Previous word
0                   Beginning of line
$                   End of line
gg                  Beginning of file
G                   End of file

x                   Delete character
dd                  Delete line
dw                  Delete word
d$                  Delete to end

yy                  Copy line
p                   Paste after
P                   Paste before

u                   Undo
Ctrl+r              Redo in Vim
.                   Repeat command

/pattern            Search forward
?pattern            Search backward
n                   Next match
N                   Previous match

:w                  Save
:q                  Quit
:wq                 Save and quit
:q!                 Quit without saving

:s/old/new/         Replace
:%s/old/new/g       Replace throughout file
:%s/old/new/gc      Replace with confirmation

:set number         Line numbers
:set nonumber       Hide line numbers

:g/pattern/p        Print matching lines
:g/pattern/d        Delete matching lines
```

```
:r file             Read another file
:!command           Execute Linux command

qa                  Record macro in register a
q                   Stop recording
@a                  Execute macro a
@@                  Repeat previous macro
```

Conclusion

The **vi editor is much more than a simple text editor**. It is a powerful command-driven editing environment that fits naturally into Linux administration and programming.

The most important idea is to understand the relationship between **commands, motions, operators, registers, search patterns, and external Linux commands**.

For example:

```
d + w
```

means:

```
delete + word movement
```

while:

```
c + i + w
```

means:

```
change + inside + word
```

and:

```
:%s/old/new/g
```

means:

```
entire file + substitute + old text + new text + global
```

Once these patterns become familiar, vi stops feeling like a collection of unrelated commands and becomes a **logical editing language**.

For Linux training, focus especially on:

```
Modes
Navigation
Insert/Delete
Yank/Paste
Search
Substitution
Regular Expressions
Line Ranges
Global Commands
Registers
Marks
Macros
External Commands
Text Objects
Vim Configuration
```

Mastering these features will allow you to edit **Linux configuration files, shell scripts, source code, logs, data files, and documentation efficiently—even on a remote terminal without a graphical interface.**