Table of Contents

$ file

Beginner

Determine the type of a file

The file command determines the type of a file by examining its contents rather than its extension. It uses magic number databases, filesystem tests, and language tests to identify files.

file is essential because Linux does not rely on file extensions for type determination. A file named photo....

Options & Flags

`-i`	Output MIME type instead of description
`-b`	Brief mode - omit filename from output
`-z`	Look inside compressed files
`-L`	Follow symbolic links
`-f`	Read filenames from a file
`-s`	Read block/character special files

Practical Examples

Example: Identify file type

```
$ file report.pdf
report.pdf: PDF document, version 1.7
```

Determines the actual type of the file by examining its contents.

Example: Check MIME type

```
$ file -i image.png
image.png: image/png; charset=binary
```

Shows the MIME type, useful for web servers and content negotiation.

Example: Check multiple files

```
$ file *
```

Identifies the type of every file in the current directory.

Example: Identify executable type

```
$ file /usr/bin/ls
/usr/bin/ls: ELF 64-bit LSB pie executable, x86-64, dynamically linked
```

Shows whether a binary is 32/64-bit, statically/dynamically linked, etc.

Example: Check encoding

```
$ file -i data.csv
data.csv: text/plain; charset=utf-8
```

Shows the character encoding of a text file (UTF-8, ASCII, etc.).

Tips & Best Practices

Pro Tip: Verify before extracting: Always check archives with file before extracting. A file named archive.tar.gz might not actually be a gzip file.

Note: MIME types for web servers: Use file -i to get MIME types when configuring web server content types or email attachments.

Warning: Not foolproof: file uses heuristics and can occasionally misidentify files. It reads file headers, not the entire content.

$ find

Intermediate

Search for files and directories in a directory hierarchy

The find command searches for files and directories in a directory hierarchy based on various criteria including name, type, size, permissions, modification time, and more. It is one of the most powerful and versatile commands in Linux.

find traverses directory trees recursively, testing each fi...

Options & Flags

`-name`	Search by filename (case-sensitive, supports wildcards)
`-iname`	Case-insensitive name search
`-type`	Filter by type: f(file), d(directory), l(symlink)
`-size`	Filter by file size (+larger, -smaller)
`-mtime`	Filter by modification time in days
`-perm`	Filter by permissions
`-exec`	Execute command on each result
`-maxdepth`	Limit directory traversal depth
`-empty`	Find empty files or directories
`-newer`	Find files newer than reference file

Practical Examples

Example: Find all PHP files

```
$ find src/ -name '*.php' -type f
```

Searches recursively for PHP files in the src directory.

Example: Find large files

```
$ find /var -type f -size +100M 2>/dev/null
```

Finds files larger than 100MB, suppressing permission errors.

Example: Find recently modified files

```
$ find /etc -type f -mtime -7
```

Finds files modified in the last 7 days.

Example: Find and delete old temp files

```
$ find /tmp -type f -mtime +30 -delete
```

Deletes files in /tmp older than 30 days.

Example: Find files with specific permissions

```
$ find / -perm -4000 -type f 2>/dev/null
```

Finds all SUID files on the system (security audit).

Tips & Best Practices

Warning: Use -print0 with xargs: Always use find -print0 | xargs -0 for filenames with spaces. Without it, files like 'my file.txt' will break.

Pro Tip: -exec + vs \;: Use -exec cmd {} + instead of -exec cmd {} \; to batch arguments like xargs, significantly improving performance.

Note: find vs locate: find searches the live filesystem (slower but current). locate uses a pre-built database (faster but may be stale). Run updatedb to refresh locate's database.

$ locate

Beginner

Find files by name using a pre-built database

The locate command finds files by name using a pre-built database, making it orders of magnitude faster than find for simple name searches. It searches a database of file paths rather than traversing the filesystem.

The database is typically updated daily by a cron job running updatedb, or can b...

Options & Flags

`-i`	Case-insensitive search
`-c`	Print only the count of matches
`-l`	Limit output to N results
`-r`	Use POSIX regex instead of glob pattern
`-e`	Print only existing files (skip deleted)
`-b`	Match only the basename (filename without path)

Practical Examples

Example: Find a file quickly

```
$ locate nginx.conf
/etc/nginx/nginx.conf\n/etc/nginx/nginx.conf.bak
```

Instantly finds all paths containing nginx.conf.

Example: Case-insensitive search

```
$ locate -i readme
```

Finds README, readme, Readme, etc.

Example: Count matching files

```
$ locate -c "*.php"
2847
```

Shows how many PHP files exist in the database.

Example: Update the database

```
$ sudo updatedb
```

Refreshes the locate database to include recently created files.

Example: Find only in basename

```
$ locate -b "\nginx.conf"
/etc/nginx/nginx.conf
```

Matches only files named exactly nginx.conf (not paths containing it).

Tips & Best Practices

Warning: Database may be stale: locate uses a pre-built database. Newly created files will not appear until updatedb runs. Use find for real-time searches.

Pro Tip: Fast file discovery: locate is 100-1000x faster than find for simple name searches. Use it as your first tool, then switch to find if you need freshness or complex criteria.

Note: mlocate vs plocate: Modern Linux uses plocate (faster) or mlocate. Both provide the locate command with compatible interfaces.

$ stat

Intermediate

Display detailed file or filesystem status

The stat command displays detailed information about files and filesystems, including size, permissions, ownership, timestamps, inode number, and link count. It provides far more detail than ls.

stat shows three timestamps: access time (atime), modification time (mtime), and change time (ctime)....

Options & Flags

`-c`	Use custom format string for output
`-f`	Display filesystem information
`-L`	Follow symbolic links
`-t`	Terse (machine-readable) output
`%s`	Format: file size in bytes
`%U/%G`	Format: owner/group name
`%a/%A`	Format: permissions (octal/human)

Practical Examples

Example: Full file information

```
$ stat /etc/passwd
```

Shows complete metadata including size, permissions, timestamps, and inode.

Example: Get file size in bytes

```
$ stat -c '%s' largefile.iso
4294967296
```

Outputs just the file size in bytes - useful in scripts.

Example: Show permissions in octal

```
$ stat -c '%a %n' /etc/shadow
640 /etc/shadow
```

Shows octal permissions and filename for security auditing.

Example: Show all timestamps

```
$ stat -c 'Access: %x Modify: %y Change: %z' file.txt
```

Displays all three timestamps in readable format.

Example: Filesystem information

```
$ stat -f /
```

Shows filesystem type, total/free space, block size, and inode info.

Tips & Best Practices

Pro Tip: Useful format strings: Common formats: %s (size), %a (octal perms), %U (owner), %G (group), %y (mtime), %i (inode). Combine them: stat -c '%a %U %s %y' file

Note: Three timestamps explained: mtime = content changed, ctime = metadata changed (permissions, owner), atime = file was read. ctime cannot be manually set.

Warning: ctime is not creation time: In Linux, ctime is change time (metadata change), NOT creation time. Birth time is only available on some filesystems (ext4: stat shows 'Birth').

$ tree

Beginner

List directory contents in a tree-like format

The tree command displays directory structures as an indented tree diagram, showing files and subdirectories in a visually organized hierarchy. It provides a much clearer picture of directory organization than ls.

tree recursively lists all files and directories, showing the parent-child relatio...

Options & Flags

`-L`	Limit recursion depth
`-d`	List directories only
`-a`	Show hidden files (starting with .)
`-h`	Print sizes in human-readable format
`-P`	Show only files matching pattern
`-I`	Exclude files matching pattern
`--dirsfirst`	List directories before files
`-J`	Output as JSON

Practical Examples

Example: View project structure

```
$ tree -L 2 src/
src/
??? controllers/
?   ??? HomeController.php
?   ??? BookController.php
```

Shows the first 2 levels of the src directory.

Example: Directories only

```
$ tree -d -L 3 /var/www
```

Shows only the directory structure without files.

Example: Exclude common clutter

```
$ tree -I 'node_modules|.git|vendor|__pycache__' .
```

Shows the project tree excluding common large directories.

Example: Show sizes

```
$ tree -h --du /var/log
```

Shows file sizes and directory totals in human-readable format.

Example: Filter by file type

```
$ tree -P '*.py' --prune src/
```

Shows only Python files, pruning empty directories.

Tips & Best Practices

Pro Tip: Exclude patterns for clean output: Use -I to exclude noise: tree -I 'node_modules|.git|vendor|dist|build' shows only your actual code structure.

Note: JSON output for tooling: tree -J outputs a JSON representation of the directory structure, useful for programmatic analysis and documentation generation.

Warning: Can be slow on large directories: Without -L depth limit, tree will recurse through everything including node_modules (which can have 50,000+ files). Always use -L or -I.

$ whereis

Beginner

Locate the binary, source, and manual page files for a command

The whereis command locates binary, source, and manual page files for a command. Unlike which (which only searches PATH), whereis searches standard system directories for all components of a program.

whereis looks in directories like /usr/bin, /usr/sbin, /usr/share/man, and other standard locati...

Options & Flags

`-b`	Search only for binaries
`-m`	Search only for man pages
`-s`	Search only for source files
`-u`	Find commands with unusual (missing) entries
`-B`	Define binary search path
`-l`	List directories that will be searched

Practical Examples

Example: Find all components

```
$ whereis nginx
nginx: /usr/sbin/nginx /usr/share/nginx /usr/share/man/man8/nginx.8.gz
```

Shows binary, man page, and config file locations.

Example: Find binary only

```
$ whereis -b python3
python3: /usr/bin/python3
```

Shows only the executable location.

Example: Find man pages

```
$ whereis -m awk
awk: /usr/share/man/man1/awk.1.gz
```

Shows only the manual page locations.

Example: List search directories

```
$ whereis -l
```

Lists all directories that whereis will search.

Example: Check multiple commands

```
$ whereis gcc make cmake
```

Shows binary, source, and man page locations for each command.

Tips & Best Practices

Note: whereis vs which: which searches PATH and finds only executables. whereis searches standard directories and finds binaries, man pages, and source files.

Pro Tip: Quick documentation check: Use whereis -m command to quickly check if man pages exist for a command before trying to read them.

Warning: Standard locations only: whereis only searches standard system directories. It may miss programs installed in non-standard locations like /opt or ~/.local/bin.

$ which

Beginner

Show the full path of shell commands

The which command locates the executable file associated with a command by searching the directories listed in the PATH environment variable. It shows which version of a program would run when you type its name.

which is essential for troubleshooting when multiple versions of a program are insta...

Options & Flags

`-a`	Show all matching executables in PATH, not just first
`(no flags)`	Show the first matching executable
`multiple`	Check multiple commands at once
`exit code`	Returns 0 if found, 1 if not found
`-s`	Silent mode (some implementations)
`in scripts`	Use in conditionals for command checks

Practical Examples

Example: Find command location

```
$ which nginx
/usr/sbin/nginx
```

Shows the full path to the nginx executable.

Example: Find all versions

```
$ which -a python
/usr/bin/python
/usr/local/bin/python
```

Shows all python executables found in PATH, in order of precedence.

Example: Check multiple commands

```
$ which python3 pip3 node npm
```

Shows paths for each command or an error if not found.

Example: Verify command exists in script

```
$ which docker > /dev/null 2>&1 && docker --version
```

Only runs docker --version if docker is found in PATH.

Example: Find compiler location

```
$ which gcc g++ make
/usr/bin/gcc\n/usr/bin/g++\n/usr/bin/make
```

Locates build tools to verify development environment.

Tips & Best Practices

Pro Tip: Use type instead for built-ins: which cannot find shell built-ins (cd, echo, etc.) or aliases. Use type -a command to find everything including built-ins and aliases.

Note: Cron jobs need full paths: Cron has a minimal PATH. Use which to find the full path of commands for cron: which php returns /usr/bin/php.

Warning: Aliases not shown: which shows executables on disk, not shell aliases. If you aliased python to python3, which python still shows the original executable.