

File Viewing and Text Processing

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. cat 1

2. less 8

3. more 14

4. head 19

5. tail 24

6. nl 30

7. wc 36

8. cut 42

9. paste 50

10. sort 55

11. uniq 62

12. tr 68

13. fmt 74

1. cat

The `cat` (short for **concatenate**) command is a Linux/Unix command used to **display, create, combine, and copy the contents of text files**.

Syntax

```
cat [OPTIONS] [FILE...]
```

- `OPTIONS` – Optional flags that modify the command's behavior.
- `FILE` – One or more files to display or combine.

Common Uses

1. Display the contents of a file

```
cat file.txt
```

Output

```
Hello
Welcome to Linux
Have a nice day
```

Displays the entire contents of `file.txt`.

---

2. Display multiple files

```
cat file1.txt file2.txt
```

Example:

file1.txt

```
Apple
Banana
```

file2.txt

```
Orange
Mango
```

Output :

```
Apple
Banana
Orange
Mango
```

The files are displayed one after another.

---

3. Create a new file

```
cat > file.txt
```

Type your text:

```
Hello Linux
Learning cat command
```

Press

```
Ctrl + D
```

to save.

---

4. Append text to a file

```
cat >> file.txt
```

Type additional text:

```
This is another line.
```

Press **Ctrl + D**.

The new text is added to the end of the file.

5. Combine multiple files

```
cat file1.txt file2.txt > combined.txt
```

Example :

```
file1.txt


----------

Linux


file2.txt

----------
Commands
```

Result :

```
combined.txt
------------
Linux
Commands
```

---

6. Copy a file

```
cat source.txt > destination.txt
```

This copies the contents of `source.txt` into `destination.txt`.

Useful Options

Show line numbers

```
cat -n file.txt
```

Output

```
     1  Linux
     2  Ubuntu
     3  Fedora
```

Number only non-empty lines

```
cat -b file.txt
```

Example :

```
Linux

Ubuntu
```

Output:

```
     1 Linux

     2 Ubuntu
```

Blank lines are not numbered.

Show end of each line

```
cat -E file.txt
```

Output

```
Linux$
Ubuntu$
Fedora$
```

$ marks the end of each line.

Display tabs

```
cat -T file.txt
```

Output

```
Name^IAge
John^I20
```

^I represents a tab character.

Show non-printing characters

```
cat -v file.txt
```

Useful for viewing control characters.

Squeeze repeated blank lines

```
cat -s file.txt
```

Before

```
Linux

Ubuntu

Fedora
```

After

```
Linux

Ubuntu

Fedora
```

Multiple consecutive blank lines become a single blank line.

Show all special characters

```
cat -A file.txt
```

Equivalent to using `-vET`.

Output

```
Linux$
Ubuntu$
Name^IAge$
```

Practical Examples

View a configuration file

```
cat /etc/hostname
```

View the password file

```
cat /etc/passwd
```

Save command output to a file

```
cat /etc/passwd > users.txt
```

Merge log files

```
cat log1.txt log2.txt log3.txt > all_logs.txt
```

Exit Status

- `0` – Command executed successfully.
- Non-zero – An error occurred (such as a missing file or insufficient permissions).

Example:

```
cat missing.txt
```

Output

```
cat: missing.txt: No such file or directory
```

Advantages

- Simple and fast.
- Can display one or multiple files.
- Useful for creating and appending to text files.
- Can combine multiple files into one.
- Frequently used in shell scripting and pipelines.

Limitations

- Not ideal for very large files because it displays the entire file at once.
- For large files, use a pager such as:

```
less file.txt
```

or

```
more file.txt
```

Summary

Command	Description
`cat file.txt`	Display a file
`cat file1 file2`	Display multiple files
`cat > file.txt`	Create a new file
`cat >> file.txt`	Append to a file
`cat file1 file2 > file3`	Combine files
`cat -n file.txt`	Show line numbers
`cat -b file.txt`	Number non-empty lines
`cat -E file.txt`	Show end-of-line markers (`$`)
`cat -T file.txt`	Show tab characters (`^I`)
`cat -A file.txt`	Show all non-printing characters
`cat -s file.txt`	Remove repeated blank lines in output

`cat` is one of the most commonly used Linux commands for quickly viewing, creating, concatenating, and redirecting the contents of text files.

2. less

The `less` command is a **pager** used to view the contents of a text file one screen at a time. Unlike `cat`, it allows you to scroll forward and backward through large files without loading the entire file into memory.

Syntax

```
Less [options] filename
```

or

```
command | less
```

Purpose

- View large text files comfortably.
- Scroll through files page by page.
- Search for text within a file.
- Read command output without it scrolling off the screen.

Basic Examples

1. View a file

```
less file.txt
```

Displays `file.txt` one screen at a time.

---

2. View a log file

```
less /var/log/syslog
```

or

```
less /var/log/messages
```

Useful for examining large log files.

3. View command output

```
ps aux | less
```

Shows the process list page by page.

Another example:

```
dmesg | less
```

Displays kernel messages in a scrollable view.

---

4. View a compressed file

```
less file.txt.gz
```

On many Linux systems, `less` can automatically display compressed files.

Navigation Keys

Key	Function
Space	Next page
b	Previous page
Enter	Next line
Up Arrow	Previous line
Down Arrow	Next line
g	Go to beginning of file
G	Go to end of file
50g	Go to line 50
q	Quit
h	Help

Searching

Search forward

```
/error
```

Searches for the word **error**.

Press:

```
n to find the next occurrence.
```

Press:

```
N
```

to find the previous occurrence.

Search backward

```
?error
```

Searches upward for the word **error**.

Useful Options

Display line numbers

```
less -N file.txt
```

Example output:

```
1  Linux
2  Windows
3  macOS
```

Ignore case while searching

```
less -i file.txt
```

Searching for `linux` also finds `Linux`.

Force open non-regular files

```
less -f filename
```

Useful for special files.

Exit automatically if the content fits on one screen

```
less -F file.txt
```

Show control characters

```
less -R file.txt
```

Useful for viewing colored output.

Example:

```
grep --color=always "error" logfile | less -R
```

Common Real-World Examples

Read a configuration file

```
less /etc/passwd
```

Read a long log

```
less /var/log/auth.log
```

Read a README file

```
less README.md
```

Search for "network" in a log

```
less /var/log/syslog
```

Inside `less`:

```
/network
```

View Git differences

```
git diff | less
```

Difference Between `cat`, `more`, and `less`

Feature	`cat`	`more`	`less`
Displays entire file at once			
Scroll forward			
Scroll backward			
Search within file		Limited	
Suitable for large files		Good	Excellent

Advantages

- Fast and memory-efficient.
- Supports forward and backward navigation.
- Powerful search capabilities.
- Can view very large files.
- Frequently used for log analysis and reading configuration files.

Summary

- **Command:** `less`
- **Purpose:** View text files or command output interactively, one screen at a time.
- **Best used for:** Large files, log files, configuration files, and long command outputs.
- **Exit:** Press **`q`**.
- **Search:** Use `/pattern` (forward) or `?pattern` (backward).

3. more

The `more` command is a **text file viewer** in Linux/Unix. It displays the contents of a file **one screen (page) at a time**, making it useful for reading large files without flooding the terminal.

Syntax

```
more [options] filename
```

Or read output from another command:

```
command | more
```

Purpose

- View large text files page by page.
- Pause after each screen of output.
- Read command output that is too long to fit on one screen.

Basic Examples

1. View a text file

```
more file.txt
```

Displays the contents of `file.txt` one screen at a time.

---

2. View a log file

```
more /var/log/syslog
```

Shows the system log page by page.

3. View command output

```
ls -l /usr/bin | more
```

Displays the output of `ls -l` one page at a time.

---

4. View multiple files

```
more file1.txt file2.txt
```

After reaching the end of `file1.txt`, `more` continues with `file2.txt`.

Navigation Keys

Key	Action
Space	Next page
Enter	Next line
b	Previous page (works only when reading from a file, not from a pipe, on many implementations)
/	Search forward for text
n	Repeat last search
q	Quit
h	Display help

Common Options

`-d`

Displays helpful prompts.

```
more -d file.txt
```

Example prompt:

```
Press space to continue, 'q' to quit.
```

`-c`

Clears the screen before displaying the next page.

```
more -c file.txt
```

`-s`

Compresses multiple blank lines into a single blank line.

```
more -s file.txt
```

`+number`

Starts displaying from a specific line.

```
more +100 file.txt
```

Starts at line 100.

`+/pattern`

Starts at the first occurrence of a pattern.

```
more +/error logfile.txt
```

Begins displaying at the first line containing "error".

Practical Examples

Read a configuration file

```
more /etc/passwd
```

Read kernel messages

```
dmesg | more
```

Read a long text file

```
more README.txt
```

Start at line 500

```
more +500 largefile.txt
```

Sample Session

Suppose `sample.txt` contains 100 lines.

```
$ more sample.txt
```

Output:

```
Line 1
Line 2
...
Line 24
--More--
```

At the `--More--` prompt:

- Press **Space** → next page
- Press **Enter** → next line
- Press **q** → quit

Exit

Press:

`q`

to quit `more`.

Difference Between `more` and `less`

Feature	`more`	`less`
Forward scrolling	Yes	Yes
Backward scrolling	Limited	Full support
Search	Basic	Advanced
Performance	Good	Better for very large files
Navigation	Basic	Rich navigation features
Recommended	Simple viewing	Everyday use

`less` is generally preferred today because it offers more powerful navigation and search capabilities.

Summary

Feature	Description
Command	`more`
Purpose	View text files one page at a time
Input	Files or piped command output
Next page	`Space`
Next line	`Enter`
Search	`/pattern`
Quit	`q`
Best for	Reading large files or lengthy command output

4. head

The `head` command is used to **display the beginning (top) part of a file**. By default, it prints the **first 10 lines** of a file.

Syntax

```
head [OPTION]... [FILE]...
```

Basic Example

```
head file.txt
```

Output:

Displays the first 10 lines of `file.txt`.

Common Options

Option	Description
`-n NUMBER`	Display the first specified number of lines.
`-c NUMBER`	Display the first specified number of bytes.
`-q`	Never print file headers when multiple files are specified.
`-v`	Always print file headers.
`--help`	Display help information.
`--version`	Display version information.

Examples

1. Display the first 10 lines (default)

```
head file.txt
```

---

2. Display the first 5 lines

```
head -n 5 file.txt
```

or

```
head -5 file.txt
```

---

3. Display the first 20 lines

```
head -n 20 file.txt
```

---

4. Display the first 100 bytes

```
head -c 100 file.txt
```

---

5. Display the first 1 KB (1024 bytes)

```
head -c 1K file.txt
```

Supported suffixes include:

- `K` = Kilobytes
- `M` = Megabytes
- `G` = Gigabytes

Example:

```
head -c 2M largefile.bin
```

6. Display the first lines of multiple files

```
head file1.txt file2.txt
```

Output:

```
==> file1.txt <==
...

==> file2.txt <==
...
```

---

7. Suppress file headers

```
head -q file1.txt file2.txt
```

---

8. Always display file headers

```
head -v file1.txt
```

---

9. Read from standard input

```
echo -e "One\nTwo\nThree\nFour" | head -n 2
```

Output:

```
One
Two
```

---

10. View the beginning of a log file

```
head /var/log/syslog
```

or

```
head /var/log/messages
```

Practical Uses

Preview a CSV file

```
head employees.csv
```

Verify the contents of a configuration file

```
head /etc/passwd
```

Check the beginning of a script

```
head backup.sh
```

Combine with another command

```
ls -l | head
```

Shows the first 10 entries from the `ls -l` output.

Difference Between `head` and `tail`

`head`	`tail`
Displays the beginning of a file	Displays the end of a file
Default: first 10 lines	Default: last 10 lines
Useful for previewing files	Useful for monitoring logs

Exit Status

- `0` – Command executed successfully.
- Non-zero – An error occurred (such as the file not existing or insufficient permissions).

Summary

- **Purpose:** Display the beginning of a file.
- **Default output:** First **10 lines**.
- **Useful options:**
  - `-n` → Show a specific number of lines.
  - `-c` → Show a specific number of bytes.
  - `-q` → Hide file headers.
  - `-v` → Always show file headers.
- Can read from **files** or **standard input** using a pipe (`|`).

The `head` command is commonly used to quickly inspect text files, log files, CSV files, configuration files, and the output of other commands.

5. tail

The `tail` command is used to **display the last part (end) of a file**. By default, it shows the **last 10 lines** of a text file. It is commonly used to view log files and monitor files that are continuously updated.

Syntax

```
tail [OPTIONS] FILE
```

Basic Example

```
tail file.txt
```

Output:

Displays the last 10 lines of `file.txt`.

Common Options

1. Display the last *N* lines (`-n`)

Shows the specified number of lines from the end.

Syntax

```
tail -n NUMBER FILE
```

Example

```
tail -n 5 file.txt
```

Output

Displays the last 5 lines.

2. Display the last *N* bytes (`-c`)

Shows the specified number of bytes from the end of the file.

Example

```
tail -c 50 file.txt
```

Displays the last 50 bytes.

---

3. Follow a file in real time (`-f`)

Continuously displays new lines added to a file.

Example

```
tail -f /var/log/syslog
```

or

```
tail -f /var/log/messages
```

This is useful for:

- Monitoring system logs
- Watching application logs
- Debugging programs

Press **Ctrl + C** to stop.

4. Follow by filename (`-F`)

Similar to `-f`, but continues following the file even if it is renamed or recreated (common during log rotation).

```
tail -F /var/log/syslog
```

---

5. Display multiple files

```
tail file1.txt file2.txt
```

Displays the last 10 lines of each file with headers indicating the filename.

---

6. Show output starting from a specific line

```
tail -n +20 file.txt
```

Displays the file starting from line 20 to the end.

Practical Examples

View the last 20 lines of a log

```
tail -20 /var/log/auth.log
```

Monitor login activity

```
tail -f /var/log/auth.log
```

Watch an application log

```
tail -f app.log
```

As new log entries are written, they appear immediately.

Combine with another command

```
dmesg | tail
```

Shows the last 10 kernel messages.

Search within the last lines

```
tail -100 logfile.log | grep ERROR
```

Displays `ERROR` messages from the last 100 lines.

Sample File

Suppose `numbers.txt` contains:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

Command

```
tail numbers.txt
```

Output:

```
6
7
8
9
10
11
12
13
14
15
```

Command

```
tail -n 3 numbers.txt
```

Output:

```
13
14
15
```

Commonly Used Options Summary

Option	Description
`-n N`	Show the last **N** lines
`-c N`	Show the last **N** bytes
`-f`	Follow the file as new data is added
`-F`	Follow the file across log rotation or recreation
`-q`	Do not display file headers when showing multiple files
`-v`	Always display file headers
`--help`	Display help information
`--version`	Show the command version

Exit Status

- `0` – Command executed successfully.
- Non-zero – An error occurred (for example, the file does not exist or you do not have permission to read it).

Typical Use Cases

- Viewing the end of log files.
- Monitoring logs in real time.
- Checking the latest entries in large files.
- Debugging applications by watching live output.
- Displaying the most recent records in data files.

Tip: `tail` is the opposite of the `head` command. While `head` displays the beginning of a file, `tail` displays the end of a file.

6. nl

The `nl` command in Linux is used to **number the lines of a file**. Unlike `cat -n`, `nl` provides more control over which lines are numbered and how they are formatted.

Syntax

```
nl [OPTIONS] FILE
```

or

```
command | nl
```

Basic Example

Suppose `sample.txt` contains:

```
Apple
Banana

Cherry
Date
```

Run:

```
nl sample.txt
```

Output:

```
     1  Apple
     2  Banana

     3  Cherry
     4  Date
```

By default, **blank lines are not numbered**.

Common Options

Option	Description
`-b STYLE`	Specify which lines to number
`-n FORMAT`	Number format
`-w WIDTH`	Width of line numbers
`-s STRING`	Separator between number and text
`-v NUMBER`	Starting line number
`-i NUMBER`	Increment line numbers
`-l NUMBER`	Number consecutive blank lines together
`-h STYLE`	Number header lines
`-f STYLE`	Number footer lines

Examples

1. Number a file

```
nl sample.txt
```

Output:

```
     1  Apple
     2  Banana

     3  Cherry
     4  Date
```

---

2. Number all lines (including blank lines)

```
nl -b a sample.txt
```

Output:

```
     1  Apple
     2  Banana
     3
     4  Cherry
     5  Date
```

Here:

- `a` = number **all** lines

---

3. Number only non-empty lines (default)

```
nl -b t sample.txt
```

Output:

```
     1  Apple
     2  Banana

     3  Cherry
     4  Date
```

`t` means **text lines only**.

---

4. Start numbering from 100

```
nl -v 100 sample.txt
```

Output:

```
   100  Apple
   101  Banana

   102  Cherry
   103  Date
```

---

5. Increment numbers by 5

```
nl -i 5 sample.txt
```

Output:

```
     1  Apple
     6  Banana

    11  Cherry
    16  Date
```

6. Change separator

Default separator is a tab.

```
nl -s ": " sample.txt
```

Output:

```
1: Apple
2: Banana

3: Cherry
4: Date
```

---

7. Set number width

```
nl -w 3 sample.txt
```

Output:

```
  1 Apple
  2 Banana

  3 Cherry
  4 Date
```

---

8. Right-justify numbers with leading zeros

```
nl -n rz sample.txt
```

Output:

```
000001  Apple
000002  Banana

000003  Cherry
000004  Date
```

`-n` formats:

Format	Meaning
`ln`	Left justified
`rn`	Right justified (default)
`rz`	Right justified with leading zeros

9. Number command output

```
ls | nl
```

Output:

```
     1  Documents
     2  Downloads
     3  Music
     4  Pictures
```

Difference Between `nl` and `cat -n`

Suppose the file contains:
```
One
Two

Three
```

Using `cat -n`

```
cat -n sample.txt
```

Output:

```
     1  One
     2  Two
     3
     4  Three
```

Every line, including blank lines, is numbered.

Using `nl`

```
nl sample.txt
```

Output:

```
     1  One
     2  Two

     3  Three
```

Blank lines are skipped by default.

Practical Uses

- Add line numbers to source code for easier review.
- Reference specific lines in configuration or log files.
- Number the output of commands.
- Prepare numbered text for reports or documentation.

Summary

Feature	Description
Command	`nl`
Purpose	Number lines in text files or command output
Default behavior	Numbers only non-empty lines
Can number blank lines	Yes (`-b a`)
Can customize	Start number, increment, width, separator, formatting
Useful for	Source code, logs, scripts, documentation, and text processing

7. wc

The `wc` (**Word Count**) command is a standard Linux/Unix utility used to count **lines, words, characters, bytes**, and more in a file or from standard input.

Syntax

```
wc [OPTION]... [FILE]...
```

- **OPTION** – Specifies what to count.
- **FILE** – One or more files to analyze.

Default Behavior

If no option is specified, `wc` displays:

1. Number of lines
2. Number of words
3. Number of bytes
4. Filename

Example:

```
wc file.txt
```

Output:

```
15  120  840  file.txt
```

Meaning:

- **15** → Lines
- **120** → Words
- **840** → Bytes

Common Options

Option	Description
`-l`	Count lines
`-w`	Count words
`-c`	Count bytes
`-m`	Count characters
`-L`	Display length of the longest line

Examples

1. Count Lines

```
wc -l file.txt
```

Output:

```
15 file.txt
```

There are **15 lines**.

---

2. Count Words

```
wc -w file.txt
```

Output:

```
120 file.txt
```

There are **120 words**.

3. Count Characters

```
wc -m file.txt
```

Output:

```
835 file.txt
```

The file contains **835 characters**.

---

4. Count Bytes

```
wc -c file.txt
```

Output:

```
840 file.txt
```

The file size is **840 bytes**.

---

5. Longest Line Length

```
wc -L file.txt
```

Output:

```
74 file.txt
```

The longest line contains **74 characters**.

6. Count Multiple Files

```
wc file1.txt file2.txt
```

Example output:

```
15 120 840 file1.txt
20 180 950 file2.txt
35 300 1790 total
```

The last line shows the combined totals.

---

7. Count from Standard Input

```
echo "Hello Linux World" | wc
```

Output:

```
1 3 18
```

Meaning:

- **1 line**
- **3 words**
- **18 bytes**

---

8. Count Number of Files in a Directory

```
ls | wc -l
```

Example output:

```
25
```

There are **25 entries** in the current directory.

9. Count Running Processes

```
ps -ef | wc -l
```

Example output:

```
215
```

There are **215 running processes** (including the header line).

---

10. Count Matching Lines

```
grep "error" log.txt | wc -l
```

Output:

```
12
```

There are **12 lines** containing the word `"error"`.

Sample File

Suppose `sample.txt` contains:

```
Linux is powerful.
It is open source.
Learning Linux is fun.
```

Command:

```
wc sample.txt
```

Output:

```
3 10 58 sample.txt
```

Explanation:

- **3** lines
- **10** words
- **58** bytes

Practical Uses

- Count the number of lines in source code.
- Count words in reports or documents.
- Determine file size in bytes.
- Count log entries.
- Count search results using pipes (`grep`, `find`, `ps`, `ls`, etc.).

Summary

Command	Purpose
`wc file.txt`	Count lines, words, and bytes
`wc -l file.txt`	Count lines
`wc -w file.txt`	Count words
`wc -m file.txt`	Count characters
`wc -c file.txt`	Count bytes
`wc -L file.txt`	Length of the longest line
`ls | wc -l`	Count directory entries
`grep "text" file | wc -l`	Count matching lines

`wc` is commonly used with pipes (`|`) to quickly count output from other Linux commands, making it one of the most useful text-processing utilities in shell scripting and system administration.

8. cut

The `cut` command in Linux is used to **extract specific columns, fields, or characters** from each line of a file or command output.

It is commonly used to process text files, CSV files, log files, and command outputs.

Syntax

```
cut [OPTION]... [FILE]...
```

If no file is specified, `cut` reads from **standard input**.

Common Options

Option	Description
`-b`	Extract specific bytes
`-c`	Extract specific characters
`-f`	Extract specific fields (columns)
`-d`	Specify the field delimiter
`--complement`	Print everything except the selected fields
`-s`	Suppress lines without delimiters
`--output-delimiter`	Specify a different output delimiter

---

1. Extract Characters (`-c`)

Extract specific character positions.

Example

File (`sample.txt`)

```
Linux
Ubuntu
Fedora
```

Command

```
cut -c 1-3 sample.txt
```

Output :

```
Lin
Ubu
Fed
```

Extract the first character

```
cut -c1 sample.txt
```

Output :

```
L
U
F
```

Extract multiple character ranges

```
cut -c1-2,5 sample.txt
```

Output :

```
Lix
Uut
Fer
```

---

2. Extract Bytes (`-b`)

Works like `-c` for plain ASCII text, but extracts bytes instead of characters.

```
cut -b1-5 sample.txt
```

Output :

```
Linux
Ubunt
Fedor
```

Note: With UTF-8 or other multibyte characters, `-b` may split a character incorrectly. Use `-c` when working with text containing non-ASCII characters.

3. Extract Fields (`-f`)

Useful for structured text separated by a delimiter.

Example file (`students.txt`)

```
101:John:CSE
102:Alice:ECE
103:David:MECH
```

Extract the first field

```
cut -d ":" -f1 students.txt
```

Output :

```
101
102
103
```

Extract the second field

```
cut -d ":" -f2 students.txt
```

Output

```
John
Alice
David
```

Extract multiple fields

```
cut -d ":" -f1,3 students.txt
```

Output

```
101:CSE
102:ECE
103:MECH
```

Extract a range of fields

```
cut -d ":" -f2-3 students.txt
```

Output :

```
John:CSE
Alice:ECE
David:MECH
```

4. Specify Delimiter (`-d`)

By default, `cut` uses the **tab** character as the delimiter.

For comma-separated values:

```
cut -d "," -f2 employees.csv
```

Example

```
101,Ravi,50000
102,John,60000
```

Output

```
Ravi
John
```

---

5. Suppress Lines Without Delimiter (`-s`)

Example file

```
John:25
Linux
David:30
```

Command

```
cut -d ":" -f1 -s file.txt
```

Output

```
John
David
```

The line `Linux` is omitted because it does not contain the `:` delimiter.

---

6. Print Everything Except Selected Fields (`--complement`)

```
cut -d ":" -f2 --complement students.txt
```

Output :

```
101:CSE
102:ECE
103:MECH
```

7. Change the Output Delimiter

Input

```
John:25:India
```

Command

```
cut -d ":" -f1,3 --output-delimiter=" | " file.txt
```

Output

```
John | India
```

Practical Examples

Display usernames from `/etc/passwd`

```
cut -d ":" -f1 /etc/passwd
```

Output

```
root
daemon
bin
sys
bharath
```

Display user IDs

```
cut -d ":" -f3 /etc/passwd
```

Output

```
0
1
2
3
1000
```

Display home directories

```
cut -d ":" -f6 /etc/passwd
```

Output :

```
/root
/usr/sbin
/home/bharath
```

Display login shells

```
cut -d ":" -f7 /etc/passwd
```

Output

```
/bin/bash
/usr/sbin/nologin
/bin/bash
```

Extract the first column from a CSV file

```
cut -d "," -f1 marks.csv
```

Extract the second word from command output

```
echo "Linux Ubuntu Fedora" | cut -d " " -f2
```

Output :

```
Ubuntu
```

Extract the first 10 characters of a log file

```
cut -c1-10 logfile.txt
```

Difference Between `cut`, `awk`, and `sed`

Command	Best Used For
`cut`	Extracting fixed columns, fields, or character positions
`awk`	Advanced text processing, calculations, filtering, and reporting
`sed`	Searching, replacing, inserting, deleting, and editing text streams

---

Advantages

- Very fast and lightweight.
- Simple syntax.
- Ideal for extracting columns from delimited files.
- Works well in shell pipelines.

Limitations

- Cannot rearrange or compute fields like `awk`.
- Works best with consistently delimited input.
- Limited text-processing capabilities compared to `awk` or `sed`.

Summary

The `cut` command extracts selected portions of text from each line of input.

Most commonly used options:

Option	Purpose
`-c`	Extract characters
`-b`	Extract bytes
`-f`	Extract fields (columns)
`-d`	Specify field delimiter
`-s`	Skip lines without the delimiter
`--complement`	Output everything except the selected fields
`--output-delimiter`	Change the delimiter in the output

`cut` is one of the most useful commands for quickly extracting usernames, IDs, columns from CSV files, or specific parts of command output in Linux shell scripts.

9. paste

The `paste` command is used to **merge lines from multiple files** by placing them **side by side**. By default, it separates the columns with a **tab character**.

Syntax

```
paste [OPTIONS] FILE1 FILE2 ...
```

If you use `-` instead of a filename, `paste` reads from **standard input (stdin)**.

How `paste` Works

Suppose you have two files:

names.txt

```
Alice
Bob
Charlie
```

marks.txt

```
85
90
78
```

Command:

```
paste names.txt marks.txt
```

Output:

```
Alice    85
Bob      90
Charlie  78
```

The two files are combined line by line.

Common Options

1. `-d` (Specify a delimiter)

Instead of a tab, use another character.

Example:

```
paste -d "," names.txt marks.txt
```

Output:

```
Alice,85
Bob,90
Charlie,78
```

Another example:

```
paste -d ":" names.txt marks.txt
```

Output:

```
Alice:85
Bob:90
Charlie:78
```

---

2. `-s` (Serial mode)

Joins all lines from each file into a single line.

Example:

numbers.txt

```
1
2
3
4
```

Command:

```
paste -s numbers.txt
```

Output:

```
1	2	3	4
```

Using a comma:

```
paste -s -d "," numbers.txt
```

Output:

```
1,2,3,4
```

3. Multiple files

Suppose:

name.txt

```
Alice
Bob
```

age.txt

```
25
30
```

city.txt

```
London
Paris
```

Command:

```
paste name.txt age.txt city.txt
```

Output:

```
Alice    25    London
Bob      30    Paris
```

Read from Standard Input

```
echo -e "A\nB\nC" | paste -s
```

Output:

```
A    B    C
```

Create a CSV File

```
paste -d "," names.txt marks.txt > students.csv
```

Output (`students.csv`):

```
Alice,85
Bob,90
Charlie,78
```

If Files Have Different Numbers of Lines

Example:

file1.txt

```
A
B
C
```

file2.txt

```
1
2
```

Command:

```
paste file1.txt file2.txt
```

Output:

```
A    1
B    2
C
```

The missing value is left blank.

Difference Between `paste` and `cat`

Command	Purpose
`cat`	Displays or concatenates files vertically (one after another).
`paste`	Merges files horizontally (side by side).

Example:

```
cat file1.txt file2.txt
```

Output:

```
A
B
C
1
2
```

```
paste file1.txt file2.txt
```

Output:

```
A    1
B    2
C
```

Practical Uses

- Merge related data from multiple files.
- Combine names, IDs, and marks into a single table.
- Create CSV files from separate columns.
- Reformat text data for reports.
- Join multiple lists into one structured output.

Summary

Feature	Description
Purpose	Merge lines from files horizontally
Default delimiter	Tab (`\t`)
`-d`	Specify a custom delimiter
`-s`	Combine all lines of a file into a single line
Input	Files or standard input
Output	Side-by-side merged text

Basic Examples

```
paste file1.txt file2.txt
paste -d "," file1.txt file2.txt
paste -s file.txt
paste -s -d "," file.txt
```

10. sort

The `sort` command in Linux is used to **arrange the lines of a text file or standard input in a specific order**. By default, it sorts lines **alphabetically (lexicographically)**.

Syntax

```
sort [OPTIONS] [FILE]
```

- **OPTIONS** – Modify how the sorting is performed.
- **FILE** – The file whose contents you want to sort. If omitted, `sort` reads from standard input.

Basic Example

Suppose `names.txt` contains:

```
David
Alice
Charlie
Bob
```

Command:

```
sort names.txt
```

Output:

```
Alice
Bob
Charlie
David
```

Common Options

1. Sort in Reverse Order (`-r`)

```
sort -r names.txt
```

Output:

```
David
Charlie
Bob
Alice
```

---

2. Sort Numerically (`-n`)

File `numbers.txt`:

```
100
25
8
45
```

Command:

```
sort -n numbers.txt
```

Output:

```
8
25
45
100
```

Without `-n`, the output would be:

```
100
25
45
8
```

because it sorts alphabetically rather than numerically.

3. Remove Duplicate Lines (`-u`)

File:

```
Apple
Banana
Apple
Orange
Banana
```

Command:

```
sort -u fruits.txt
```

Output:

```
Apple
Banana
Orange
```

---

4. Ignore Case (`-f`)

File:

```
apple
Banana
APPLE
banana
```

Command:

```
sort -f fruits.txt
```

Treats uppercase and lowercase letters as equivalent while sorting.

---

5. Sort by a Specific Column (`-k`)

Suppose `students.txt` contains:

```
John 85
Alice 92
Bob 78
```

Sort by the second column:

```
sort -k2 students.txt
```

Output:

```
Bob 78
John 85
Alice 92
```

---

6. Sort by Month (`-M`)

File:

```
Mar
Jan
Dec
```

Command:

```
sort -M months.txt
```

Output:

```
Jan
Mar
Dec
```

---

7. Check if a File is Already Sorted (`-c`)

```
sort -c data.txt
```

- No output → File is already sorted.
- If not sorted:

```
sort: data.txt:3: disorder: Apple
```

8. Specify a Field Separator (`-t`)

For a CSV file:

```
John,85
Alice,92
Bob,78
```

Sort by marks:

```
sort -t',' -k2 -n marks.csv
```

Output:

```
Bob,78
John,85
Alice,92
```

---

9. Save Sorted Output to a File

```
sort names.txt > sorted_names.txt
```

---

10. Sort Multiple Files

```
sort file1.txt file2.txt
```

Sorts the combined contents of both files.

Practical Examples

Sort `/etc/passwd` by username

```
sort /etc/passwd
```

Sort file by file size (second column)

```
fileA 500
fileB 200
fileC 800
```

Command:

```
sort -k2 -n files.txt
```

Output:

```
fileB 200
fileA 500
fileC 800
```

Display Unique Sorted Words

```
cat words.txt | sort -u
```

Count Duplicate Entries

```
sort names.txt | uniq -c
```

Output:

```
3 Alice
2 Bob
1 Charlie
```

Commonly Used Options

Option	Description
`-r`	Reverse sort order
`-n`	Numeric sort
`-u`	Remove duplicate lines
`-f`	Ignore letter case
`-k N`	Sort by column N
`-t X`	Use X as field separator
`-M`	Sort month names
`-c`	Check whether input is sorted
`-o FILE`	Write output directly to a file
`-V`	Version (natural) sort (`1`, `2`, `10` instead of `1`, `10`, `2`)
`-h`	Sort human-readable sizes (e.g., `1K`, `50M`, `2G`)

Real-World Examples

Sort log entries alphabetically:

```
sort system.log
```

Sort process IDs numerically:

```
ps -e -o pid | sort -n
```

Sort disk usage by size:

```
du -h | sort -h
```

Sort software versions naturally:

```
printf "v1\nv10\nv2\n" | sort -V
```

Output:

```
v1
v2
v10
```

Summary

- `sort` arranges lines of text in ascending order by default.
- It can sort **alphabetically**, **numerically**, by **specific columns**, **months**, **human-readable sizes**, or **version numbers**.
- It works with files or standard input and is commonly combined with commands such as `cat`, `uniq`, `cut`, `awk`, `grep`, and `ps` for text processing.

11. uniq

The `uniq` command is used to **filter out repeated adjacent lines** from a text file or standard input. It can also display only duplicate lines or count how many times each line appears.

Important: `uniq` only works on **consecutive (adjacent) duplicate lines**. If duplicate lines are scattered throughout the file, you should sort the file first.

Syntax

```
uniq [OPTION]... [INPUT_FILE] [OUTPUT_FILE]
```

- `INPUT_FILE` – The file to read.
- `OUTPUT_FILE` – (Optional) The file where the result is written. If omitted, output is displayed on the terminal.

Basic Example

Suppose `fruits.txt` contains:

```
Apple
Apple
Banana
Banana
Banana
Orange
Apple
```

Run:

```
uniq fruits.txt
```

Output:

```
Apple
Banana
Orange
Apple
```

Notice the last `Apple` remains because it is **not adjacent** to the first two.

Remove All Duplicates

Sort the file first:

```
sort fruits.txt | uniq
```

Output:

```
Apple
Banana
Orange
```

Common Options

1. `-c` (Count Occurrences)

Displays the number of consecutive occurrences.

```
uniq -c fruits.txt
```

Output:

```
      2 Apple
      3 Banana
      1 Orange
      1 Apple
```

---

2. `-d` (Show Only Duplicate Lines)

```
uniq -d fruits.txt
```

Output:

```
Apple
Banana
```

Only lines that occur consecutively more than once are shown.

3. `-u` (Show Only Unique Lines)

Shows lines that appear only once consecutively.

```
uniq -u fruits.txt
```

Output:

```
Orange
Apple
```

---

4. `-i` (Ignore Case)

Suppose `names.txt` contains:

```
John
john
JOHN
Alice
```

Run:

```
sort -f names.txt | uniq -i
```

Output:

```
Alice
John
```

---

5. `--all-repeated`

Displays all repeated groups.

```
uniq --all-repeated fruits.txt
```

Useful for examining duplicate groups in detail.

6. `-f N` (Skip Fields)

Skip the first **N fields** before comparing.

Example (`employees.txt`):

```
101 John
102 John
103 Alice
104 Alice

uniq -f 1 employees.txt
```

Comparison starts from the second field (the names).

---

7. `-s N` (Skip Characters)

Skip the first **N characters** before comparison.

```
uniq -s 3 file.txt
```

---

8. `-w N` (Compare Only First N Characters)

```
uniq -w 5 file.txt
```

Only the first five characters of each line are compared.

Practical Examples

Remove duplicate usernames

```
sort users.txt | uniq
```

Count repeated IP addresses

```
sort access.log | uniq -c
```

Find duplicate email addresses

```
sort emails.txt | uniq -d
```

Find entries that occur only once

```
sort records.txt | uniq -u
```

Count word frequency

```
cat words.txt | sort | uniq -c | sort -nr
```

Example output:

```
15 Linux
10 Ubuntu
 8 Bash
 3 Shell
```

Exit Status

- `0` – Command completed successfully.
- Non-zero – An error occurred (such as an unreadable input file).

Difference Between `sort` and `uniq`

Command	Purpose
`sort`	Arranges lines in alphabetical or numerical order.
`uniq`	Removes or reports adjacent duplicate lines.
`sort | uniq`	Sorts the input first, then removes all duplicates.

Summary

Command	Description
`uniq file.txt`	Remove adjacent duplicate lines
`uniq -c file.txt`	Count occurrences of adjacent lines
`uniq -d file.txt`	Show only duplicate lines
`uniq -u file.txt`	Show only unique (non-duplicated) lines
`sort file.txt | uniq`	Remove all duplicates after sorting
`sort file.txt | uniq -c`	Count occurrences of each unique line
`sort file.txt | uniq -d`	Display duplicate lines after sorting
`uniq -i file.txt`	Ignore differences in letter case

When to use `uniq`

Use `uniq` whenever you need to:

- Remove duplicate lines from text files.
- Count repeated entries in logs.
- Find duplicate usernames, email addresses, or IP addresses.
- Generate frequency counts after sorting.
- Clean up sorted datasets by eliminating repeated records.

12. tr

The `tr` (**translate**) command is a Unix/Linux utility used to **translate, replace, delete, or squeeze characters** from standard input. It does **not** modify files directly; it reads input from standard input (stdin) and writes the result to standard output (stdout).

Syntax

```
tr [OPTION] SET1 [SET2]
```

- **SET1** – Characters to search for.
- **SET2** – Characters to replace them with (if translating).

Common Options

Option	Description
`-d`	Delete characters in SET1
`-s`	Squeeze repeated characters into one
`-c`	Complement SET1 (operate on characters NOT in SET1)
`-t`	Truncate SET1 to match the length of SET2

Examples

1. Convert lowercase to uppercase

```
echo "linux" | tr 'a-z' 'A-Z'
```

Output:

```
LINUX
```

---

2. Convert uppercase to lowercase

```
echo "LINUX" | tr 'A-Z' 'a-z'
```

Output:

```
linux
```

3. Replace one character with another

```
echo "banana" | tr 'a' 'o'
```

Output:

```
bonono
```

Every a becomes o.

---

4. Replace spaces with underscores

```
echo "Linux Shell Commands" | tr ' ' '_'
```

Output

```
Linux_Shell_Commands
```

---

5. Delete characters

Remove all vowels:

```
echo "Operating System" | tr -d 'aeiouAEIOU'
```

Output

```
prtng Systm
```

---

6. Remove digits

```
echo "abc123xyz456" | tr -d '0-9'
```

Output

```
abcxyz
```

7. Squeeze repeated spaces

```
echo "Linux    Shell      Commands" | tr -s ' '
```

Output

```
Linux Shell Commands
```

Multiple spaces become a single space.

---

8. Squeeze repeated characters

```
echo "heeellooo" | tr -s 'eo'
```

Output

```
helo
```

Repeated `e` and `o` characters are reduced to one occurrence.

---

9. Delete everything except digits

```
echo "Phone: 98765-43210" | tr -cd '0-9'
```

Output

```
9876543210
```

Explanation:

- `-c` → Complement (everything except digits)
- `-d` → Delete those characters

---

10. Replace tabs with spaces

```
cat file.txt | tr '\t' ' '
```

11. Count characters

```
echo "Hello World" | tr -cd 'a-zA-Z' | wc -c
```

Output :

```
10
```

Counts only alphabetic characters.

---

12. Convert newline to space

```
printf "one\ntwo\nthree\n" | tr '\n' ' '
```

Output :

```
one two three
```

---

13. Convert colon-separated data to newline-separated

```
echo "apple:banana:orange" | tr ':' '\n'
```

Output :

```
apple
banana
orange
```

---

14. Remove carriage returns (Windows → Linux text)

```
tr -d '\r' < windows.txt > linux.txt
```

---

15. Read from a file

Input (`names.txt`):

```
alice
bob
charlie
```

Command:

```
tr 'a-z' 'A-Z' < names.txt
```

Output :

```
ALICE
BOB
CHARLIE
```

Character Classes

Instead of ranges, you can use POSIX character classes.

Class	Meaning
`[:lower:]`	Lowercase letters
`[:upper:]`	Uppercase letters
`[:alpha:]`	Alphabetic characters
`[:digit:]`	Digits (0–9)
`[:alnum:]`	Letters and digits
`[:space:]`	Spaces, tabs, newlines
`[:punct:]`	Punctuation
`[:blank:]`	Spaces and tabs

Example:

```
echo "linux" | tr '[:lower:]' '[:upper:]'
```

Output :

```
LINUX
```

Practical Use Cases

- Convert text to uppercase or lowercase.
- Replace delimiters (`,` ↔ `:` ↔ `|`).
- Remove unwanted characters.
- Clean log files.
- Normalize whitespace.

- Extract numbers from mixed text.
- Convert Windows text files to Unix format.
- Prepare data for shell scripts and text processing.

Limitations

- `tr` works **only on characters**, not on whole words or strings.
- It reads from **standard input** and writes to **standard output**.
- It **cannot edit files in place**. To save changes, redirect the output:

```
tr 'a-z' 'A-Z' < input.txt > output.txt
```

Summary

Task	Command
Uppercase	`tr 'a-z' 'A-Z'`
Lowercase	`tr 'A-Z' 'a-z'`
Delete characters	`tr -d 'chars'`
Remove digits	`tr -d '0-9'`
Replace spaces	`tr ' ' '_'`
Squeeze spaces	`tr -s ' '`
Keep only digits	`tr -cd '0-9'`
Convert `:` to newline	`tr ':' '\n'`

`tr` is one of the simplest and fastest Unix text-processing tools. It's commonly combined with commands like `echo`, `cat`, `cut`, `grep`, `sort`, `uniq`, and `awk` in shell pipelines to transform character data efficiently.

13. fmt

The **`fmt`** command in Linux is used to **format plain text** by wrapping long lines to a specified width. It is commonly used to make text files, emails, and documents easier to read.

Syntax

```
fmt [OPTION]... [FILE]...
```

If no file is specified, `fmt` reads from **standard input (stdin)**.

Common Options

Option	Description
`-w WIDTH`	Set the maximum line width (default is 75 characters).
`-WIDTH`	Short form of `-w WIDTH`. Example: `-60`.
`-s`	Split only long lines; do not join short lines.
`-u`	Uniform spacing (reduces multiple spaces between words to one, except after sentence endings).
`-t`	Preserve indentation of the first line of a paragraph.
`-c`	Crown margin mode (preserves indentation patterns).
`-p PREFIX`	Reformat only lines beginning with the specified prefix.

Examples

1. Format a text file

```
fmt file.txt
```

Input (`file.txt`):

```
Linux is a powerful operating system that provides many utilities for text
processing and system administration.
```

Output:

```
Linux is a powerful operating system that provides many
utilities for text processing and system administration.
```

---

2. Set line width to 40 characters

```
fmt -w 40 file.txt
```

Output:

```
Linux is a powerful operating
system that provides many
utilities for text processing
and system administration.
```

---

3. Read from standard input

```
echo "Linux provides many useful command line tools for administrators." | fmt
```

Output:

```
Linux provides many useful command line tools for
administrators.
```

---

4. Save formatted output to another file

```
fmt file.txt > formatted.txt
```

---

5. Split only long lines

```
fmt -s file.txt
```

This wraps only lines that are longer than the specified width, without combining adjacent short lines into a paragraph.

6. Use a width of 50 characters

```
fmt -50 file.txt
```

Equivalent to:

```
fmt -w 50 file.txt
```

---

7. Uniform spacing

Input:

```
Hello,    world!    Linux     is    great.
```

Command:

```
fmt -u file.txt
```

Output:

```
Hello, world! Linux is great.
```

Practical Uses

- Reformat long text files.
- Wrap paragraphs for terminal viewing.
- Clean up plain-text emails.
- Prepare README or documentation files with consistent line lengths.
- Improve readability of log or report files.

Exit Status

- **0** – Successfully formatted the text.
- **>0** – An error occurred (for example, the input file does not exist).

Difference Between `fmt` and `fold`

Feature	`fmt`	`fold`
Wraps at word boundaries	Yes	No (may split words)
Reflows paragraphs	Yes	No
Preserves word readability	Yes	May break words
Best for	Paragraph formatting	Fixed-width line splitting

Example

Original:

```
Linux provides many useful command line utilities for system administrators.
```

Using `fmt -30`:

```
Linux provides many useful
command line utilities for
system administrators.
```

Using `fold -w 30`:

```
Linux provides many useful co
mmand line utilities for syst
em administrators.
```

Summary

- **Command:** `fmt`
- **Purpose:** Reformat and wrap plain text into neatly formatted paragraphs.
- **Default width:** 75 characters.
- **Most common option:** `-w WIDTH` to set the desired line length.
- **Best use case:** Formatting text files while keeping words intact and improving readability.