

Seven Fundamental File Types in Linux

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 Linux File Types 1
2. Regular Files 2
    2.1. What is a Regular File ? 3
    2.2. Creating a Regular File 4
    2.3. Writing Data to a Regular File 4
    2.4. Binary Regular Files 5
    2.5. Executable Regular Files 5
    2.6. Regular Files and Permissions 6
3. Directories 6
    3.1. What is a Directory ? 7
    3.2. Creating a Directory 7
    3.3. Directory Permissions 8
    3.4. Important Directory Entries 8
    3.5. Directory Tree 9
4. Character Device Files 9
    4.1. What is a Character Device ? 10
    4.2. /dev/null 10
    4.3. /dev/zero 10
    4.4. /dev/random and /dev/urandom 11
    4.5. Terminal Devices 11
    4.6. Major and Minor Numbers 11
5. Block Device Files 12
    5.1. What is a Block Device ? 12
    5.2. SATA Disk Example 13
    5.3. NVM-e Devices 13
    5.4. Character Device vs Block Device 14
    5.5. Why Block Devices Are Important 14

6. Symbolic Links 15
6.1. What is a Symbolic Link ? 15
6.2. Creating a Symbolic Link 16
6.3. Viewing a Symbolic Link 16
6.4. Relative and Absolute Symbolic Links 16
6.5. Broken Symbolic Links 17
6.6. Symbolic Link vs Regular File 17
6.7. Symbolic Links in Linux Administration 18
7. Named Pipes / FIFO 18
7.1. What is a Pipe ? 18
7.2. What is a Named Pipe ? 19
7.3. Creating a FIFO 19
7.4. Using a FIFO 20
7.5. FIFO Does Not Normally Store Data Like a Regular File 20
7.6. Uses of Named Pipes 20
8. Sockets 21
8.1. What is a Socket ? 21
8.2. Unix Domain Sockets 21
8.3. Network Sockets 22
8.4. Socket Example 22
8.5. Viewing Sockets 22
9. Comparing the Seven File Types 23
10. Identifying File Types Using **ls** 23
11. Using the **file** Command 24
12. Using **stat** 24
13. Inodes and File Types 25
14. File Type and Inode 26
15. File Descriptors 27
16. /dev and Device Files 27
17. Special Files 28
18. Practical Example: Creating Different File Types 28
19. File Type Testing in Shell Scripts 29
20. Important File-Test Operators 30

21. A Practical Shell Script 30
22. The find Command and File Types 31
23. Why Linux Has Seven File Types 32
24. Difference Between File Extension and File Type 33
25. Seven File Types Example 34
26. Major Differences 34
27. Important Commands to Remember 35
28. Common Beginner Mistakes 36
29. Easy Way to Memorize the Seven Types 38
30. Conclusion 39

Seven Fundamental File Types in Linux

Linux follows a powerful philosophy: **“Everything is a file.”** This does not mean that every object is an ordinary text file. Instead, Linux provides a unified way of interacting with files, directories, devices, sockets, pipes, and other resources through the filesystem interface.

The Linux filesystem recognizes **seven fundamental file types**:

1. **Regular file (`-`)**
2. **Directory (`d`)**
3. **Character device file (`c`)**
4. **Block device file (`b`)**
5. **Symbolic link (`l`)**
6. **Named pipe / FIFO (`p`)**
7. **Socket (`s`)**

These file types can be identified using commands such as `ls -l`, `file`, and `stat`.

---

1. Introduction to Linux File Types

In Linux, files are not limited to documents such as `.txt`, `.pdf`, or `.jpg`. The filesystem treats many different resources as file-like objects.

For example:

```
ls -l
```

may display:

```
-rw-r--r--  1 user user  1200 Aug 24 08:00 notes.txt
drwxr-xr-x  2 user user  4096 Aug 24 08:01 Documents
lrwxrwxrwx  1 user user    11 Aug 24 08:02 current -> Documents
crw-rw-rw-  1 root root  1, 3 Aug 24 08:03 null
brw-rw----  1 root disk  8, 0 Aug 24 08:04 sda
prw-r--r--  1 user user     0 Aug 24 08:05 mypipe
srwxr-xr-x  1 user user     0 Aug 24 08:06 mysocket
```

The **first character** of each line indicates the file type.

First character	File type
`-`	Regular file
`d`	Directory
`c`	Character device
`b`	Block device
`l`	Symbolic link
`p`	Named pipe / FIFO
`s`	Socket

For example:

```
-rw-r--r--
^
|
Regular file
```

and:

```
drwxr-xr-x
^
|
Directory
```

Understanding these seven types is important for Linux administration, shell scripting, system programming, permissions, storage management, networking, and troubleshooting.

---

2. Regular Files

The first and most common Linux file type is the **regular file**.

Its symbol is:

`-`

For example:

```
-rw-r--r-- 1 bharath bharath 1250 Aug 24 08:00 notes.txt
```

The first character is `-`, so this is a regular file.

2.1. What is a Regular File ?

A regular file is a file used to store ordinary data.

Examples include:

- Text files
- Configuration files
- Shell scripts
- Source-code files
- Images
- Videos
- Audio files
- PDF documents
- Database files
- Executable programs
- Compressed archives

Examples:

```
notes.txt
report.pdf
photo.jpg
movie.mkv
script.sh
program
database.db
archive.tar.gz
```

Linux does not fundamentally classify these files according to their extensions. The extension is primarily a naming convention.

For example:

```
movie.mkv
```

may be a video file, but Linux identifies the underlying file as a **regular file**.

2.2. Creating a Regular File

The `touch` command can create an empty regular file:

```
touch notes.txt
```

Check it:

```
ls -l notes.txt
```

Output may be:

```
-rw-r--r-- 1 user user 0 Aug 24 08:10 notes.txt
```

The first character is:

```
-
```

Therefore, it is a regular file.

2.3. Writing Data to a Regular File

You can use `echo`:

```
echo "Hello Linux" > notes.txt
```

Read it:

```
cat notes.txt
```

Output:

```
Hello Linux
```

You can also use:

```
nano notes.txt
```

or:

```
vi notes.txt
```

2.4. Binary Regular Files

Regular files do not have to contain readable text.

For example:

```
photo.jpg
movie.mkv
program
archive.zip
```

are generally binary files.

You can check a file using:

```
file movie.mkv
```

Example:

```
movie.mkv: Matroska data
```

The important point is that a video file, image file, executable file, and text file can all be **regular files**.

2.5. Executable Regular Files

Linux programs are often stored as regular files.
For example:

```
ls -l /bin/ls
```

may show:

```
-rwxr-xr-x 1 root root ... /bin/ls
```

The first character is `-`, meaning it is a regular file.

The `x` permissions indicate that it is executable.

2.6. Regular Files and Permissions

A regular file can have permissions such as:

```
-rwxr-xr--
```

These permissions represent:

```
Owner:        rwx
Group:        r-x
Others:       r--
```

where:

- `r` = read
- `w` = write
- `x` = execute

Permissions are an important part of Linux security.

---

3. Directories

The second fundamental file type is the **directory**.

Its symbol is:

```
d
```

Example:

```
drwxr-xr-x 2 user user 4096 Aug 24 08:10 Documents
```

The first character is `d`.

Therefore:

```
Documents
```

is a directory.

3.1. What is a Directory?

A directory is a filesystem object used to organize and reference other files and directories.

For example:

```
/home/user/Documents
```

contains files and possibly other directories.

A simplified filesystem may look like:

```
/
├── home
│   └── user
│       ├── Documents
│       ├── Downloads
│       ├── Pictures
│       └── Videos
├── etc
├── var
├── usr
└── tmp
```

Directories provide structure to the filesystem.

3.2. Creating a Directory

Use:

```
mkdir Documents
```

Then:

```
ls -ld Documents
```

Example:

```
drwxr-xr-x 2 user user 4096 Aug 24 08:15 Documents
```

The d indicates a directory.

3.3. Directory Permissions

Directory permissions behave somewhat differently from regular-file permissions.

Read permission

`r` allows the user to list directory entries.

Write permission

`w` allows creation, deletion, or renaming of entries, subject to other restrictions.

Execute permission

`x` allows entering/traversing the directory and accessing entries when permissions permit.

For example:

```
drwxr-x---
```

means:

```
Owner: rwx
Group: r-x
Others: ---
```

3.4. Important Directory Entries

Every normal directory contains two special entries:

```
.
..
```

`.` means the current directory.

`..` means the parent directory.

You can see them with:

```
ls -la
```

Example:

```
.
..
notes.txt
Documents
```

3.5. Directory Tree

Linux organizes directories in a hierarchical tree.

The top is:

```
/
```

This is called the **root directory**.

Common directories include:

```
/bin
/boot
/dev
/etc
/home
/lib
/media
/mnt
/opt
/proc
/root
/run
/sbin
/srv
/sys
/tmp
/usr
/var
```

Different Linux distributions may vary somewhat in exact layout, especially with merged `/usr` layouts, but these names are foundational.

---

4. Character Device Files

The third fundamental file type is the **character device file**.

Its symbol is:

```
c
```

Example:

```
crw-rw-rw- 1 root root 1, 3 Aug 24 08:20 /dev/null
```

The first character is `c`.

4.1. What is a Character Device?

A character device represents a device or kernel interface that generally transfers data as a stream of characters or bytes.

Examples can include:

```
/dev/tty
/dev/zero
/dev/random
/dev/null
```

Some hardware devices can also appear as character devices.

4.2. `/dev/null`

One of the most famous Linux character devices is:

```
/dev/null
```

It discards data written to it.

Example:

```
echo "Hello" > /dev/null
```

The text is discarded.

This is useful when you don't want command output displayed.

For example:

```
command > /dev/null
```

4.3. `/dev/zero`

Another important character device is:

```
/dev/zero
```

It provides a stream of zero bytes.

For example:

```
head -c 10 /dev/zero | od -An -t x1
```

This can demonstrate zero-valued bytes.

4.4. `/dev/random` and `/dev/urandom`

Linux also provides random-data interfaces:

```
/dev/random
/dev/urandom
```

These are commonly used for obtaining random bytes.

For example:

```
head -c 16 /dev/urandom | od -An -t x1
```

4.5. Terminal Devices

Terminal interfaces can also appear under `/dev`.

For example:

```
ls -l /dev/tty
```

might show:

```
crw-rw-rw- 1 root tty 5, 0 ... /dev/tty
```

The `c` tells us that it is a character device.

4.6. Major and Minor Numbers

Character and block devices usually have **major and minor device numbers**.

For example:

```
crw-rw-rw- 1 root root 1, 3 /dev/null
```

Here:

```
Major number = 1
Minor number = 3
```

The major number generally identifies the device driver or device class, while the minor number distinguishes particular devices handled by that driver.

Modern Linux device handling is more sophisticated than simply thinking of these numbers as a complete description, but the major/minor concept remains fundamental.

---

5. Block Device Files

The fourth fundamental file type is the **block device**.

Its symbol is:

```
b
```

Example:

```
brw-rw---- 1 root disk 8, 0 Aug 24 08:25 /dev/sda
```

The first character is:

```
b
```

Therefore it is a block device.

5.1. What is a Block Device?

A block device represents storage that is accessed in blocks.

Typical examples include:

- Hard disk drives
- SSDs
- Storage partitions
- Some removable storage devices
- Virtual block devices

Examples:

```
/dev/sda
/dev/sda1
/dev/nvme0n1
/dev/nvme0n1p1
```

The exact device names depend on the hardware and kernel configuration.

5.2. SATA Disk Example

A SATA disk might appear as:

```
/dev/sda
```

Its partitions could be:

```
/dev/sda1
/dev/sda2
/dev/sda3
```

You can inspect block devices with:

```
lsblk
```

Example:

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0   500G  0 disk
├─sda1   8:1    0   100G  0 part /
└─sda2   8:2    0   400G  0 part /home
```

The TYPE column indicates:

```
disk
part
```

5.3. NVMe Devices

Modern SSDs often appear as:

```
/dev/nvme0n1
```

Partitions may appear as:

```
/dev/nvme0n1p1
/dev/nvme0n1p2
```

Check them using:

```
lsblk
```

5.4. Character Device vs Block Device

The main conceptual difference is the way the kernel interface handles data.

Character device

`c`

Generally provides a stream-oriented interface.

Examples:

```
/dev/tty
/dev/null
```

Block device

`b`

Provides block-oriented access to storage devices.

Examples:

```
/dev/sda
/dev/nvme0n1
```

5.5. Why Block Devices Are Important

Filesystems are commonly created on block devices.

For example:

```
sudo mkfs.ext4 /dev/sdb1
```

This creates an ext4 filesystem on a partition.

The partition is a block device.

It can then be mounted:

```
sudo mount /dev/sdb1 /mnt
```

6. Symbolic Links

The fifth fundamental file type is the **symbolic link**, commonly called a **symlink**.

Its symbol is:

```
l
```

Example:

```
lrwxrwxrwx 1 user user 11 Aug 24 08:30 current -> Documents
```

The first character is:

```
l
```

6.1. What is a Symbolic Link?

A symbolic link is a special filesystem object containing a pathname that refers to another file or directory.

For example:

```
Documents
```

could be the actual directory.

We can create:

```
current
```

as a symbolic link to it.

Command:

```
ln -s Documents current
```

Now:

```
current -> Documents
```

6.2. Creating a Symbolic Link

Syntax:

```
ln -s TARGET LINK_NAME
```

Example:

```
ln -s /home/user/Documents /home/user/docs
```

Now:

```
docs -> /home/user/Documents
```

6.3. Viewing a Symbolic Link

Use:

```
ls -l docs
```

Example:

```
lrwxrwxrwx 1 user user 21 Aug 24 08:35 docs -> /home/user/Documents
```

The arrow:

```
->
```

shows the target.

6.4. Relative and Absolute Symbolic Links

A symbolic link can contain an absolute path:

```
/home/user/Documents
```

or a relative path:

```
Documents
```

For example:

```
ln -s Documents current
```

creates a relative symbolic link.

6.5. Broken Symbolic Links

A symbolic link can point to something that no longer exists.

Example:

```
ln -s missing.txt broken
```

Now:

```
broken -> missing.txt
```

but:

```
missing.txt
```

doesn't exist.

This is called a **broken symlink**.

You can identify such links with:

```
find . -xtype l
```

6.6. Symbolic Link vs Regular File

Suppose:

```
original.txt
link.txt -> original.txt
```

`original.txt` is a regular file.

`link.txt` is a symbolic link.

They are different filesystem objects.

Check:

```
ls -l
```

You might see:

```
-rw-r--r-- 1 user user 100 original.txt
lrwxrwxrwx 1 user user  12 link.txt -> original.txt
```

6.7. Symbolic Links in Linux Administration

Symbolic links are widely used in Linux.

Examples include:

- Configuration management
- Alternative versions of software
- Web server document roots
- Library version management
- Application deployment
- Shortcut-like filesystem organization

They are especially useful when a program expects a specific pathname.

---

7. Named Pipes / FIFO

The sixth fundamental file type is the **named pipe**, also called a **FIFO**.

Its symbol is:

```
p
```

Example:

```
prw-r--r-- 1 user user 0 Aug 24 08:40 mypipe
```

The first character is:

```
p
```

7.1. What is a Pipe?

A pipe provides a mechanism for transferring data from one process to another.

The classic Linux pipe is:

```
command1 | command2
```

For example:

```
ls | grep txt
```

The output from `ls` is passed to `grep`.

This is an example of an **unnamed pipe**.

7.2. What is a Named Pipe?

A named pipe is a filesystem object that allows processes to communicate through a pathname.

It is called a FIFO because data is generally handled in:

```
First In
First Out
```

order.

7.3. Creating a FIFO

Use:

```
mkfifo mypipe
```

Check it:

```
ls -l mypipe
```

Example:

```
prw-r--r-- 1 user user 0 Aug 24 08:45 mypipe
```

The first character `p` identifies the FIFO.

7.4. Using a FIFO

Open one terminal and run:

```
cat > mypipe
```

In another terminal:

```
cat mypipe
```

Now text entered into the first terminal can be read by the second process.

The FIFO provides communication between the processes.

7.5. FIFO Does Not Normally Store Data Like a Regular File

This is an important distinction.

A regular file stores data persistently:

```
file.txt
```

A FIFO is primarily a communication mechanism.

Data written to it is consumed by another process.

It is not simply a normal disk file containing permanent content.

7.6. Uses of Named Pipes

Named pipes can be used for:

- Inter-process communication
- Shell scripting
- Data-processing pipelines
- Connecting independent programs
- Simple producer-consumer designs
- Temporary communication between processes

8. Sockets

The seventh fundamental file type is the **socket**.

Its symbol is:

```
s
```

Example:

```
srwxr-xr-x 1 user user 0 Aug 24 08:50 mysocket
```

The first character is:

```
s
```

8.1. What is a Socket?

A socket is an endpoint for communication between processes.

Sockets are especially important for:

- Networking
- Client/server applications
- Inter-process communication
- Web servers
- Database servers
- Linux system services

8.2. Unix Domain Sockets

Linux supports **Unix domain sockets**, which can appear as filesystem entries.

For example:

```
/run/some-service/socket
```

A Unix domain socket is generally used for communication between processes on the same system.

8.3. Network Sockets

Sockets can also be used for network communication.

For example:

```
TCP
UDP
```

Applications can communicate across machines using network sockets.

A web browser communicating with a web server is an example of socket-based communication.

8.4. Socket Example

A server program might create a Unix socket:

```
/tmp/server.sock
```

A client can connect to this socket.

The server receives requests and sends responses.

The filesystem pathname provides a way for clients to locate the communication endpoint.

8.5. Viewing Sockets

You can use:

```
ss -l
```

or:

```
ss -lx
```

for Unix domain sockets.

You can also use:

```
find /run -type s
```

to find socket filesystem objects under `/run`.

9. Comparing the Seven File Types

The seven types can be summarized as follows:

Symbol	File type	Main purpose	Example
`-`	Regular file	Store ordinary data	`notes.txt`
`d`	Directory	Organize filesystem entries	`/home/user`
`c`	Character device	Stream-oriented device access	`/dev/null`
`b`	Block device	Block-oriented storage access	`/dev/sda`
`l`	Symbolic link	Refer to another pathname	`current -> Documents`
`p`	FIFO	Process communication	`mypipe`
`s`	Socket	Communication endpoint	`/run/service.sock`

---

10. Identifying File Types Using `ls`

One of the easiest methods is:

```
ls -l
```

The first character identifies the type.

Example:

```
-rw-r--r-- file.txt
drwxr-xr-x Documents
lrwxrwxrwx current -> Documents
crw-rw-rw- /dev/null
brw-rw---- /dev/sda
prw-r--r-- mypipe
srwxr-xr-x mysocket
```

Therefore:

```
- = regular file
d = directory
l = symbolic link
c = character device
b = block device
p = FIFO
s = socket
```

11. Using the `file` Command

The `file` command attempts to determine the type/content of a filesystem object.

Example:

```
file notes.txt
```

Possible output:

```
notes.txt: ASCII text
```

For a directory:

```
file Documents
```

may show:

```
Documents: directory
```

For a symbolic link:

```
file current
```

may report that it is a symbolic link and identify its target.

The `file` command is particularly useful for determining the **content format** of regular files, whereas `ls -l` is particularly useful for seeing the filesystem object type.

---

12. Using `stat`

The `stat` command provides detailed filesystem information.

Example:

```
stat notes.txt
```

Output includes information such as:

```
File: notes.txt
Size: 1200
Blocks: ...
IO Block: ...
regular file
Device: ...
Inode: ...
Links: 1
Access: ...
Modify: ...
Change: ...
```

The output identifies the object as:

```
regular file
```

For a directory:

```
stat Documents
```

it identifies:

```
directory
```

For a symlink:

```
stat current
```

it identifies the symbolic link.

---

13. Inodes and File Types

To understand Linux file types more deeply, it is useful to understand **inodes**.

An inode is a filesystem data structure that stores metadata about a filesystem object.

Depending on the filesystem, inode metadata can include:

- File type
- Permissions
- Owner
- Group
- File size
- Timestamps
- Link count
- Pointers/references to data blocks

You can view inode numbers with:

```
ls -li
```

Example:

```
123456 -rw-r--r-- 1 user user 100 notes.txt
```

Here:

```
123456
```

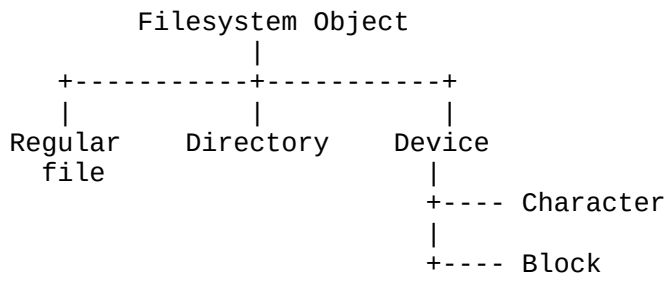
is the inode number.

---

14. File Type and Inode

The inode records the type of filesystem object.

Conceptually:



Other filesystem objects include:

```
Symbolic link
FIFO
Socket
```

This is why Linux can distinguish between:

```
file
directory
device
link
pipe
socket
```

even though many are accessed through file descriptors and filesystem paths.

15. File Descriptors

Linux processes interact with many resources through **file descriptors**.

A file descriptor is a small integer representing an open file-like resource.

The standard descriptors are:

```
0 = stdin
1 = stdout
2 = stderr
```

For example:

```
echo "Hello" > output.txt
```

The shell arranges for standard output to be redirected to a file.

A process can also have file descriptors associated with:

- Regular files
- Pipes
- Sockets
- Devices
- Other kernel-supported objects

This contributes to the power of the Linux “everything is a file” model.

---

16. `/dev` and Device Files

Device files are primarily found under:

```
/dev
```

You can examine them with:

```
ls -l /dev
```

Examples include:

```
/dev/null
/dev/zero
/dev/random
/dev/tty
/dev/sda
```

Some are character devices:

```
crw-rw-rw- /dev/null
```

Some are block devices:

```
brw-rw---- /dev/sda
```

The `/dev` directory is therefore an important place for understanding Linux device files.

Modern Linux systems commonly use `udev` and related mechanisms to dynamically manage device nodes.

---

17. Special Files

The term **special file** is often used for filesystem objects that provide interfaces to devices or communication mechanisms rather than ordinary persistent data.

Examples include:

```
Character devices
Block devices
FIFOs
Sockets
```

The exact terminology can vary depending on context and documentation.

For Linux administration, remembering the seven types and their symbols is more useful than relying only on the phrase “special file.”

---

18. Practical Example: Creating Different File Types

We can create several of these file types ourselves.

Create a regular file:

```
touch regular.txt
```

Create a directory:

```
mkdir mydir
```

Create a symbolic link:

```
ln -s regular.txt mylink
```

Create a FIFO:

```
mkfifo mypipe
```

Sockets are normally created by applications rather than with a simple everyday shell command.

Now:

```
ls -l
```

could show:

```
-rw-r--r-- 1 user user 0 regular.txt
drwxr-xr-x 2 user user 4096 mydir
lrwxrwxrwx 1 user user 11 mylink -> regular.txt
prw-r--r-- 1 user user 0 mypipe
```

19. File Type Testing in Shell Scripts

Linux shell scripts can test file types using the `test` command or `[ ]`.

For example:

```
[ -f file.txt ]
```

checks whether `file.txt` is a regular file.

Directory:

```
[ -d Documents ]
```

Symbolic link:

```
[ -L mylink ]
```

Character device:

```
[ -c /dev/null ]
```

Block device:

```
[ -b /dev/sda ]
```

FIFO:

```
[ -p mypipe ]
```

Socket:

```
[ -S mysocket ]
```

These tests are extremely useful in shell scripting.

20. Important File-Test Operators

Test	Meaning
`-f`	Regular file
`-d`	Directory
`-L`	Symbolic link
`-c`	Character device
`-b`	Block device
`-p`	Named pipe/FIFO
`-S`	Socket
`-e`	Exists
`-r`	Readable
`-w`	Writable
`-x`	Executable/searchable as applicable

Example:

```
if [ -f "$file" ]; then
    echo "Regular file"
fi
```

Another example:

```
if [ -d "$dir" ]; then
    echo "Directory"
fi
```

---

21. A Practical Shell Script

A simple script can determine the type of an object:

```
#!/bin/bash

file="$1"

if [ -f "$file" ]; then
    echo "Regular file"
elif [ -d "$file" ]; then
    echo "Directory"
elif [ -L "$file" ]; then
    echo "Symbolic link"
elif [ -c "$file" ]; then
    echo "Character device"
elif [ -b "$file" ]; then
    echo "Block device"
elif [ -p "$file" ]; then
    echo "Named pipe"
```

```
elif [ -S "$file" ]; then
    echo "Socket"
else
    echo "Unknown or unsupported type"
fi
```

Save it as:

```
checktype.sh
```

Make it executable:

```
chmod +x checktype.sh
```

Run:

```
./checktype.sh notes.txt
```

Output:

```
Regular file
```

---

22. The `find` Command and File Types

The `find` command can search for specific file types.

Regular files:

```
find . -type f
```

Directories:

```
find . -type d
```

Symbolic links:

```
find . -type l
```

Character devices:

```
find /dev -type c
```

Block devices:

```
find /dev -type b
```

FIFOs:

```
find . -type p
```

Sockets:

```
find /run -type s
```

This is very useful for Linux system administration.

---

23. Why Linux Has Seven File Types

The seven file types allow Linux to provide a unified interface to many different resources.

For example:

Regular file

Stores data.

```
notes.txt
```

Directory

Organizes filesystem objects.

```
Documents/
```

Character device

Provides byte/stream-oriented device access.

```
/dev/null
```

Block device

Provides block-oriented storage access.

```
/dev/sda
```

Symbolic link

Provides an alternate pathname.

```
current -> Documents
```

FIFO

Provides one-way/stream-like process communication.

```
mypipe
```

Socket

Provides process/network communication.

```
server.sock
```

Together, these types provide a flexible interface between applications and the Linux kernel.

---

24. Difference Between File Extension and File Type

A common beginner mistake is to think:

```
.txt = text file type
.jpg = image file type
.mkv = video file type
```

These are **not Linux filesystem types**.

For Linux filesystem purposes, they can all be:

```
-
```

meaning:

```
regular file
```

For example:

```
movie.mkv
photo.jpg
notes.txt
report.pdf
```

could all show:

```
-rw-r--r--
```

The extension helps applications and users identify the intended content format, but the filesystem's object type is still a regular file.

25. Seven File Types Example

Consider this directory:

```
/home/user/example
```

It contains:

```
document.txt
Documents/
shortcut
mypipe
```

And the system has:

```
/dev/null
/dev/sda
```

The corresponding types might be:

```
document.txt       → Regular file
Documents/         → Directory
shortcut           → Symbolic link
mypipe             → FIFO
/dev/null          → Character device
/dev/sda           → Block device
server.sock        → Socket
```

Therefore all seven fundamental Linux filesystem object types are represented.

---

26. Major Differences

Feature	Regular	Directory	Char	Block	Symlink	FIFO	Socket
Symbol	-	d	c	b	l	p	s
Stores ordinary data	Yes	No	No	No	No	No	No
Organizes entries	No	Yes	No	No	No	No	No
Device interface	No	No	Yes	Yes	No	No	No
Points to another path	No	No	No	No	Yes	No	No
IPC	Sometimes	No	Possible	No	No	Yes	Yes
Networking	No	No	No	No	No	No	Yes

27. Important Commands to Remember

Display file types

```
ls -l
```

Display hidden files

```
ls -la
```

Display inode numbers

```
ls -li
```

Determine file information

```
file filename
```

Display detailed metadata

```
stat filename
```

Create regular file

```
touch file
```

Create directory

```
mkdir directory
```

Create symbolic link

```
ln -s target link
```

Create FIFO

```
mkfifo pipe
```

Find regular files

```
find . -type f
```

Find directories

```
find . -type d
```

Find symbolic links

```
find . -type l
```

Find character devices

```
find /dev -type c
```

Find block devices

```
find /dev -type b
```

Find FIFOs

```
find . -type p
```

Find sockets

```
find /run -type s
```

---

28. Common Beginner Mistakes

Mistake 1: Thinking directories are not files

In Linux filesystem terminology, a directory is itself a filesystem object with a file type.

Its type is:

```
d
```

Mistake 2: Thinking `.txt` is a Linux file type

`.txt` describes the filename/content convention.

The filesystem type is:

```
-
```

for a normal text file.

Mistake 3: Confusing a symlink with a copy

Suppose:

```
ln -s original.txt link.txt
```

This does **not** create another copy of the contents.

It creates a reference to the pathname.

Mistake 4: Confusing a device file with the actual hardware

For example:

```
/dev/sda
```

is a device node representing an interface to a block device.

It is not itself the physical hard disk sitting outside the computer.

Mistake 5: Thinking a FIFO is a normal storage file

A FIFO is primarily for communication between processes.

It should not be treated like a normal text file.

Mistake 6: Confusing sockets with regular files

A socket may have a pathname, but its purpose is communication rather than ordinary persistent file storage.

29. Easy Way to Memorize the Seven Types

Remember this table:

```
-  = Regular file
d  = Directory
c  = Character device
b  = Block device
l  = Link
p  = Pipe
s  = Socket
```

A useful memory sequence is:

```
- d c b l p s
```

Or:

```
Regular
Directory
Character
Block
Link
Pipe
Socket
```

When you run:

```
ls -l
```

look at the first character.

30. Conclusion

Linux has seven fundamental filesystem object types:

```
1. Regular file       -
2. Directory          d
3. Character device   c
4. Block device       b
5. Symbolic link      l
6. FIFO / named pipe  p
7. Socket             s
```

The most commonly encountered are **regular files, directories, and symbolic links**. Linux administrators also frequently work with **character and block devices** under `/dev`, while system programmers and advanced administrators encounter **FIFOs and sockets** for inter-process communication.

The single most important command for recognizing these types is:

```
ls -l
```

because the first character immediately tells you what kind of filesystem object you are looking at.

For example:

```
-rw-r--r--    → Regular file
drwxr-xr-x    → Directory
lrwxrwxrwx    → Symbolic link
crw-rw-rw-    → Character device
brw-rw----    → Block device
prw-r--r--    → FIFO
srwxr-xr-x    → Socket
```

Once you understand these seven types, commands such as `ls`, `find`, `stat`, `file`, `ln`, `mkfifo`, `mount`, and shell file tests become much easier to understand. This concept is also fundamental for Linux system administration because it explains how Linux presents **data, directories, hardware, communication channels, and alternate paths through one unified filesystem interface**.