LINUX PROGAMMING -10

ANUSHKA K

ENG24CY0009

touch Command

1) History, Invention, and Origin of the Command

The touch command is one of the early utilities created during the development of Unix at Bell Labs in the late 1970s.

It first appeared in Version 7 Unix (1979), which was a major release that standardized many tools.

Creators :

Although the exact single author is not publicly credited, the command emerged from the work of the original Unix team, including pioneers like Ken Thompson and Dennis Ritchie, who designed many foundational tools. The concept of “touching” a file—i.e., updating timestamps—came from the need to simplify file management and build processes.

As Unix evolved, touch became part of the POSIX standard, ensuring its presence in all major Linux and Unix systems.

2) Why the Command Is Useful for System Administrators

System Administrators use touch frequently because it is:

-> A Quick Way to Create Empty Files

Useful for placeholders

Helps prepare configuration files

Helps automate script output files

-> Essential for Scripts and Cron Jobs

Some tasks rely on file timestamps (e.g., backup scripts, log rotations)

Updating a timestamp can trigger programs to perform action

-> Great for Testing and Troubleshooting

Create sample files instantly

Temporary logs or dummy files can be produced on the spot

->Timestamp Manipulation

Modifying access or modification time helps:

reset file statuses

simulate file changes

adjust build processes

Overall, touch is a small but powerful tool essential in server maintenance, automation, and development workflows.

3) How the Command Works (with Example)

Basic Working

touch has two main functions:

1. Create a file (if it doesn't exist)

2. Update timestamps (access time and modification time)

Example 1: Creating an empty file

touch sample.txt

If sample.txt does not exist → it will be created with size 0 bytes.

Example 2: Updating timestamps

touch existing.txt

If existing.txt already exists → only its access and modification times change.

Example 3: Creating multiple files at once

touch a.txt b.txt c.txt

->Below are text-based illustrations similar to what would appear on a Linux terminal.

Illustration 1: Creating a file

$ touch report.txt

$ ls -l report.txt

-rw-r--r-- 1 user user 0 Feb 10 12:45 report.txt

Illustration 2: Timestamp update

$ ls -l notes.txt

-rw-r--r-- 1 user user 120 Feb 10 11:00 notes.txt

$ touch notes.txt

$ ls -l notes.txt

-rw-r--r-- 1 user user 120 Feb 10 12:47 notes.txt

Illustration 3: Multiple file creation

$ touch file1 file2 file3

$ ls

file1 file2 file3

5) Five to Six Important Options/Flags of touch

Here are the most commonly used and important options.

1. -a – Update only access time

touch -a example.txt

Only the access time changes; modification time stays unchanged.

2. -m – Update only modification time

touch -m example.txt

3. -c or --no-create – Do not create a file if it does not exist

touch -c not_here.txt

If the file does not exist → do nothing

4. -t – Specify a custom timestamp

Format: [[CC]YY]MMDDhhmm[.ss]

Example:

touch -t 202402101200 file.txt

Sets date/time → 2024 Feb 10, 12:00

5. -r – Copy the timestamp from another file

touch -r source.txt target.txt

Makes both files have identical timestamps.

6. --date – Set a timestamp using readable date formats

(Available in GNU/Linux)

touch --date="2024-02-10 09:15" new.txt

cat Command – Detailed Explanation

1) What Is the Command? (History, Origin, Author)

The cat command (short for concatenate) is one of the oldest and most commonly used Unix utilities. It first appeared in Version 1 Unix (1971), developed at Bell Labs.

The early Unix development team—led by Ken Thompson and Dennis Ritchie—designed many small tools that do one job well. cat was created to:

Read file contents

Combine (concatenate) multiple files

Display text output to standard output (terminal)

Over time, cat became a foundational command in almost all Unix-like systems and is now part of the POSIX standard, ensuring compatibility across Linux, macOS, and BSD systems.

2) Why the Command Is Useful for System Administrators

System administrators frequently use cat because it is:

Fast and Simple for Viewing File Contents

No need to open an editor—just display the file immediately.

Very Useful for Log and Configuration Checks

Quickly inspect:

log files

configuration files

temporary script outputs

✓ Reliable for Debugging

Used to:

verify script results

check server processes

inspect error messages

✓ Helpful for Combining Files

Admins often merge:

backup files

configuration fragments

text-based reports

✓ Ideal in Shell Scripting

Used inside scripts for automation, printing messages, or handling text streams.

3) How the Command Works (with Examples)

Basic Working Principle

The cat command:

1. Opens the file

2. Reads its contents

3. Sends the output to stdout (terminal)

If multiple files are given, cat prints them in the same order

Example 1: Viewing a file

cat message.txt

Example 2: Viewing multiple files

cat part1.txt part2.txt

Example 3: Creating a file using cat

cat > notes.txt

This is my note.

Press CTRL + D to save.

Example 4: Appending text to a file

cat >> notes.txt

More lines added.

Press CTRL + D again.

Example 5: Redirecting output to another file

cat file1 file2 > merged.txt

4) Snapshot-Style Illustrations of Command Execution

Below are text-based simulations similar to real Linux terminal output

Illustration 1: Viewing a File

$ cat story.txt

Once upon a time,

Linux made computing simple.

Illustration 2: Combining File

$ cat header.txt body.txt footer.txt

Report Title

Main Data Section

End of Report

Illustration 3: Creating a File

$ cat > demo.txt

Hello World

(This line typed by user)

^D ← CTRL + D pressed

$ cat demo.txt

Hello World

Illustration 4: Line Numbering Output

$ cat -n code.sh

```
1  #!/bin/bash
2  echo "Hello"
3  echo "World"
```

5) Important Options and Flags (5–6 Fully Explained)

Below are the most useful and commonly used options.

1. -n — Display Line Numbers

cat -n file.txt

Adds a number to every line.

2. -b — Number Only Non-Empty Lines

cat -b file.txt

Empty lines are not numbered.

3. -E — Show End-of-Line Marker

cat -E file.txt

Displays $ at the end of each line.

Useful for checking spaces/trailing characters.

4. -s — Squeeze Repeated Blank Lines

cat -s file.txt

If a file has many blank lines, they are reduced to one.

5. -A — Show All Special Characters

Equivalent to using -vET

cat -A file.txt

Shows:

tabs

non-printable characters

end-of-line markers

Excellent for debugging formatting issues.

6. -T — Show Tabs as ^I

cat -T file.txt

Reveals tab characters, useful in coding and configuration files.

Sources for Detailed Flags and Options

To see the full official documentation:

1. Linux Manual Pages

man cat

2. GNU Coreutils Documentation

Provides detailed descriptions for all GNU/Linux versions of cat.

3. POSIX Standard Documentation

For portable version behavior across Unix-like systems.