++101 Linux Commands

BOBBY ILIEV

Table of Contents

101 Linux commands Open-source eBook

This is an open-source eBook with 101 Linux commands that everyone should know. No matter if you are a DevOps/SysOps engineer, developer, or just a Linux enthusiast, you will most likely have to use the terminal at some point in your career.

Hacktoberfest

4

This eBook is made possible thanks to Hacktoberfest and the open source community!

About me

My name is Bobby Iliev, and I have been working as a Linux DevOps Engineer since 2014. I am an avid Linux lover and supporter of the open-source movement philosophy. I am always doing that which I cannot do in order that I may learn how to do it, and I believe in sharing knowledge.

I think it's essential always to keep professional and surround yourself with good people, work hard, and be nice to everyone. You have to perform at a consistently higher level than others. That's the mark of a true professional.

For more information, please visit my blog at https://bobbyiliev.com, follow me on Twitter @bobbyiliev_ and YouTube.

DigitalOcean

DigitalOcean is a cloud services platform delivering the simplicity developers love and businesses trust to run production applications at scale.

It provides highly available, secure, and scalable compute, storage, and networking solutions that help developers build great software faster.

Founded in 2012 with offices in New York and Cambridge, MA, DigitalOcean offers transparent and affordable pricing, an elegant user interface, and one of the largest libraries of open source resources available.

For more information, please visit https://www.digitalocean.com or follow @digitalocean on Twitter.

If you are new to DigitalOcean, you can get a free $200 credit and spin up your own servers via this referral link here:

Free $200 Credit For DigitalOcean

DevDojo

The DevDojo is a resource to learn all things web development and web design. Learn on your lunch break or wake up and enjoy a cup of coffee with us to learn something new.

Join this developer community, and we can all learn together, build together, and grow together.

Join DevDojo

For more information, please visit https://www.devdojo.com or follow @thedevdojo on Twitter.

Ebook Generation Tool

This ebook was generated by Ibis Next developed by Roberto Butti, forked from the original Ibis by Mohamed Said.

Ibis Next is a PHP tool that helps you write eBooks in markdown and supports generating PDF, EPUB, and HTML formats.

Book Cover

8

The cover for this ebook was created by Suhail Kakar.

License

MIT License

Copyright (c) 2020 Bobby Iliev

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The `ls` command

The `ls` command lets you see the files and directories inside a specific directory *(current working directory by default)*. It normally lists the files and directories in ascending alphabetical order.

Examples:

1. To show the files inside your current working directory:

```
ls
```

2. To show the files and directory inside a specific Directory:

```
ls {Directory_Path}
```

3. List all files including hidden ones in long format with human-readable sizes:

```
ls -lah
```

4. Sort files by modification time, newest first:

```
ls -lt
```

5. List files by size, largest first:

```
ls -lhS
```

6. Show only directories in the current path:

```
ls -d */
```

7. List all text files with details:

```
ls -lh *.txt
```

8. Recursively list all files in subdirectories:

```
ls -R
```

Syntax:

```
ls [-OPTION] [DIRECTORY_PATH]
```

Interactive training

In this interactive tutorial, you will learn the different ways to use the `ls` command:

[The ls command by Tony](#)

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
`-l`	-	Show results in long format
`-S`	-	Sort results by file size
`-t`	-	Sort results by modification time
`-r`	`--reverse`	Show files and directories in reverse order *(descending alphabetical order)*
`-a`	`--all`	Show all files, including hidden files *(file names which begin with a period `.`)*
`-la`	-	Show long format files and directories including hidden files
`-lh`	-	list long format files and directories with readable size
`-A`	`--almost-all`	Shows all like `-a` but without showing `.`(current working directory) and `..` (parent directory)
`-d`	`--directory`	Instead of listing the files and directories inside the directory, it shows any information about the directory itself, it can be used with `-l` to show long formatted information
`-F`	`--classify`	Appends an indicator character to the end of each listed name, as an example: `/` character is appended after each directory name listed
`-h`	`--human-readable`	like `-l` but displays file size in human-readable unit not in bytes
`-i`	`--inode`	Display inode number for each file
`-R`	`--recursive`	List subdirectories recursively
`-1`	-	List one file per line

Short Flag	Long Flag	Description
`-c`	-	Sort by change time (when file metadata was last changed)
`-u`	-	Sort by access time (when file was last accessed)
`-X`	-	Sort alphabetically by file extension
-	`--color=auto`	Colorize output to distinguish file types
`-g`	-	Like `-l` but without showing owner
`-o`	-	Like `-l` but without showing group
`-C`	-	List entries by columns
`-s`	`--size`	Print the allocated size of each file
-	`--group-directories-first`	List directories before files

File Type Indicators:

When using the `-F` or `--classify` flag, `ls` appends special characters to filenames to indicate their type:

Indicator	File Type	Example
/	Directory	`docs/`
*	Executable file	`script.sh*`
@	Symbolic link	`link@`
|	FIFO (named pipe)	`pipe|`
=	Socket	`socket=`
No indicator	Regular file	`file.txt`

Example:

```
ls -F
documents/  script.sh*  link@  data.txt  pipe|  socket=
```

SELinux Support on Red Hat-Based Systems:

On Red Hat-based distributions (RHEL, CentOS, Fedora, Rocky Linux, AlmaLinux) that use SELinux, the `ls` command provides additional options to display SELinux security context information:

Short Flag	Long Flag	Description
-Z	--context	Display SELinux security context for files and directories
-lZ	-	Show long format with SELinux security context

Example Output:

```
ls -Z
unconfined_u:object_r:user_home_t:s0 file.txt
unconfined_u:object_r:user_home_t:s0 directory
```

```
ls -lZ
-rw-rw-r--. 1 user user unconfined_u:object_r:user_home_t:s0
1234 Jan 15 10:30 file.txt
drwxrwxr-x. 2 user user unconfined_u:object_r:user_home_t:s0
4096 Jan 15 10:25 directory
```

The SELinux context format is: `user:role:type:level`

Note: The `-Z` option is only functional on systems with SELinux enabled. On non-SELinux systems, this option may not be available or will show no additional information.

Understanding Long Format Output:

When using `ls -l`, each line displays detailed information about a file or directory:

```
-rw-r--r-- 1 user group 1234 Jan 15 10:30 file.txt
```

Column breakdown:

1. **File Type and Permissions** (`-rw-r--r--`):

   - First character: File type (`-` = regular file, `d` = directory, `l` = symlink, `b` = block device, `c` = character device, `p` = pipe, `s` = socket)
   - Next 9 characters: Permissions in three groups (owner, group, others)
     - `r` = read, `w` = write, `x` = execute, `-` = no permission

2. **Link Count** (`1`): Number of hard links to the file

3. **Owner** (`user`): Username of the file owner

4. **Group** (`group`): Group name that owns the file

5. **Size** (`1234`): File size in bytes (use `-h` flag for human-readable format like `1.2K`, `3.4M`)

6. **Modification Time** (`Jan 15 10:30`): Date and time the file was last modified

7. **Filename** (`file.txt`): Name of the file or directory

Example with human-readable sizes:

```
ls -lh
drwxr-xr-x 2 user group 4.0K Jan 15 10:25 documents
-rwxr-xr-x 1 user group 2.3M Jan 14 09:15 script.sh
-rw-r--r-- 1 user group  15K Jan 16 14:30 data.csv
```

Using Wildcards and Patterns:

The `ls` command supports wildcards for pattern matching:

Wildcard	Description	Example	Matches
*	Matches any number of characters	`ls *.txt`	All files ending with `.txt`
?	Matches exactly one character	`ls file?.txt`	`file1.txt`, `file2.txt`, but not `file10.txt`
[]	Matches any character within brackets	`ls file[123].txt`	`file1.txt`, `file2.txt`, `file3.txt`
[!]	Matches any character NOT in brackets	`ls file[!3].txt`	`file1.txt`, `file2.txt`, but not `file3.txt`
{}	Matches any of the comma-separated patterns	`ls *.{jpg,png,gif}`	All image files with these extensions

Examples:

```
# List all PDF and DOCX files
ls *.{pdf,docx}

# List files starting with 'test' followed by a single digit
ls test?.log

# List all files starting with uppercase letters
ls [A-Z]*

# List all hidden configuration files
ls .config*
```

SELinux Support on Red Hat-Based Systems:

On Red Hat-based distributions (RHEL, CentOS, Fedora, Rocky Linux, AlmaLinux) that use SELinux, the `ls` command provides additional options to display SELinux security context information:

Short Flag	Long Flag	Description
-Z	--context	Display SELinux security context for files and directories
-lZ	-	Show long format with SELinux security context

Example Output:

```
ls -Z
unconfined_u:object_r:user_home_t:s0 file.txt
unconfined_u:object_r:user_home_t:s0 directory
```

```
ls -lZ
-rw-rw-r--. 1 user user unconfined_u:object_r:user_home_t:s0
1234 Jan 15 10:30 file.txt
drwxrwxr-x. 2 user user unconfined_u:object_r:user_home_t:s0
4096 Jan 15 10:25 directory
```

The SELinux context format is: `user:role:type:level`

Note: The `-Z` option is only functional on systems with SELinux enabled. On non-SELinux systems, this option may not be available or will show no additional information.

Setting Persistent Options:

Customizing command behavior in Linux is easy using the `alias` command. To make these changes permanent, follow these steps:

1. **Create the Alias**: Define your alias with the desired options. For example, to enhance the `ls` command:

   ```
   alias ls="ls --color=auto -lh"
   ```

2. **Persistence**: This alias is effective only for the current session. To make it permanent, add the alias to your shell's configuration file:

   - **Bash**: Append the alias to `~/.bashrc`:

     ```
     echo 'alias ls="ls --color=auto -lh"' >> ~/.bashrc
     source ~/.bashrc
     ```

3. **Verification**: Open a new terminal session, and the `ls` command will display files as configured.

Performance Tips:

1. Avoid recursive listing on large directories:

- The `-R` flag can be slow on directories with many subdirectories and files
- Consider using `find` command for more control: `find /path -type f`

2. Disable color output in scripts:

- Use `--color=never` when piping output or in scripts to improve performance
- Example: `ls --color=never | grep pattern`

3. Limit output for large directories:

- Combine with `head` to see only first few entries: `ls -1 | head -n 20`
- Or use `ls -1 | wc -l` to just count files without displaying them

4. Use specific paths instead of wildcards when possible:

- `ls /var/log/syslog` is faster than `ls /var/log/sys*` when you know the exact filename

Common Use Cases:

1. Find the largest files in a directory:

```
ls -lhS | head -n 10
```

2. List only directories:

```
ls -d */
```

Or with full details:

```
ls -ld */
```

3. Count files in a directory:

```
ls -1 | wc -l
```

4. List files modified in the last 24 hours:

```
ls -lt | head
```

5. Show files sorted by extension:

```
ls -lX
```

6. List files with their inode numbers (useful for debugging):

```
ls -li
```

7. Display directory contents one per line (useful for scripting):

```
ls -1
```
                                        21

8. Combine multiple sort options (size + reverse):

```
ls -lhSr
```

The cd command

The cd command is used to change the current working directory *(i.e., the directory in which the current user is working)*. The "cd" stands for "**c**hange **d**irectory" and it is one of the most frequently used commands in the Linux terminal.

The cd command is often combined with the ls command (see chapter 1) when navigating through a system. You can also press the TAB key to auto-complete directory names, or press TAB twice to list available directories in the current location.

Syntax:

```
cd [OPTIONS] [directory]
```

Basic Examples:

1. **Change to a specific directory:**

```
cd /path/to/directory
```

2. **Change to your home directory:**

```
cd ~
```

OR simply:

```
cd
```

3. **Change to the previous directory:**

```
cd -
```

This will also print the absolute path of the previous directory.

4. **Change to the system's root directory:**

```
cd /
```

5. **Move up one directory level (parent directory):**

```
cd ..
```

6. **Move up multiple directory levels:**

```
cd ../../..
```

This example moves up three levels.

Practical Examples:

Using relative paths:

```
cd Documents/Projects/MyApp
```

Using absolute paths:

```
cd /usr/local/bin
```

Combining with home directory shortcut:

```
cd ~/Downloads
```

Navigate to a directory with spaces in the name:

```
cd "My Documents"
```

OR

```
cd My\ Documents
```

Switch between two directories:

```
cd /var/log
cd /etc
cd -  # Returns to /var/log
cd -  # Returns to /etc
```

Additional Flags and Their Functionalities

Short flag	Long flag	Description
-L	-	Follow symbolic links (default behavior). The cd command will follow symlinks and update the working directory to the target location.
-P	-	Use the physical directory structure without following symbolic links. Shows the actual path instead of the symlink path.

Example of -L vs -P with symbolic links:

```
# Assume /var/www is a symlink to /home/user/web
cd -L /var/www      # Working directory shows as /var/www
pwd                 # Outputs: /var/www

cd -P /var/www      # Working directory resolves to actual
path
pwd                 # Outputs: /home/user/web
```

Common Errors and Troubleshooting

Permission denied:

```
cd /root
# bash: cd: /root: Permission denied
```

Solution: You need appropriate permissions to access the directory. Try using sudo if necessary.

No such file or directory:

```
cd /invalid/path
# bash: cd: /invalid/path: No such file or directory
```

Solution: Verify the path exists using `ls` or check for typos. Remember that paths are case-sensitive.

Not a directory:

```
cd /etc/passwd
# bash: cd: /etc/passwd: Not a directory
```

Solution: Ensure you're navigating to a directory, not a file.

Important Notes:

- **Case sensitivity:** Linux file systems are case-sensitive. `cd Documents` is different from `cd documents`.
- **Special characters:** Directory names with spaces or special characters need to be quoted or escaped.
- **The `cd` command is a shell built-in:** It's not an external program, which is why it can change the shell's current directory.
- **No output on success:** By default, `cd` produces no output when successful (except `cd -` which prints the new path).

The cat command

The cat command allows us to create single or multiple files, to view the content of a file or to concatenate files and redirect the output to the terminal or files.

The "cat" stands for 'concatenate.' and it's one of the most frequently used commands in the Linux terminal.

Examples of uses:

1. To display the content of a file in terminal:

```
cat <specified_file_name>
```

2. To display the content of multiple files in terminal:

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

3. To create a file with the cat command:

```
cat > file_name
```

4. To display all files in current directory with the same filetype:

```
cat *.<filetype>
```

5. To display the content of all the files in current directory:

```
cat *
```

6. To put the output of a given file into another file:

```
cat old_file_name > new_file_name
```

7. Use cat command with `more` and `less` options:

```
cat filename | more
cat filename | less
```

8. Append the contents of file1 to file2:

```
cat file1 >> file2
```

9. To concatenate two files together in a new file:

```
cat file1_name file2_name merge_file_name
```

10. Some implementations of cat, with option -n, it's possible to show line numbers:

```
cat -n file1_name file2_name > new_numbered_file_name
```

Syntax:

```
cat [OPTION] [FILE]...
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-A	--show-all	equivalent to -vET
-b	--number-nonblank	number nonempty output lines, overrides -n
-e	-	equivalent to -vE
-T	-	Display tab separated lines in file opened with cat command.
-E	-	To show $ at the end of each file.
-E	-	Display file with line numbers.
-n	--number	number all output lines
-s	--squeeze-blank	suppress repeated empty output lines
-u	-	(ignored)
-v	--show-nonprinting	use ^ and M- notation, except for LFD and TAB
-	--help	display this help and exit
-	--version	output version information and exit

The `tac` command

`tac` is a Linux command that allows you to view files line-by-line, beginning from the last line. (tac doesn't reverse the contents of each individual line, only the order in which the lines are presented.) It is named by analogy with `cat`.

Examples of uses:

1. To display the content of a file in terminal:

```
tac <specified_file_name>
```

2. This option attaches the separator before instead of after.

```
tac -b concat_file_name tac_example_file_name
```

3. This option will interpret the separator as a regular expression.

```
tac -r concat_file_name tac_example_file_name
```

4. This option uses STRING as the separator instead of newline.

```
tac -s concat_file_name tac_example_file_name
```

5. This option will display the help text and exit.

```
tac --help
```

6. This option will give the version information and exit.

```
tac --version
```

Syntax:

```
tac [OPTION]... [FILE]...
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-b	--before	attach the separator before instead of after
-r	--regex	interpret the separator as a regular expression
-s	--separator=STRING	use STRING as the separator instead of newline
-	--help	display this help and exit
-	--version	output version information and exit

The head command

The head command prints the first ten lines of a file.

Example:

```
head filename.txt
```

Syntax:

```
head [OPTION] [FILENAME]
```

Get a specific number of lines:

Use the -n option with a number (should be an integer) of lines to display.

Example:

```
head -n 10 foo.txt
```

This command will display the first ten lines of the file foo.txt.

Syntax:

```
head -n <number> foo.txt
```

Additional Flags and their Functionalities

Short Flag	Long Flag	Description
`-c`	`--bytes=[-]NUM`	Print the first NUM bytes of each file; with the leading '-', print all but the last NUM bytes of each file
`-n`	`--lines=[-]NUM`	Print the first NUM lines instead of the first 10; with the leading '-', print all but the last NUM lines of each file
`-q`	`--quiet, --silent`	Never print headers giving file names
`-v`	`--verbose`	Always print headers giving file names
`-z`	`--zero-terminated`	Line delimiter is NUL, not newline
	`--help`	Display this help and exit
	`--version`	Output version information and exit

The `tail` command

The `tail` command prints the last ten lines of a file.

Example:

```
tail filename.txt
```

Syntax:

```
tail [OPTION] [FILENAME]
```

Common Use Cases

Get a specific number of lines:

Use the `-n` option with a number (should be an integer) of lines to display.

Example:

```
tail -n 10 foo.txt
```

This command will display the last ten lines of the file `foo.txt`.

Monitor new entry on files in real-time

It is possible to let tail output any new line added to the file you are looking into. So, if a new line is written to the file, it will immediately be shown in your output. This can be done using the `--follow` or `-f` option. This is especially useful for monitoring log files.

Example:

```
tail -f foo.txt
```

Press `Ctrl+C` to stop following.

Combine options for log monitoring

```
tail -n 100 -f app.log        # Show last 100 lines, then follow
tail -f -s 2 slow.log         # Follow with 2-second refresh
interval
```

Follow multiple files simultaneously

```
tail -f /var/log/nginx/access.log /var/log/nginx/error.log
```

Display specific byte ranges

```
tail -c 100 file.txt          # Last 100 bytes
tail -c +50 file.txt          # From byte 50 to end
```

Follow logs even after rotation

```
tail -F /var/log/app.log
```

The -F option is crucial for monitoring logs managed by logrotate.

Skip header lines

```
tail -n +2 data.csv           # Start from line 2 (skip header)
tail -n +11 file.txt          # Start from line 11 onwards
```

Quiet mode for multiple files

```
tail -q -n 5 *.log          # No filename headers
```

Additional Flags and their Functionalities

Short Flag	Long Flag	Description
-c	--bytes=[+]NUM	Output the last NUM bytes; or use -c +NUM to output starting with byte NUM of each file
-f	--follow[={name|descriptor}]	Output appended data as the file grows; an absent option argument means 'descriptor'
-F		Same as --follow=name --retry
-n	--lines=[+]NUM	Output the last NUM lines, instead of the last 10; or use -n +NUM to output starting with line NUM
	--max-unchanged-stats=N	with --follow=name, reopen a FILE which has not changed size after N (default 5) iterations to see if it has been unlinked or rename (this is the usual case of rotated log files); with inotify, this option is rarely useful
	--pid=PID	with -f, terminate after process ID, PID dies
-q	--quiet, --silent	Never output headers giving file names
	--retry	keep trying to open a file if it is inaccessible

Short Flag	Long Flag	Description
-s	--sleep-interval=N	With -f, sleep for approximately N seconds (default 1.0) between iterations; with inotify and --pid=P, check process P at least once every N seconds
-v	--verbose	Always output headers giving file names
-z	--zero-terminated	Line delimiter is NUL, not newline
	--help	Display this help and exit
	--version	Output version information and exit

Tips

39

- Use `-F` instead of `-f` for production log monitoring
- Combine with `grep`, `awk`, or `sed` for filtered monitoring
- For large files, `tail` is much faster than opening in an editor
- Use `-n` `+N` to start from a specific line (useful for skipping headers)

The pwd command

The pwd stands for Print Working Directory. It prints the path of the current working directory, starting from the root.

Example:

```
pwd
```

The output would be your current directory:

```
/home/your_user/some_directory
```

Syntax:

```
pwd [OPTION]
```

Tip: You can also check this by printing out the $PWD variable:

```
echo $PWD
```

The output would be the same as of the pwd command.

Options:

Short Flag	Long Flag	Description
-L	--logical	If the environment variable $PWD contains an absolute name of the current directory with no "." or ".." components, then output those contents, even if they contain symbolic links. Otherwise, fall back to default (-P) behavior.
-P	--physical	Print a fully resolved name for the current directory, where all components of the name are actual directory names, and not symbolic links.
	--help	Display a help message, and exit.
	--version	Display version information, and exit.

By default, `pwd'` `behaves as if` -L' were specified.

The `touch` Command

The `touch` command modifies a file's timestamps. If the file specified doesn't exist, an empty file with that name is created.

Syntax

```
touch [OPTION]... FILE...
```

Options

Short Flag	Long Flag	Description
-a	-	Change only the access time.
-c	--no-create	Do not create any files.
-d STRING	--date=STRING	Parse *STRING* and use it instead of the current time.
-f	-	(Ignored) This option does nothing but is accepted to provide compatibility with BSD versions of the `touch` command.
-h	--no-dereference	Affect each symbolic link instead of any referenced file (useful only on systems that can change the timestamps of a symlink). This option implies `-c`, nothing is created if the file does not exist.
-m	-	Change only the modification time.
-r=FILE	--reference=FILE	Use this file's times instead of the current time.

Short Flag	Long Flag	Description
`-t STAMP`	-	Use the numeric time *STAMP* instead of the current time. The format of *STAMP* is [[**CC**]**YY**]**MMDDhhmm**[**.ss**].
-	`--time=WORD`	An alternate way to specify which type of time is set (e.g. *access*, *modification*, or *change*). This is equivalent to specifying `-a` or `-m`.

- WORD is `access`, `atime`, or `use`: equivalent to `-a`.
- WORD is `modify` or `mtime`: equivalent to `-m`.

An alternate way to specify what type of time to set (as with **-a** and **-m**).| |

-

|`--help`|Display a help message, and exit.| |

-

|`--version`|Display version information, and exit.|

Examples

1. If **file.txt** exists, set all of its timestamps to the current system time. If **file.txt** doesn't exist, create an empty file with that name.

```
touch file.txt
```

2. If **file.txt** exists, set its times to the current system time. If it does not exist, do nothing.

```
touch -c file.txt
```

3. Change the *access* time of **file.txt**. The *modification* time is not changed. The *change* time is set to the current system time. If **file.txt** does not exist, it is created.

```
touch -a file.txt
```

4. Change the times of file **symboliclink**. If it's a symbolic link, change the times of the symlink, ***NOT*** the times of the referenced file.

```
touch -h symboliclink
```

5. Change the *access* and *modification* times of **file-b.txt** to match the times of **file-a.txt**. The *change* time will be set to the current system time. If **file-b.txt** does not exist, it is not created. Note, **file-a.txt** must already exist in this context.

```
touch -cr file-a.txt file-b.txt
```

6. Set the *access* time and *modification* time of **file.txt** to ***February 1st*** of the current year. The *change* time is set to the current system time.

```
touch -d "1 Feb" file.txt
```

The `cal` Command

The `cal` command displays a formatted calendar in the terminal. If no options are specified, cal displays the current month, with the current day highlighted.

Syntax:

```
cal [general options] [-jy] [[month] year]
```

Options:

Option	Description
`-h`	Don't highlight today's date.
`-m month`	Specify a month to display. The month specifier is a full month name (e.g., February), a month abbreviation of at least three letters (e.g., Feb), or a number (e.g., 2). If you specify a number, followed by the letter "f" or "p", the month of the following or previous year, respectively, display. For instance, `-m 2f` displays February of next year.
`-y year`	Specify a year to display. For example, `-y 1970` displays the entire calendar of the year 1970.
`-3`	Display last month, this month, and next month.
`-1`	Display only this month. This is the default.
`-A num`	Display num months occurring after any months already specified. For example, `-3 -A 3` displays last month, this month, and four months after this one; and `-y 1970 -A 2` displays every month in 1970, and the first two months of 1971.

Option	Description
`-B num`	Display num months occurring before any months already specified. For example, `-3 -B 2` displays the previous three months, this month, and next month.
`-d YYYY-MM`	Operate as if the current month is number MM of year YYYY.

Examples:

1. Display the calendar for this month, with today highlighted.

```
cal
```

2. Same as the previous command, but do not highlight today.

```
cal -h
```

3. Display last month, this month, and next month.

```
cal -3
```

4. Display this entire year's calendar.

```
cal -y
```

5. Display the entire year 2000 calendar.

```
cal -y 2000
```

6. Same as the previous command.

```
cal 2000
```
47

7. Display the calendar for December of this year.

```
cal -m [December, Dec, or 12]
```

10. Display the calendar for December 2000.

```
cal 12 2000
```

The bc command

The bc command provides the functionality of being able to perform mathematical calculations through the command line.

Examples:

1 . Arithmetic:

```
Input : $ echo "11+5" | bc
Output : 16
```

2 . Increment:

- var –++ : Post increment operator, the result of the variable is used first and then the variable is incremented.
- – ++var : Pre increment operator, the variable is increased first and then the result of the variable is stored.

```
Input: $ echo "var=3;++var" | bc
Output: 4
```

3 . Decrement:

- var – – : Post decrement operator, the result of the variable is used first and then the variable is decremented.
- – – var : Pre decrement operator, the variable is decreased first and then the result of the variable is stored.

```
Input: $ echo "var=3;--var" | bc
Output: 2
```

4 . Assignment:

- var = value : Assign the value to the variable
- var += value : similar to var = var + value
- var -= value : similar to var = var – value
- var *= value : similar to var = var * value
- var /= value : similar to var = var / value
- var ^= value : similar to var = var ^ value
- var %= value : similar to var = var % value

```
Input: $ echo "var=4;var" | bc
Output: 4
```

5 . Comparison or Relational:

- If the comparison is true, then the result is 1. Otherwise,(false), returns 0
- expr1<expr2 : Result is 1, if expr1 is strictly less than expr2.
- expr1<=expr2 : Result is 1, if expr1 is less than or equal to expr2.
- expr1>expr2 : Result is 1, if expr1 is strictly greater than expr2.
- expr1>=expr2 : Result is 1, if expr1 is greater than or equal to expr2.
- expr1==expr2 : Result is 1, if expr1 is equal to expr2.
- expr1!=expr2 : Result is 1, if expr1 is not equal to expr2.

```
Input: $ echo "6<4" | bc
Output: 0
```

```
Input: $ echo "2==2" | bc
Output: 1
```

6 . Logical or Boolean:

- expr1 && expr2 : Result is 1, if both expressions are non-zero.
- expr1 || expr2 : Result is 1, if either expression is non-zero.
- ! expr : Result is 1, if expr is 0.

```
Input: $ echo "! 1" | bc
Output: 0

Input: $ echo "10 && 5" | bc
Output: 1
```

Syntax:

```
bc [ -hlwsqv ] [long-options] [  file ... ]
```

Additional Flags and their Functionalities:

Note: This does not include an exhaustive list of options.

Short Flag	Long Flag	Description
-i	--interactive	Force interactive mode
-l	--mathlib	Use the predefined math routines
-q	--quiet	Opens the interactive mode for bc without printing the header
-s	--standard	Treat non-standard bc constructs as errors

Short Flag	Long Flag	Description
`-w`	`--warn`	Provides a warning if non-standard bc constructs are used

Notes:

1. The capabilities of `bc` can be further appreciated if used within a script. Aside from basic arithmetic operations, `bc` supports increments/decrements, complex calculations, logical comparisons, etc.
2. Two of the flags in `bc` refer to non-standard constructs. If you evaluate `100>50 | bc` for example, you will get a strange warning. According to the POSIX page for bc, relational operators are only valid if used within an `if`, `while`, or `for` statement.

The `df` command

The `df` command in Linux/Unix is used to show the disk usage & information. `df` is an abbreviation for "disk free".

`df` displays the amount of disk space available on the file system containing each file name argument. If no file name is given, the space available on all currently mounted file systems is shown.

Syntax

```
df [OPTION]... [FILE]...
```

Options

Short Flag	Long Flag	Description
`-a`	`--all`	Include pseudo, duplicate, inaccessible file systems.
`-B`	`--block-size=SIZE`	Scale sizes by SIZE before printing them; e.g., `-BM` prints sizes in units of 1,048,576 bytes; see SIZE format below.
`-h`	`--human-readable`	Print sizes in powers of 1024 (e.g., 1023M).
`-H`	`--si`	Print sizes in powers of 1000 (e.g., 1.1G).
`-i`	`--inodes`	List inode information instead of block usage.

Short Flag	Long Flag	Description
`-k`	-	Like `--block-size=1K`.
`-l`	`--local`	Limit listing to local file systems.
-	`--no-sync`	Do not invoke `sync` before getting usage info (default).
-	`--output[=FIELD_LIST]`	Use the output format defined by `FIELD_LIST`, or print all fields if FIELD_LIST is omitted.
`-P`	`--portability`	Use the `POSIX` output format
-	`--sync`	Invoke sync before getting usage info.
-	`--total`	Elide all entries insignificant to available space, and produce a grand total.
`-t`	`--type=TYPE`	Limit listing to file systems of type `TYPE`.
`-T`	`--print-type`	Print file system type.
`-x`	`--exclude-type=TYPE`	Limit listing to file systems not of type `TYPE`.
`-v`	-	Ignored; included for compatibility reasons.
-	`--help`	Display help message and exit.
-	`--version`	Output version information and exit.

Examples:

1. Show available disk space **Action:** --- Output the available disk space and where the directory is mounted

Details: --- Outputted values are not human-readable (are in bytes)

Command:

```
df
```

54

2. Show available disk space in human-readable form **Action:** ---
   Output the available disk space and where the directory is
   mounted

Details: --- Outputted values ARE human-readable (are in GBs/MBs)

Command:

```
df -h
```

3. Show available disk space for the specific file system **Action:** ---
   Output the available disk space and where the directory is
   mounted

Details: --- Outputted values are only for the selected file system

Command:

```
df -hT file_system_name
```

4. Show available inodes **Action:** --- Output the available inodes for
   all file systems

Details: --- Outputted values are for inodes and not available space

Command:

```
df -i
```

5. Show file system type **Action:** --- Output the file system types

55

Details: --- Outputted values are for all file systems

Command:

```
df -T
```

6. Exclude file system type from the output **Action:** --- Output the information while excluding the chosen file system type

Details: --- Outputted values are for all file systems EXCEPT the chosen file system type

Command:

```
df -x file_system_type
```

The `help` command

The `help` command displays information about builtin commands. Display information about builtin commands.

If a PATTERN is specified, this command gives detailed help on all commands matching the PATTERN, otherwise the list of available help topics is printed.

Quick Tip: The `help` command only works for commands that are "built-in" to the Bash shell itself (like `cd`, `pwd`, `echo`, `read`). It will not work for standalone programs like `ls`, `grep`, or `find`. To get help for those, you should use the `man` command (e.g., `man ls`).

Syntax

57

```
$ help [-dms] [PATTERN ...]
```

Options

58

Option Description

`-d`   Output short description for each topic.

`-m`   Display usage in pseudo-manpage format.

`-s`   Output only a short usage synopsis for each topic matching the provided PATTERN.

Examples of uses:

59

1. We get the complete information about the cd command

```
$ help cd
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

    Change the current directory to DIR.  The default DIR is
the value of the
    HOME shell variable.
...
(additional details about options and exit status follow)
```

2. We get a short description about the pwd command

```
$ help -d pwd
pwd: pwd [-LP]
    Print the name of the current working directory.
```

3. We get the syntax of the cd command

```
$ help -s cd
cd [-L|[-P [-e]] [-@]] [dir]
```

The `factor` command

The `factor` command prints the prime factors of each specified integer
`NUMBER`. If none are specified on the command line, it will read them
from the standard input.

Syntax

61

```
$ factor [NUMBER]...
```

OR:

```
$ factor OPTION
```

Options

62

Option	Description
`--help`	Display this a help message and exit.
`--version`	Output version information and exit.

Examples

1. Print prime factors of a prime number.

```
$ factor 50
```

2. Print prime factors of a non-prime number.

```
$ factor 75
```

The `uname` command

The `uname` command lets you print out system information and defaults to outputting the kernel name.

Syntax:

65

```
$ uname [OPTION]
```

Syntax:

Examples

1. Print out all system information.

```
$ uname -a
```

2. Print out the kernel version.

```
$ uname -v
```

Options

67

Short Flag	Long Flag	Description
-a	--all	Print all information, except omit processor and hardware platform if unknown.
-s	--kernel-name	Print the kernel name.
-n	--nodename	Print the network node hostname.
-r	--kernel-release	Print the kernel release.
-v	--kernel-version	Print the kernel version.
-m	--machine	Print the machine hardware name.
-p	--processor	Print the processor type (non-portable).
-i	--hardware-platform	Print the hardware platform (non-portable).
-o	--operating-system	Print the operating system.

The `mkdir` command

The `mkdir` command in Linux/Unix is used to create a directory.

Syntax

69

```
$ mkdir [-m=mode] [-p] [-v] [-Z=context] directory [directory
...]
```

Examples

1. Make a directory named **myfiles**.

```
$ mkdir myfiles
```

2. Create a directory named **myfiles** at the home directory:

```
$ mkdir ~/myfiles
```

3. Create the **mydir** directory, and set its file mode (`-m`) so that all users (`a`) may read (`r`), write (`w`), and execute (`x`) it.

```
$ mkdir -m a=rwx mydir
```

You can also create sub-directories of a directory. It will create the parent directory first, if it doesn't exist. If it already exists, then it move further to create the sub-directories without any error message.

For directories, this means that any user on the system may view ("read"), and create/modify/delete ("write") files in the directory. Any user may also change to ("execute") the directory, for example with the `cd` command.

4. Create the directory **/home/test/src/python**. If any of the parent directories **/home**, **/home/test**, or **/home/test/src** do not already exist, they are automatically created.

```
$ mkdir -p /home/test/src/python
```

Options

71

Short Flags	Long Flags	Descriptions
`-m`	`--mode=MODE`	Set file mode (as in chmod), not `a=rwx -umask`.
`-p`	`--parents`	No error if existing, make parent directories as needed.
`-v`	`--verbose`	Print a message for each created directory.
`-Z`	`--context=CTX`	Set the SELinux security context of each created directory to CTX.
-	`--help`	Display a help message and exit.
-	`--version`	Output version information and exit.

The `gzip` command

The `gzip` command in Linux/Unix is used to compress/decompress data.

Usage

73

Compress a file

74

Action: --- Compressing a file

Details: --- Reduce the size of the file by applying compression

Command:

```
gzip file_name
```

Decompress a file

75

Action: --- Decompressing a file

Details: --- Restore the file's original form in terms of data and size

Command:

```
gzip -d archive_01.gz
```

Compress multiple files:

Action: --- Compress multiple files

Details: --- Compress multiple files into multiple archives

Command:

```
gzip file_name_01 file_name_02 file_name_03
```

Decompress multiple files:

Action: --- Decompress multiple files

Details: --- Decompress multiple files from multiple archives

Command:

```
gzip -d archive_01.gz archive_02.gz archive_03.gz
```

Compress a directory:

Action: --- Compress all the files in a directory

Details: --- Compress multiple files under a directory in one single archive

Command:

```
gzip -r directory_name
```

Decompress a directory:

Action: --- Decompress all the files in a directory

Details: --- Decompress multiple files under a directory from one single archive

Command:

```
gzip -dr directory_name
```

Verbose (detailed) output while compressing:

Action: --- Compress a file in a more verbose manner

Details: --- Output more information about the action of the command

Command:

```
gzip -v file_name
```

The `whatis` command

The `whatis` command is used to display one-line manual page descriptions for commands. It can be used to get a basic understanding of what a (unknown) command is used for.

Examples of uses:

1. To display what `ls` is used for:

```
whatis ls
```

2. To display the use of all commands which start with `make`, execute the following:

```
whatis -w make*
```

Syntax:

```
whatis [-OPTION] [KEYWORD]
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-d	--debug	Show debugging messages
-r	--regex	Interpret each keyword as a regex

Short Flag Long Flag Description

-w   --wildcard The keyword(s) contain wildcards

The who command

The who command lets you print out a list of logged-in users, the current run level of the system and the time of last system boot.

Examples

1. Print out all details of currently logged-in users

```
who -a
```

2. Print out the list of all dead processes

```
who -d -H
```

Syntax:

```
who [options] [filename]
```

Additional Flags and their Functionalities

Short Flag	Description
-r	prints all the current runlevel
-d	print all the dead processes
-q	print all the login names and total number Lof logged on users

Short Flag	Description
-h	print the heading of the columns displayed
-b	print the time of last system boot

The `free` command

The `free` command in Linux/Unix is used to show memory (RAM/SWAP) information.

Usage

86

Show memory usage

Action: --- Output the memory usage - available and used, as well as swap

Details: --- The values are shown in kibibytes by default.

Command:

```
free
```

Show memory usage in human-readable form

Action: --- Output the memory usage - available and used, as well as swap

Details: --- Outputted values ARE human-readable (are in GB / MB)

Command:

```
free -h
```

Show memory usage with a total line

Action: --- Output the memory usage and also add a summary line with the total.

Details: --- The `-t` flag is useful for seeing the combined total of memory and swap.

Command:

```
free -t
```

The `top/htop` command

`top` is the default command-line utility that comes pre-installed on Linux distributions and Unix-like operating systems. It is used for displaying information about the system and its top CPU-consuming processes as well as RAM usage.

`htop` is interactive process-viewer and process-manager for Linux and Unix-like operating system based on ncurses. If you take top and put it on steroids, you get htop.

Comparison between top and htop:

Feature	top	htop
Type	Interactive system-monitor, process-viewer and process-manager	Interactive system-monitor, process-viewer and process-manager
Operating System	Linux distributions, macOS	Linux distributions, macOS
Installation	Built-in and is always there. Also has more adoption due to this fact.	Doesn't come preinstalled on most Linux distros. Manual installation is needed
User Interface	Basic text only	Colorful and nicer text-graphics interface
Scrolling Support	No	Yes, supports horizontal and vertical scrolling
Mouse Support	No	Yes
Process utilization	Displays processes but not in tree format	Yes, including user and kernel threads
Scrolling Support	No	Yes, supports horizontal and vertical scrolling
Mouse Support	No	Yes
Process utilization	Displays processes but not in tree format	Yes, including user and kernel threads
Network Utilization	No	No
Disk Utilization	No	No
Comments	Has a learning curve for some advanced options like searching, sending messages to processes, etc. It is good to have some knowledge of top because it is the default process viewer on many systems.	Easier to use and supports vi like searching with /. Sending messages to processes (kill, renice) is easier and doesn't require typing in the process number like top.

Examples:

top

1. To display dynamic real-time information about running processes:

```
top
```

2. Sorting processes by internal memory size (default order - process ID):

```
top -o mem
```

3. Sorting processes first by CPU, then by running time:

```
top -o cpu -O time
```

4. Display only processes owned by given user:

```
top -user {user_name}
```

htop

1. Display dynamic real-time information about running processes. An enhanced version of `top`.

```
htop
```

2. displaying processes owned by a specific user:

```
htop --user {user_name}
```

3. Sort processes by a specified `sort_item` (use `htop --sort help` for available options):

```
htop --sort {sort_item}
```
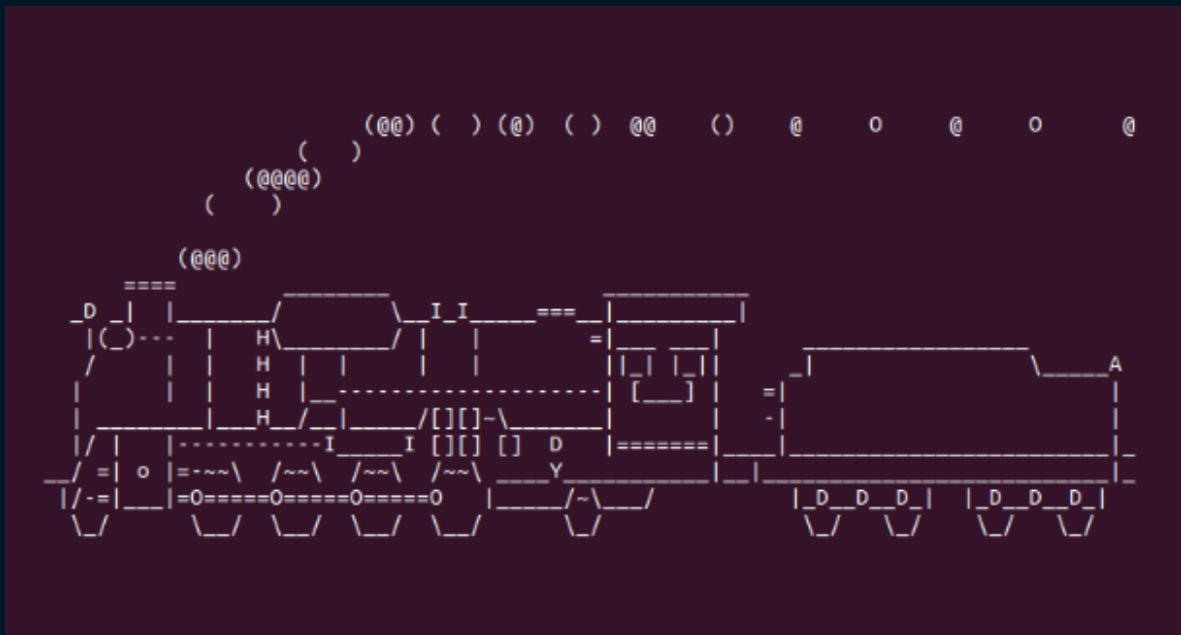
Syntax:

94

```
top [OPTIONS]
```

```
htop [OPTIONS]
```

Syntax:

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-a	-	Sort by memory usage.
-b	-	Batch mode operation. Starts top in 'Batch mode', which could be useful for sending output from top to other programs or to a file. In this mode, top will not accept input and runs until the iterations limit you've set with the '-n' command-line option or until killed.
-h	-	`top --user {user_name}` Only display processes owned by user.
-U	-user	Help.
-u	-	This is an alias equivalent to: -o cpu -O time.

The `sl` command

The `sl` command in Linux is a humorous program that runs a steam locomotive(sl) across your terminal.

```
                         (@@) (  ) (@)  ( )  @@    ()    @     O     @     O      @
                    (   )
                 (@@@@)
              (    )

            (@@@)
      ====        _____                _____
  _D _|  |_____/        \__I_I_____===__|_____|
   |(_)---  |   H_____/ |   |        =|___ ___|
   /     |  |   H  |  |     |   |         ||_| |_||
  |      |  |   H  |__--------------------| [___] |   =|
  | _____|___H__/__|_____/[][]~_____|       |   -|
  |/ |   |-----------I_____I [][] []  D   |=======|____|
 __/ =| o |=-~~\  /~~\  /~~\  /~~\ ____Y_____|__|_____
  |/-=|___|=0=====O=====O=====O====|_____/~\___/          |_D__D__D_|  |_D__D__D_|_
   \_/      \__/  \__/  \__/  \__/      \_/               \_/   \_/    \_/   \_/
```

Installation

97

Install the package before running.

```
sudo apt install sl
```

Syntax

98

```
sl
```

The `echo` command

The `echo` command lets you display the line of text/string that is passed as an argument

Examples:

1. To Show the line of text or string passed as an argument:

```
echo Hello There
```

2. To show all files/folders similar to the `ls` command:

```
echo *
```

3. To save text to a file named foo.bar:

```
echo "Hello There" > foo.bar
```

4. To append text to a file named foo.bar:

```
echo "Hello There" >> foo.bar
```

Syntax:

```
echo [option] [string]
```

It is usually used in shell scripts and batch files to output status text to the screen or a file.The -e used with it enables the interpretation of backslash escapes

Additional Options and their Functionalities:

Option Description

\b	removes all the spaces in between the text
\c	suppress trailing new line with backspace interpretor '-e' to continue without emitting new line.
\n	creates new line from where it is used
\t	creates horizontal tab spaces
\r	carriage returns with backspace interpretor '-e' to have specified carriage return in output
\v	creates vertical tab spaces
\a	alert returns with a backspace interpretor '-e' to have sound alert
-n	omits echoing trailing newline .

The `finger` Command

The `finger` command displays information about local system users by querying files such as `/etc/passwd`, `/var/run/utmp`, and `/var/log/wtmp`. It is a local command and does not rely on any service or daemon to run. This command helps to quickly retrieve user-related details such as login times, idle status, and other system information.

Examples:

1. View details about a particular user.

```
finger abc
```

Output

```
Login: abc                          Name: (null)
Directory: /home/abc                Shell: /bin/bash
On since Mon Nov  1 18:45 (IST) on :0 (messages off)
On since Mon Nov  1 18:46 (IST) on pts/0 from :0.0
New mail received Fri May  7 10:33 2013 (IST)
Unread since Sat Jun  7 12:59 2003 (IST)
No Plan.
```

2. View login details and idle status about a user.

```
finger -s root
```

Output

```
                              102

  Login      Name                       Tty       Idle  Login Time
  Office        Office Phone
  root          root            *1      19d Wed 17:45
  root          root            *2       3d Fri 16:53
  root          root            *3          Mon 20:20
  root          root            *ta      2  Tue 15:43
  root          root            *tb      2  Tue 15:44
```

Syntax:

```
finger [-l] [-m] [-p] [-s] [username]
```

Additional Flags and Their Functionalities:

Flag Description

-l   Force long output format.

-m   Match arguments only on username (not first or last name).

-p   Suppress printing of the .plan file in a long format printout.

-s   Force short output format.

Additional Information:

Default Format:
The default format includes items like login name, full username, terminal name, and write status. The command provides details like idle time, login time, and site-specific information.

Longer Format:
In a long format, the command adds details such as the user's home directory, login shell, and the contents of .plan and .project files.

103

Privacy Considerations

While the `finger` command is useful for retrieving information about system users, it may also expose sensitive details in shared or multi-user environments:

1. **Usernames and Login Times**: Displays login times, which can be used to track user activity.
2. **Home Directories**: Exposes paths to users' home directories.
3. **Idle Status**: Shows how long a user has been inactive, potentially signaling whether they are actively using their system.
4. **Mail Status**: Displays mail information, which may inadvertently reveal user engagement.

Potential Risks:

In environments with untrusted users, the information exposed by `finger` could be exploited for:

- **Social Engineering Attacks**: Malicious actors could use this information to craft personalized phishing attacks.
- **Timing Attacks**: Knowing when a user is idle or active could give attackers an advantage in timing their attempts.
- **Targeted Attacks**: Knowledge of user home directories can focus attacks on those locations.

Mitigating Privacy Risks:

To mitigate these risks, consider limiting access to the `finger` command in environments where user privacy is important.

105

The `in.fingerd` Service

It's important to distinguish between the `finger` command and the **`in.fingerd` service**. The `finger` command is local, while `in.fingerd` is a network daemon that allows remote queries of user information. This service is typically disabled by default in modern systems due to potential security risks.

If enabled, the `in.fingerd` service can expose user information over the network, which could be exploited by attackers. To mitigate this risk, system administrators should ensure the service is disabled if it is not needed.

Disabling the `in.fingerd` Service:

If you are concerned about remote queries, you can disable the `in.fingerd` service:

```
sudo systemctl disable in.fingerd
sudo systemctl stop in.fingerd
```

By disabling the `in.fingerd` service, you prevent remote querying of user information, enhancing system security.

The `groups` command

In Linux, there can be multiple users (those who use/operate the system), and groups (a collection of users). Groups make it easy to manage users with the same security and access privileges. A user can be part of different groups.

Important Points:

The `groups` command prints the names of the primary and any supplementary groups for each given username, or the current process if no names are given. If more than one name is given, the name of each user is printed before the list of that user's groups and the username is separated from the group list by a colon.

Syntax:

```
groups [username]
```

Example 1

Provided with a username

```
groups demon
```

In this example, username demon is passed with groups command and the output shows the groups in which the user demon is present, separated by a colon.

Example 2

When no username is passed then this will display the group membership for the current user:

```
groups
```

Here the current user is demon . So when we run the `groups` command without arguments we get the groups in which demon is a user.

Example 3

Passing root with groups command:

```
$demon# groups
```

Note: Primary and supplementary groups for a process are normally inherited from its parent and are usually unchanged since login. This means that if you change the group database after logging in, groups will not reflect your changes within your existing login session. The only options are –help and –version.

The man command

The man command is used to display the manual of any command that we can run on the terminal. It provides information like: DESCRIPTION, OPTIONS, AUTHORS and more.

Examples:

1. Man page for printf:

```
man printf
```

2. Man page section 2 for intro:

```
man 2 intro
```

3. Viewing the Manual for a Local File (using the -l flag):

```
man -l [LOCAL-FILE]
```

Syntax:

```
man [SECTION-NUM] [COMMAND NAME]
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-f	-	Return the sections of an command
-a	-	Display all the manual pages of an command
-k	-	Searches the given command with RegEx in all man pages
-w	-	Returns the location of a given command man page
-I	-	Searches the command manual case sensitive

The `passwd` command

In Linux, `passwd` command changes the password of user accounts. A normal user may only change the password for their own account, but a superuser may change the password for any account. `passwd` also changes the account or associated password validity period.

Example

112

```
$ passwd
```

The syntax of the `passwd` command is :

```
$ passwd [options] [LOGIN]
```

options

114

```
-a, --all
        This option can be used only with -S and causes show
status for all users.

-d, --delete
        Delete a user's password.

-e, --expire
        Immediately expire an account's password.

-h, --help
        Display help message and exit.

-i, --inactive
        This option is used to disable an account after the
password has been expired for a number of days.

-k, --keep-tokens
        Indicate password change should be performed only for
expired authentication tokens (passwords).

-l, --lock
        Lock the password of the named account.

-q, --quiet
        Quiet mode.

-r, --repository
        change password in repository.

-S, --status
        Display account status information.
```

The w command

The w command displays information about the users that are currently active on the machine and their processes.

Examples:

1. Running the w command without arguments shows a list of logged on users and their processes.

```
w
```

2. Show information for the user named *hope*.

```
w hope
```

Syntax:

```
finger [-l] [-m] [-p] [-s] [username]
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-h	--no-header	Don't print the header.

Short Flag	Long Flag	Description
`-u`	`--no-current`	Ignores the username while figuring out the current process and cpu times. *(To see an example of this, switch to the root user with `su` and then run both `w` and `w -u`.)*
`-s`	`--short`	Display abbreviated output *(don't print the login time, JCPU or PCPU times).*
`-f`	`--from`	Toggle printing the from *(remote hostname)* field. The default as released is for the from field to not be printed, although your system administrator or distribution maintainer may have compiled a version where the from field is shown by default.
`--help`	-	Display a help message, and exit.
`-V`	`--version`	Display version information, and exit.
`-o`	`--old-style`	Old style output *(prints blank space for idle times less than one minute).*
user	-	Show information about the specified the user only.

Additional Information

The <u>header</u> of the output shows (in this order): the current time, how long the system has been running, how many users are currently logged on, and the system <u>load</u> averages for the past 1, 5, and 15 minutes.

The following entries are displayed for each user:

- login name the <u>tty</u>
- name the <u>remote</u>
- <u>host</u> they are
- logged in from the amount of time they are logged in their
- <u>idle</u> time JCPU

- PCPU
- command line of their current process

The JCPU time is the time used by all processes attached to the tty. It does not include past background jobs, but does include currently running background jobs.

The PCPU time is the time used by the current process, named in the "what" field.

The `whoami` command

The `whoami` command displays the username of the current effective user. In other words it just prints the username of the currently logged-in user when executed.

To display your effective user id just type `whoami` in your terminal:

```
manish@godsmack:~$ whoami
# Output:
manish
```

Syntax:

```
whoami [-OPTION]
```

There are only two options which can be passed to it :

`--help`: Used to display the help and exit

Example:

```
whoami --help
```

Output:

```
Usage: whoami [OPTION]...
Print the user name associated with the current effective user
ID.
Same as id -un.

      --help     display this help and exit
      --version  output version information and exit
```

--version: Output version information and exit

Example:

```
whoami --version
```

Output:

```
whoami (GNU coreutils) 8.32
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<https://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute
it.
There is NO WARRANTY, to the extent permitted by law.

Written by Richard Mlynarik.
```

The `history` command

The `history` command displays a list of previously executed commands from your current shell session and past sessions. This allows you to review, search, and re-execute commands without retyping them.

Command Syntax

```
history [options] [n]
```

How History Works

Your command history is stored in a file in your home directory:

- **Bash**: `~/.bash_history`
- **Zsh**: `~/.zsh_history`

The number of commands stored is controlled by shell variables:

- `HISTSIZE`: Maximum number of commands to keep in memory during a session
- `HISTFILESIZE`: Maximum number of commands to keep in the history file

You can check these values with:

```
echo $HISTSIZE
echo $HISTFILESIZE
```

Common Options

- `history n` - Display the last `n` commands
- `history -c` - Clear the history list (current session only)
- `history -d offset` - Delete the history entry at position `offset`
- `history -a` - Append new history lines to the history file
- `history -w` - Write the current history to the history file

Examples

1. Display your full command history:

```
history
```

2. Show only the last 10 commands:

```
history 10
```

3. Search for specific commands in your history:

```
history | grep artisan
history | grep git
history | grep docker
```

4. Execute a command by its history number:

```
!123
```

This re-runs the command at position 123 in your history.

5. Execute the most recent command starting with a specific string:

```
!git
```

This runs the most recent command that started with "git".

6. Execute the previous command:

```
!!
```

7. Execute the previous command with sudo:

```
sudo !!
```

8. Reuse arguments from the previous command:

```
# If you ran: cat /var/log/syslog
# You can use the last argument with:
vim !$
# This runs: vim /var/log/syslog
```

9. Clear your command history:

```
history -c
```

10. Delete a specific entry from history:

```
history -d 456
```

Reverse Search (Ctrl+R)

One of the most powerful features is **reverse incremental search**:

1. Press `Ctrl+R`
2. Start typing part of a command
3. The most recent matching command appears
4. Press `Ctrl+R` again to cycle through older matches
5. Press `Enter` to execute, or `Esc` to edit the command

History Control Variables

You can customize history behavior using these variables in your `~/.bashrc` or `~/.zshrc`:

```
# Increase history size
export HISTSIZE=10000
export HISTFILESIZE=20000

# Ignore duplicate commands
export HISTCONTROL=ignoredups

# Ignore commands starting with a space
export HISTCONTROL=ignorespace

# Combine both options
export HISTCONTROL=ignoreboth

# Ignore specific commands from being saved
export HISTIGNORE="ls:cd:pwd:exit:history"

# Add timestamps to history
export HISTTIMEFORMAT="%Y-%m-%d %H:%M:%S "
```

Security Considerations

Preventing sensitive commands from being saved:

1. **Prefix with a space** (if `HISTCONTROL=ignorespace` is set):

```
  mysql -u root -p
```

2. **Temporarily disable history:**

```
set +o history
# Run sensitive commands here
set -o history
```

3. **Remove specific entries:**

```
history -d <line_number>
```

4. **Clear history before logging out:**

```
history -c && history -w
```

Practical Use Cases

- **Debugging**: Review the sequence of commands that led to an error
- **Documentation**: Copy command sequences for scripts or documentation
- **Efficiency**: Quickly re-execute complex commands without retyping
- **Learning**: Review commands used by others when sharing a system (in appropriate contexts)
- **Audit trail**: Track what commands were executed and when (with timestamps enabled)

The `login` Command

The `login` command initiates a user session.

Syntax

127

```
$ login [-p] [-h host] [-H] [-f username|username]
```

Flags and their functionalities

Short Flag	Description
-f	Used to skip a login authentication. This option is usually used by the getty(8) autologin feature.
-h	Used by other servers (such as telnetd(8) to pass the name of the remote host to login so that it can be placed in utmp and wtmp. Only the superuser is allowed use this option.
-p	Used by getty(8) to tell login to preserve the environment.
-H	Used by other servers (for example, telnetd(8)) to tell login that printing the hostname should be suppressed in the login: prompt.
--help	Display help text and exit.
-v	Display version information and exit.

Examples

129

To log in to the system as user abhishek, enter the following at the login prompt:

```
$ login: abhishek
```

If a password is defined, the password prompt appears. Enter your password at this prompt.

`lscpu` **command**

`lscpu` in Linux/Unix is used to display CPU Architecture info. `lscpu` gathers CPU architecture information from `sysfs` and `/proc/cpuinfo` files.

For example :

```
manish@godsmack:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    2
Core(s) per socket:    2
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 142
Model name:            Intel(R) Core(TM) i5-7200U CPU @
2.50GHz
Stepping:              9
CPU MHz:               700.024
CPU max MHz:           3100.0000
CPU min MHz:           400.0000
BogoMIPS:              5399.81
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              3072K
NUMA node0 CPU(s):     0-3
```

130

Options

`-a, --all` Include lines for online and offline CPUs in the output (default for -e). This option may only specified together with option -e or -p. For example: `lsof -a`

`-b, --online` Limit the output to online CPUs (default for -p). This option may only be specified together with option -e or -p. For example: `lscpu -b`

`-c, --offline` Limit the output to offline CPUs. This option may only be specified together with option -e or -p.

`-e, --extended [=list]` Display the CPU information in human readable format. For example: `lsof -e`

For more info: use `man lscpu` or `lscpu --help`

The cp command

The cp is a command-line utility for copying files and directory. cp stands for copy. This command is used to copy files or group of files or directory. It creates an exact image of a file on a disk with different file name. The cp command requires at least two filenames in its arguments.

Examples:

1. To copy the contents of the source file to the destination file.

```
cp sourceFile destFile
```

If the destination file doesn't exist then the file is created and the content is copied to it. If it exists then the file is overwritten.

2. To copy a file to another directory specify the absolute or the relative path to the destination directory.

```
cp sourceFile /folderName/destFile
```

3. To copy a directory, including all its files and subdirectories

```
cp -R folderName1 folderName2
```

The command above creates the destination directory and recursively

copies all files and subdirectories from the source to the destination directory.

If the destination directory already exists, the source directory itself and its content are copied inside the destination directory.

4. To copy only the files and subdirectories but not the source directory

```
cp -RT folderName1 folderName2
```

Syntax:

The general syntax for the cp command is as follows:

```
cp [OPTION] SOURCE DESTINATION
cp [OPTION] SOURCE DIRECTORY
cp [OPTION] SOURCE-1 SOURCE-2 SOURCE-3 SOURCE-n DIRECTORY
```

The first and second syntax is used to copy Source file to Destination file or Directory. The third syntax is used to copy multiple Sources(files) to Directory.

Some useful options

1. `-i` (interactive) `i` stands for Interactive copying. With this option system first warns the user before overwriting the destination file. cp prompts for a response, if you press y then it overwrites the file and with any other option leave it uncopied.

```
$ cp -i file1.txt fileName2.txt
cp: overwrite 'file2.txt'? y
```

2. -b(backup) -b(backup): With this option cp command creates the backup of the destination file in the same folder with the different name and in different format.

```
$ ls
a.txt b.txt

$ cp -b a.txt b.txt

$ ls
a.txt b.txt b.txt~
```

3. -f(force) If the system is unable to open destination file for writing operation because the user doesn't have writing permission for this file then by using -f option with cp command, destination file is deleted first and then copying of content is done from source to destination file.

```
$ ls -l b.txt
-r-xr-xr-x+ 1 User User 3 Nov 24 08:45 b.txt
```

User, group and others doesn't have writing permission.

Without -f option, command not executed

```
$ cp a.txt b.txt
cp: cannot create regular file 'b.txt': Permission denied
```

With -f option, command executed successfully

```
$ cp -f a.txt b.txt
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
`-i`	--interactive	prompt before overwrite
`-f`	--force	If an existing destination file cannot be opened, remove it and try again
`-b`	-	Creates the backup of the destination file in the same folder with the different name and in different format.
`-r or -R`	`--recursive`	**cp** command shows its recursive behavior by copying the entire directory structure recursively.
`-n`	`--no-clobber`	do not overwrite an existing file (overrides a previous -i option)
`-p`	-	preserve the specified attributes (default: mode,ownership,timestamps), if possible additional attributes: context, links, xattr, all

The mv command

The mv command lets you **move one or more files or directories** from one place to another in a file system like UNIX. It can be used for two distinct functions:

- To rename a file or folder.
- To move a group of files to a different directory.

Note: *No additional space is consumed on a disk during renaming, and the mv command doesn't provide a prompt for confirmation*

Syntax:

```
mv [options] source (file or directory)  destination
```

Examples:

1. To rename a file called old_name.txt:

```
mv old_name.txt new_name.txt
```

2. To move a file called *essay.txt* from the current directory to a directory called *assignments* and rename it *essay1.txt*:

```
mv essay.txt assignments/essay1.txt
```

3. To move a file called *essay.txt* from the current directory to a directory called *assignments* without renaming it

```
mv essay.txt assignments
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-f	--force	Force move by overwriting destination file without prompt
-i	--interactive	Interactive prompt before overwrite
-u	--update	Move only when the source file is newer than the destination file or when the destination file is missing
-n	--no-clobber	Do not overwrite an existing file
-v	--verbose	Print source and destination files
-b	--backup	Create a Backup of Existing Destination File

The ps command

The ps command is used to identify programs and processes that are running on the system and the resources they are using. Its frequently pipelined with other commands like grep to search for a program/process or less so that the user can analyze the output one page at a time.

Let's say you have a program like openshot which is notorious for hogging system resources when exporting a video, and you want to close it, but the GUI has become unresponsive.

Example

1. You want to find the PID of openshot and kill it.

```
ps aux | grep openshot
kill - <openshot PID>
```

2. To Show all the running processes:

```
ps -A
```

Syntax

```
ps [options]
```

When run without any options, it's useless and will print: CMD - the

executable processes/(program) running, their `PID` - process ID, `TTY` - terminal type and `Time` - How long the process has utilized the CPU or thread.

Common Option

If you are going to remember only one thing from this page let it be these three letter `aux`: `a` - which displays all processes running, including those being run by other users. `u` - which shows the effective user of a process, i.e. the person whose file access permissions are used by the process. `x` - which shows processes that do not have a `TTY` associated with them.

Additional Options:

Option	Description
a	Shows list all processes with a terminal (tty)
-A	Lists all processes. Identical to `-e`
-a	Shows all processes except both session leaders and processes not associated with a terminal
-d	Select all processes except session leaders
--deselect	Shows all processes except those that fulfill the specified conditions. Identical to `-N`
-e	Lists all processes. Identical to `-A`
-N	Shows all processes except those that fulfill the specified conditions. Identical to `-deselect`
T	Select all processes associated with this terminal. Identical to the `-t` option without any argument
r	Restrict the selection to only running processes
--help simple	Shows all the basic options
--help all	Shows every available options

Another useful command which give a realtime snapshot of the

processes and the resources they are using about every ten seconds is
`top`.

The `kill` command

`kill` command in Linux (located in /bin/kill), is a built-in command which is used to terminate processes manually. The `kill` command sends a signal to a process which terminates the process. If the user doesn't specify any signal which is to be sent along with kill command then default *TERM* signal is sent that terminates the process.

Signals can be specified in three ways:

- **By number (e.g. -5)**
- **With SIG prefix (e.g. -SIGkill)**
- **Without SIG prefix (e.g. -kill)**

Syntax

```
kill [OPTIONS] [PID]...
```

Examples:

1. To display all the available signals you can use below command option:

```
kill -l
```

2. To show how to use a *PID* with the *kill* command.

```
$kill pid
```
                                142

3. To show how to send signal to processes.

```
kill {-signal | -s signal} pid
```

4. Specify Signal:

• using numbers as signals

```
kill -9 pid
```

• using SIG prefix in signals

```
kill -SIGHUP pid
```

• without SIG prefix in signals

```
kill -HUP pid
```

Arguments:

The list of processes to be signaled can be a mixture of names and
PIDs.

```
   pid     Each pid can be expressed in one of the following
ways:

        n       where n is larger than 0.  The process with
PID n is signaled.

        0       All processes in the current process group
are signaled.

        -1      All processes with a PID larger than 1 are
signaled.

        -n      where n is larger than 1.  All processes in
process group  n  are  signaled.
                When  an  argument  of  the  form '-n' is
given, and it is meant to denote a
                process group, either a signal must be
specified first, or the argument must
                be  preceded  by  a '--' option, otherwise it
will be taken as the signal to
                send.

   name    All processes invoked using this name will be
signaled.
```

Options:

```
   -s, --signal signal
          The signal to send.  It may be given as a name or a
number.

   -l, --list [number]
          Print a list of signal names, or convert the given
signal number to  a  name.   The
          signals can be found in /usr/include/linux/signal.h.

   -L, --table
          Similar to -l, but it will print signal names and
their corresponding numbers.

   -a, --all
          Do  not  restrict the command-name-to-PID conversion
to processes with the same UID
          as the present process.

   -p, --pid
          Only print the process ID (PID) of the named
processes, do not send any signals.

   --verbose
          Print PID(s) that will be signaled with kill along
with the signal.
```

The `killall` command

`killall` sends a signal to **all** processes running any of the specified commands. If no signal name is specified, `SIGTERM` is sent. In general, `killall` command kills all processes by knowing the name of the process.

Signals can be specified either by name (e.g. `-HUP` or `-SIGHUP`) or by number (e.g. `-1`) or by option `-s`.

If the command name is not a regular expression (option `-r`) and contains a slash (`/`), processes executing that particular file will be selected for killing, independent of their name.

`killall` returns a zero return code if at least one process has been killed for each listed command, or no commands were listed and at least one process matched the `-u` and `-Z` search criteria. `killall` returns non-zero otherwise.

A `killall` process never kills itself (but may kill other `killall` processes).

Examples:

1. Kill all processes matching the name `conky` with `SIGTERM`:

```
killall conky
# OR
killall -SIGTERM conky
# OR
kilall -15 conky
```

I was able to kill Wine (which are Windows exe files running on Linux) applications this way too.

```
killall TQ.exe
```

2. List all the supported signals:

```
$ killall -l
HUP INT QUIT ILL TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE
ALRM TERM STKFLT
CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH
POLL PWR SYS
```

As for the numbers.

```
$ for s in $(killall -l); do echo -n "$s " && kill -l $s; done
HUP 1
INT 2
QUIT 3
ILL 4
TRAP 5
ABRT 6
BUS 7
FPE 8
KILL 9
USR1 10
SEGV 11
USR2 12
PIPE 13
ALRM 14
TERM 15
STKFLT 16
CHLD 17
CONT 18
STOP 19
TSTP 20
TTIN 21
TTOU 22
URG 23
XCPU 24
XFSZ 25
VTALRM 26
PROF 27
WINCH 28
POLL 29
PWR 30
SYS 31
```

3. Ask before killing, to prevent unwanted kills:

```
$ killall -i conky
Kill conky(1685) ? (y/N)
```

4. Kill all processes and wait until the processes die.

```
killall -w conky
```

5. Kill based on time:

```
# Kill all firefox younger than 2 minutes
killall -y 2m  firefox

# Kill all firefox older than 2 hours
killall -o 2h  firefox
```

Syntax:

```
killall [OPTION]... [--] NAME...
killall -l, --list
killall -V, --version
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-e	--exact	require an exact match for very long names
-I	--ignore-case	case insensitive process name match
-g	--process-group	kill process group instead of process
-y	--younger-than	kill processes younger than TIME
-o	--older-than	kill processes older than TIME
-i	--interactive	ask for confirmation before killing
-l	--list	list all known signal names
-q	--quiet	don't print complaints
-r	--regexp	interpret NAME as an extended regular expression

Short Flag	Long Flag	Description
-s	--signal SIGNAL	send this signal instead of SIGTERM
-u	--user USER	kill only process(es) running as USER
-v	--verbose	report if the signal was successfully sent
-w	--wait	wait for processes to die
-n	--ns PID	match processes that belong to the same namespaces as PID
-Z	--context	REGEXP kill only process(es) having context (must precede other arguments)

Related commands

kill, pidof

The env command

The env command in Linux/Unix is used to either print a list of the current environment variables or to run a program in a custom environment without changing the current one.

Syntax

```
env [OPTION]... [-] [NAME=VALUE]... [COMMAND [ARG]...]
```

Usage

1. Print out the set of current environment variables

   ```
   env
   ```

2. Run a command with an empty environment

   ```
    env -i command_name
   ```

3. Remove variable from the environment

   ```
   env -u variable_name
   ```

4. End each output with NULL

   ```
   env -0
   ```

Full List of Options

153

Short Flag	Long Flag	Description
`-i`	`--ignore-environment`	Start with an empty environment
`-0`	`--null`	End each output line with NUL, not newline
`-u`	`--unset=NAME`	Remove variable from the environment
`-C`	`--chdir=DIR`	Change working directory to DIR
`-S`	`--split-string=S`	Process and split S into separate arguments. It's used to pass multiple arguments on shebang lines
`-v`	`--debug`	Print verbose information for each processing step
-	`--help`	Print a help message
-	`--version`	Print the version information

The `printenv` command

The `printenv` prints the values of the specified [environment VARIABLE(s)](). If no [VARIABLE]() is specified, print name and value pairs for them all.

Examples:

1. Display the values of all environment variables.

   ```
   printenv
   ```

2. Display the location of the current user's [home directory]().

   ```
   printenv HOME
   ```

3. To use the `--null` command line option as the terminating character between output entries.

   ```
   printenv --null SHELL HOME
   ```

NOTE: By default, the `printenv` command uses newline as the terminating character between output entries.

Syntax:

```
printenv [OPTION]... PATTERN...
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-0	--null	End each output line with **0** byte rather than newline.
--help	-	Display a help message, and exit.

The `hostname` command

`hostname` is used to display the system's DNS name, and to display or set its hostname or NIS domain name.

Syntax:

```
hostname [-a|--alias] [-d|--domain] [-f|--fqdn|--long] [-A|--all-fqdns] [-i|--ip-address] [-I|--all-ip-addresses] [-s|--short] [-y|--yp|--nis]
```

Examples:

1. `hostname -a`, `hostname --alias` Display the alias name of the host (if used). This option is deprecated and should not be used anymore.

2. `hostname -s`, `hostname --short` Display the short host name. This is the host name cut at the first dot.

3. `hostname -V`, `hostname --version` Print version information on standard output and exit successfully.

Help Command

Run below command to view the complete guide to `hostname` command.

```
man hostname
```

The `nano` command

The `nano` command lets you create/edit text files.

Installation:

Nano text editor is pre-installed on macOS and most Linux distros. It's an alternative to `vi` and `vim`. To check if it is installed on your system type:

```
nano --version
```

If you don't have `nano` installed you can do it by using the package manager:

Ubuntu or Debian:

```
sudo apt install nano
```

Examples:

1. Open an existing file, type `nano` followed by the path to the file:

```
nano /path/to/filename
```

2. Create a new file, type `nano` followed by the filename:

```
nano filename
```

159

3. Open a file with the cursor on a specific line and character use the following syntax:

```
nano +line_number,character_number filename
```

Overview of some Shortcuts and their Functionalities:

Shortcut Description
Ctrl + S Save current file
Ctrl + O Offer to write file ("Save as")
Ctrl + X Close buffer, exit from nano
Ctrl + K Cut current line into cutbuffer
Ctrl + U Paste contents of cutbuffer
Alt + 6   Copy current line into cutbuffer
Alt + U   Undo last action
Alt + E   Redo last undone action

The `rm` command

`rm` which stands for "remove" is a command used to remove *(delete)* specific files. It can also be used to remove directories by using the appropriate flag.

Example:

```
rm filename.txt
```

Syntax

```
rm [OPTION] [FILE|DIRECTORY]
```

Flags and their Functionalities:

Short Flag	Long Flag	Description
-f	--force	Ignore nonexistance of files or directories, never prompt
-i	-	Prompt before every removal
-I	-	Prompt once before removal of more than 3 files, or when removing recursively
-d	--dir	remove empty directories
-v	--verbose	explain what is being done

Short Flag	Long Flag	Description
`-r` or `-R`	`--recursive`	remove directories and their contents recursively
`-`	`--help`	Display help then exit
`-`	`--version`	First, Print version Information, Then exit
`-`	`--no-preserve-root`	do not treat `/` specially
`-`	`-preserve-root[=all]`	do not remove `/` (default) with 'all', reject any command line argument on a separate device from its parent
`-`	`--interactive[=WHEN]`	prompt according to WHEN, never, once `-I`, or always `-i`, without WHEN, prompt always
`-`	`--one-file-system`	when removing a hierarchy recursively, skip any directory that is on a file system different from that of the corresponding command line argument0

IMPORTANT NOTICE:

1. `rm` doesn't remove directories by default, so use `-r`, `-R`, `--recursive` options to remove each listed directory, along with all of its contents.
2. To remove a file whose name starts with `-` such as `-foo`, use one of the following commands:
    - `rm -- -foo`
    - `rm ./-foo`
3. To ensure that files/directories being deleted are truly unrecoverable, consider using the `shred` command.

The `ifconfig` command

`ifconfig` is used to configure the kernel-resident network interfaces. It is used at boot time to set up interfaces as necessary. After that, it is usually only needed when debugging or when system tuning is needed.

If no arguments are given, `ifconfig` displays the status of the currently active interfaces. If a single interface argument is given, it displays the status of the given interface only; if a single -a argument is given, it displays the status of all interfaces, even those that are down. Otherwise, it configures an interface.

Syntax:

```
ifconfig [-v] [-a] [-s] [interface]
ifconfig [-v] interface [aftype] options
```

Examples:

1. To display the currently active interfaces:

```
ifconfig
```

2. To show all interfaces which are currently active, even if down:

```
ifconfig -a
```

162

3. To show all the error conditions:

```
ifconfig -v
```

4. To show a short list:

```
ifconfig -s
```

5. To display details of the specific network interface (say `eth0`):

```
ifconfig eth0
```

6. To activate the driver for a interface (say `eth0`):

```
ifconfig eth0 up
```

7. To deactivate the driver for a interface (say `eth0`):

```
ifconfig eth0 down
```

8. To assign a specific IP address to a network interface (say `eth0`):

```
ifconfig eth0 10.10.1.23
```

9. To change MAC(Media Access Control) address of a network interface (say `eth0`):

```
ifconfig eth0 hw ether AA:BB:CC:DD:EE:FF
```

10. To define a netmask for a network interface (say `eth0`):

```
ifconfig eth0 netmask 255.255.255.224
```

11. To enable promiscous mode on a network interface (say `eth0`):

```
ifconfig eth0 promisc
```

In normal mode, when a packet is received by a network card, it verifies that it belongs to itself. If not, it drops the packet normally. However, in the promiscuous mode, it accepts all the packets that flow through the network card.

12. To disable promiscous mode on a network interface (say `eth0`):

```
ifconfig eth0 -promisc
```

13. To set the maximum transmission unit to a network interface (say `eth0`):

```
ifconfig eth0 mtu 1000
```

The MTU allows you to set the limit size of packets that are transmitted on an interface. The MTU is able to handle a maximum number of octets to an interface in one single transaction.

14. To add additional IP addresses to a network interface, you can configure a network alias to the network interface:

```
ifconfig eth0:0 10.10.1.24
```

Please note that the alias network address is in the same subnet mask of the network interface. For example, if your eth0 network ip address is `10.10.1.23`, then the alias ip address can be `10.10.1.24`. Example of an invalid IP address is `10.10.2.24` since the interface subnet mask is `255.255.255.224`

15. To remove a network alias:

```
ifconfig eth0:0 down
```

Remember that for every scope (i.e. same net with address/netmask combination) all aiases are deleted, if you delete the first alias.

Help Command

Run below command to view the complete guide to `ifconfig` command.

```
man ifconfig
```

The `ip` command

The `ip` command is present in the net-tools which is used for performing several network administration tasks. IP stands for Internet Protocol. This command is used to show or manipulate routing, devices, and tunnels. It can perform tasks like configuring and modifying the default and static routing, setting up tunnel over IP, listing IP addresses and property information, modifying the status of the interface, assigning, deleting and setting up IP addresses and routes.

Examples:

1. To assign an IP Address to a specific interface (eth1) :

```
ip addr add 192.168.50.5 dev eth1
```

2. To show detailed information about network interfaces like IP Address, MAC Address information etc. :

```
ip addr show
```

Syntax:

```
ip [ OPTIONS ] OBJECT { COMMAND | help }
```

166

Additional Flags and their Functionalities:

Flag Description

Flag	Description
`-a`	Display and modify IP Addresses
`-l`	Display and modify network interfaces
`-r`	Display and alter the routing table
`-n`	Display and manipulate neighbor objects (ARP table)
`-ru`	Rule in routing policy database.
`-s`	Output more information. If the option appears twice or more, the amount of information increases
`-f`	Specifies the protocol family to use
`-r`	Use the system's name resolver to print DNS names instead of host addresses
`-c`	To configure color output

The `clear` command

In linux, the `clear` command is used to clear terminal screen.

Example
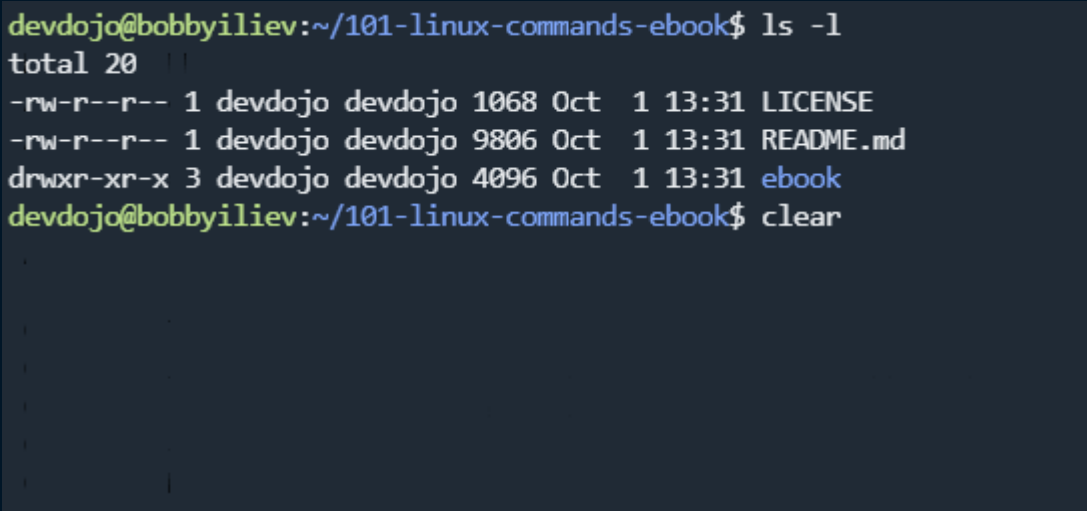
169

```
$ clear
```

Before:

170

```
$ echo Hello World
Hello World

$ clear
```

Before:

After executing clear command:

```
$
```

Screenshot:

After running the command your terminal screen will be clear:

The `su` command

In linux, `su` allows you to run commands with a substitute user and group ID.

When called without arguments, `su` defaults to running an interactive shell as root.

Example :

173

```
$ su
```

In case that you wanted to switch to a user called `devdojo`, you could do that by running the following command:

```
$ su devdojo
```

The syntax of the `su` command is :

```
$ su [options] [-] [<user>[<argument>...]]
```

Options :

```
-m, -p          --> do not reset environment variables
-w              --> do not reset specified variables
-g              --> specify the primary group
-G              --> specify a supplemental group
-l              --> make the shell a login shell
-f              --> pass -f to the shell (for csh or tcsh)
-s              --> run <shell> if /etc/shell allows it
-p              --> create a new pseudo terminal
-h              --> display this help
-v              --> display version
```

The `wget` command

The `wget` command is used for downloading files from the Internet. It supports downloading files using HTTP, HTTPS and FTP protocols. It allows you to download several files at once, download in the background, resume downloads, limit the bandwidth, mirror a website, and much more.

Syntax

The `wget` syntax requires you to define the downloading options and the URL the to be downloaded file is coming from.

```
$ wget [options] [URL]
```

Examples

In this example we will download the Ubuntu 20.04 desktop iso file from different sources. Go over to your terminal or open a new one and type in the below `wget`. This will stat the download. The download may take a few minutes to complete.

1. Starting a regular download

```
wget
https://releases.ubuntu.com/20.04/ubuntu-20.04.3-desktop-amd64
.iso
```

2. You can resume a download using the `-c` option

```
wget -c
https://mirrors.piconets.webwerks.in/ubuntu-mirror/ubuntu-rele
ases/20.04.3/ubuntu-20.04.3-desktop-amd64.iso
```

3. To download in the background, use the `-b` option

```
wget -b
https://mirrors.piconets.webwerks.in/ubuntu-mirror/ubuntu-rele
ases/20.04.3/ubuntu-20.04.3-desktop-amd64.iso
```

More options

179

On top of downloading, `wget` provides many more features, such as downloading multiple files, dowloading in the background, limiting download bandwith and resuming stopped downloads. View all `wget` options in its man page.

```
man wget
```

Additional Flags and their Functionalities

Short Flag	Description
-v	prints version of the wget available on your system
-h	print help message displaying all the possible options
-b	This option is used to send a process to the background as soon as it starts.
-t	This option is used to set number of retries to a specified number of times
-c	This option is used to resume a partially downloaded file

The `curl` command

In Linux, curl is a powerful command-line tool used to transfer data from or to a server using a wide variety of protocols, including HTTP, HTTPS, and FTP. It is often used for testing APIs, downloading files, and automating web-related tasks.

The syntax of the `curl` command :

```
$ curl [options...] <url>
```

The command will print the source code of the example.com homepage in the terminal window.

Common Options :

182

curl has over 200 options! Here are some of the most common and useful ones.

Option	Long Version	Description
`-O`	`--remote-name`	Downloads the file and saves it with the same name as the remote file.
`-o <file>`	`--output <file>`	Saves the downloaded output to a specific filename.
`-L`	`--location`	Follows redirects if the server reports that the requested page has moved.
`-X <METHOD>`	`--request <METHOD>`	Specifies the HTTP request method to use (e.g., POST, PUT, DELETE).
`-H <header>`	`--header <header>`	Allows you to add a custom HTTP header to your request.

Examples :

1. View the source code of a webpage

This is the simplest use of curl. It will fetch the content from the URL and print its HTML source code directly to your terminal.

```
$ curl example.com
```

2. Download a file

The -O flag is used to download a file. curl will save it in your current directory using the same name as the remote file.

```
$ curl -O
https://github.com/bobbyiliev/101-linux-commands/archive/refs/
tags/v1.0.zip
```

3. Download a file and rename it

Using the -o flag, you can specify a new name for the downloaded file.

```
$ curl -o linux-commands.zip
https://github.com/bobbyiliev/101-linux-commands/archive/refs/
tags/v1.0.zip
```

Installation:

184

The curl command comes with most of the Linux distributions. But, if the system does not carry the curl by default. You need to install it manually. To install the curl, execute the following commands:

Update the system by executing the following commands:

```
$ sudo apt update
$ sudo apt upgrade
```

Now, install the curl utility by executing the below command:

```
$ sudo apt install curl
```

Verify the installation by executing the below command:

```
$ curl -version
```

The above command will display the installed version of the curl command.

The yes command

The yes command in linux is used to print a continuous output stream of given *STRING*. If *STRING* is not mentioned then it prints 'y'. It outputs a string repeatedly unit killed (using something like ctrl + c).

Examples :

1. Prints hello world infinitely in the terminal until killed :

```
yes hello world
```

2. A more generalized command:

```
yes [STRING]
```

Options

186

It accepts the following options:

1. --help
   display this help and exit

2. --version
   output version information and exit

The `last` command

This command shows you a list of all the users that have logged in and out since the creation of the `var/log/wtmp` file. There are also some parameters you can add which will show you for example when a certain user has logged in and how long he was logged in for.

If you want to see the last 5 logs, just add `-5` to the command like this:

```
last -5
```

And if you want to see the last 10, add `-10`.

Another cool thing you can do is if you add `-F` you can see the login and logout time including the dates.

```
last -F
```

There are quite a lot of stuff you can view with this command. If you need to find out more about this command you can run:

```
last --help
```

The `locate` command

The `locate` command searches the file system for files and directories whose name matches a given pattern through a database file that is generated by the `updatedb` command.

Examples:

1. Running the `locate` command to search for a file named `.bashrc`.

```
locate .bashrc
```

Output

```
/etc/bash.bashrc
/etc/skel/.bashrc
/home/linuxize/.bashrc
/usr/share/base-files/dot.bashrc
/usr/share/doc/adduser/examples/adduser.local.conf.examples/bash.bashrc
/usr/share/doc/adduser/examples/adduser.local.conf.examples/skel/dot.bashrc
```

The `/root/.bashrc` file will not be shown because we ran the command as a normal user that doesn't have access permissions to the `/root` directory.

If the result list is long, for better readability, you can pipe the output to the `less` command:

```
locate .bashrc | less
```

2. To search for all `.md` files on the system

```
locate *.md
```

3. To search all `.py` files and display only 10 results

```
locate -n 10 *.py
```

4. To performs case-insensitive search.

```
locate -i readme.md
```

Output

```
/home/linuxize/p1/readme.md
/home/linuxize/p2/README.md
/home/linuxize/p3/ReadMe.md
```

5. To return the number of all files containing `.bashrc` in their name.

```
locate -c .bashrc
```

Output

```
6
```

6. The following would return only the existing `.json` files on the file system.

```
locate -e *.json
```

7. To run a more complex search the `-r` (`--regexp`) option is used. To search for all `.mp4` and `.avi` files on your system and ignore case.

```
locate --regex -i "(\.mp4|\.avi)"
```

Syntax:

```
1.  locate [OPTION]... PATTERN...
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-A	--all	It is used to display only entries that match all PATTERNs instead of requiring only one of them to match.
-b	--basename	It is used to match only the base name against the specified patterns.
-c	--count	It is used for writing the number matching entries instead of writing file names on standard output.
-d	--database DBPATH	It is used to replace the default database with DBPATH.
-e	--existing	It is used to display only entries that refer to existing files during the command is executed.

Short Flag	Long Flag	Description
`-L`	`--follow`	If the `--existing` option is specified, It is used for checking whether files exist and follow trailing symbolic links. It will omit the broken symbolic links to the output. This is the default behavior. The opposite behavior can be specified using the `--nofollow` option.
`-h`	`--help`	It is used to display the help documentation that contains a summary of the available options.
`-i`	`--ignore-case`	It is used to ignore case sensitivity of the specified patterns.
`-p`	`--ignore-spaces`	It is used to ignore punctuation and spaces when matching patterns.
`-t`	`--transliterate`	It is used to ignore accents using iconv transliteration when matching patterns.
`-l`	`--limit, -n LIMIT`	If this option is specified, the command exit successfully after finding LIMIT entries.
`-m`	`--mmap`	It is used to ignore the compatibility with BSD, and GNU locate.
`-0`	`--null`	It is used to separate the entries on output using the ASCII NUL character instead of writing each entry on a separate line.
`-S`	`--statistics`	It is used to write statistics about each read database to standard output instead of searching for files.
`-r`	`--regexp REGEXP`	It is used for searching a basic regexp REGEXP.
`--regex`	`-`	It is used to describe all PATTERNs as extended regular expressions.
`-V`	`--version`	It is used to display the version and license information.
`-w`	`--wholename`	It is used for matching only the whole path name in specified patterns.

191

192

The `iostat` command

The `iostat` command in Linux is used for monitoring system input/output statistics for devices and partitions. It monitors system input/output by observing the time the devices are active in relation to their average transfer rates. The iostat produce reports may be used to change the system configuration to raised balance the input/output between the physical disks. iostat is being included in sysstat package. If you don't have it, you need to install first.

Syntax:

```
iostat [ -c ] [ -d ] [ -h ] [ -N ] [ -k | -m ] [ -t ] [ -V ] [
-x ]
      [ -z ] [ [ [ -T ] -g group_name ] { device [...] | ALL
} ]
      [ -p [ device [,...] | ALL ] ] [ interval [ count ] ]
```

Examples:

1. Display a single history-since-boot report for all CPU and Devices:

```
iostat -d 2
```

2. Display a continuous device report at two-second intervals:

```
iostat -d 2 6
```

3.Display, for all devices, six reports at two-second intervals:

```
iostat -x sda sdb 2 6
```

4.Display, for devices sda and sdb, six extended reports at two-second intervals:

```
iostat -p sda 2 6
```

Additional Flags and their Functionalities:

Short Flag	Description
-x	Show more details statistics information.
-c	Show only the cpu statistic.
-d	Display only the device report
`-xd	Show extended I/O statistic for device only.
-k	Capture the statistics in kilobytes or megabytes.
-k23	Display cpu and device statistics with delay.
-j ID mmcbkl0 sda6 -x -m 2 2	Display persistent device name statistics.
-p	Display statistics for block devices.
-N	Display lvm2 statistic information.

The sudo command

The `sudo` ("substitute user do" or "super user do") command allows a user with proper permissions to execute a command as another user, such as the superuser.

This is the equivalent of "run as administrator" option in Windows. The `sudo` command allows you to elevate your current user account to have root privileges. Also, the root privilege in `sudo` is only valid for a temporary amount of time. Once that time expires, you have to enter your password again to regain root privilege.

WARNING: Be very careful when using the `sudo` command. You can cause irreversible and catastrophic changes while acting as root!

Syntax:

```
sudo [-OPTION] command
```

Additional Flags and their Functionalities:

Flag	Description
-V	The -V (version) option causes sudo to print the version number and exit. If the invoking user is already root, the -V option prints out a list of the defaults sudo was compiled with and the machine's local network addresses
-l	The -l (list) option prints out the commands allowed (and forbidden) the user on the current host.

Flag Description

-L
The -L (list defaults) option lists out the parameters set in a Defaults line with a short description for each. This option is useful in conjunction with grep.

-h
The -h (help) option causes sudo to print a usage message and exit.

-v
If given the -v (validate) option, sudo updates the user's timestamp, prompting for the user's password if necessary. This extends the sudo timeout for another 5 minutes (or whatever the timeout is set to in sudoers) but does not run a command.

-K
The -K (sure kill) option to sudo removes the user's timestamp entirely. Likewise, this option does not require a password.

-u
The -u (user) option causes sudo to run the specified command as a user other than root. To specify a uid instead of a username, use #uid.

-s
The -s (shell) option runs the shell specified by the SHELL environment variable if it's set or the shell as specified in the file passwd.

--
The -- flag indicates that sudo should stop processing command line arguments. It is most useful in conjunction with the -s flag.

Examples

This command switches your command prompt to the BASH shell as a root user:

```
sudo bash
```

Your command line should change to:

```
root@hostname:/home/[username]
```

Adding a string of text to a file is often used to add the name of a software repository to the sources file, without opening the file for editing. Use the following syntax with echo, sudo and tee command:

```
echo 'string-of-text' | sudo tee -a [path_to_file]
```

Example:

```
echo "deb http://nginx.org/packages/debian `lsb_release -cs`
nginx" \ | sudo tee /etc/apt/sources.list.d/nginx.list
```

The `apt` command

`apt` (Advantage package system) command is used for interacting with `dpkg` (packaging system used by debian). There is already the `dpkg` command to manage `.deb` packages. But `apt` is a more user-friendly and efficient way.

In simple terms `apt` is a command used for installing, deleting and performing other operations on debian based Linux.

You will be using the `apt` command mostly with `sudo` privileges.

Installing packages:

`install` followed by `package_name` is used with `apt` to install a new package.

Syntax:

```
sudo apt install package_name
```

Example:

```
sudo apt install g++
```

This command will install g++ on your system.

Removing packages:

remove followed by package_name is used with apt to remove a specific package.

Syntax:

```
sudo apt remove package_name
```

Example:

```
sudo apt remove g++
```

This command will remove g++ from your system.

Searching for a package:

search followed by the package_name used with apt to search a package across all repositories.

Syntax:

```
apt search package_name
```

note: sudo not required

Example:

```
apt search g++
```

Removing unused packages:

Whenever a new package that depends on other packages is installed on the system, the package dependencies will be installed too. When the package is removed, the dependencies will stay on the system. This leftover packages are no longer used by anything else and can be removed.

Syntax:

```
sudo apt autoremove
```

This command will remove all unused from your system.

Updating package index:

apt package index is nothing but a database that stores records of available packages that are enabled on your system.

Syntax:

```
sudo apt update
```

This command will update the package index on your system.

Upgrading packages:

If you want to install the latest updates for your installed packages you may want to run this command.

Syntax:

```
sudo apt upgrade
```

The command doesn't upgrade any packages that require removal of installed packages.

If you want to upgrade a single package, pass the package name:

Syntax:

```
sudo apt upgrade package_name
```

This command will upgrade your packages to the latest version.

The `yum` command

The `yum`command is the primary package management tool for installing, updating, removing, and managing software packages in Red Hat Enterprise Linux. It is an acronym for *Yellow Dog Updater, Modified*.

`yum` performs dependency resolution when installing, updating, and removing software packages. It can manage packages from installed repositories in the system or from .rpm packages.

Syntax:

```
yum -option command
```

Examples:

1. To see an overview of what happened in past transactions:

```
yum history
```

2. To undo a previous transaction:

```
yum history undo <id>
```

3. To install firefox package with 'yes' as a response to all

202

confirmations

```
yum -y install firefox
```

4. To update the mysql package it to the latest stable version

```
yum update mysql
```

Commonly used commands along with yum:

Command	Description
install	Installs the specified packages
remove	Removes the specified packages
search	Searches package metadata for keywords
info	Lists the description
update	Updates each package to the latest version
repolist	Lists repositories
history	Displays what has happened in past transactions
groupinstall	To install a particular package group
clean	To clean all cached files from enabled repository

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-C	--cacheonly	Runs entirely from system cache, doesn't update the cache and use it even in case it is expired.
-	--security	Includes packages that provide a fix for a security issue. Applicable for the upgrade command.
-y	--assumeyes	Automatically answer yes for all questions.

Short Flag	Long Flag	Description
-	`--skip-broken`	Resolves depsolve problems by removing packages that are causing problems from the transaction. It is an alias for the strict configuration option with value False.
`-v`	`--verbose`	Verbose operation, show debug messages.

The `zip` command

The `zip` command is used to compress files and reduce their size. It outputs an archive containing one or more compressed files or directories.

Examples:

In order to compress a single file with the `zip` command the syntax would be the following:

```
zip myZipFile.zip filename.txt
```

This also works with multiple files as well:

```
zip multipleFiles.zip file1.txt file2.txt
```

If you are compressing a whole directory, don't forget to add the `-r` flag:

```
zip -r zipFolder.zip myFolder/
```

Syntax:

```
zip [OPTION] zipFileName filesList
```

Possible options:

Flag Description

-d	Removes the file from the zip archive. After creating a zip file, you can remove a file from the archive using the -d option
-u	Updates the file in the zip archive. This option can be used to update the specified list of files or add new files to the existing zip file. Update an existing entry in the zip archive only if it has been modified more recently than the version already in the zip archive.
-m	Deletes the original files after zipping.
-r	To zip a directory recursively, it will recursively zip the files in a directory. This option helps to zip all the files present in the specified directory.
-x	Exclude the files in creating the zip
-v	Verbose mode or print diagnostic version info. Normally, when applied to real operations, this option enables the display of a progress indicator during compression and requests verbose diagnostic info about zip file structure oddities

The `unzip` command

The `unzip` command extracts all files from the specified ZIP archive to the current directory.

Examples:

In order to extract the files the syntax would be the following:

```
unzip myZipFile.zip
```

To unzip a ZIP file to a different directory than the current one, don't forget to add the `-d` flag:

```
unzip myZipFile.zip -d /path/to/directory
```

To unzip a ZIP file and exclude specific file or files or directories from being extracted, don't forget to add the `-x` flag:

```
unzip myZipFile.zip -x file1.txt file2.txt
```

Syntax:

```
unzip zipFileName [OPTION] [PARAMS]
```

Possible options:

Flag	Description	Params
-d	Unzip an archive to a different directory.	/path/to/directory
-x	Extract the archive but do not extract the specified files.	filename(s)
-j	Unzip without creating new folders, if the zipped archive contains a folder structure.	-
-l	Lists the contents of an archive file without extracting it.	-
-n	Do not overwrite existing files; supply an alternative filename instead.	-
-o	Overwrite files.	-
-P	Supplies a password to unzip a protected archive file.	password
-q	Unzips without writing status messages to the standard output.	-
-t	Tests whether an archive file is valid.	-
-v	Displays detailed (verbose) information about the archive without extracting it.	-

The `shutdown` command

The `shutdown` command lets you bring your system down in a secure way. When `shutdown` is executed the system will notify all logged-in users and disallow further logins. You have the option to shut down your system immediately or after a specific time.

Only users with root (or sudo) privileges can use the `shutdown` command.

Examples:

1. Shut down your system immediately:

```
sudo shutdown now
```

2. Shut down your system after 10 minutes:

```
sudo shutdown +10
```

3. Shut down your system with a message after 5 minutes:

```
sudo shutdown +5 "System will shutdown in 5 minutes"
```

Syntax:

```
shutdown [OPTIONS] [TIME] [MESSAGE]
```

Additional Flags and their Functionalities:

Short Flag Long Flag Description

Short Flag	Long Flag	Description
-r	-	Reboot the system
-c	-	Cancel an scheduled shut down

The `dir` command

The `dir` command lists the contents of a directory(*the current directory by default*). **It differs from ls command in the format of listing the content**. By default, the dir command lists the files and folders in columns, sorted vertically and special characters are represented by backslash escape sequences.

Syntax:

```
dir [OPTIONS] [FILE]
```

Examples:

1. To list files in the current directory:

```
dir
```

2. To list even the hidden files in the current directory:

```
dir -a
```

3. To list the content with detailed information for each entry

```
dir -l
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-a	--all	It displays all the hidden files(starting with `.`) along with two files denoted by `.` and `..`
-A	--almost-all	It is **similar to -a** option except that it *does not display files that signals the current directory and previous directory.*
-l	-	Display detailed information for each entry
-s	--size	Print the allocated size of each file, in blocks File
-h	--human-readable	Used with with -l and -s, to print sizes like in human readable format like 1K, 2M and so on
-F	-	Classifies entries into their type based on appended symbol (`/`, `*`, `@`, `%`, `=`)
-v	--verbose	Print source and destination files
-	--group-directories-first	To group directories before files
-R	--recursive	To List subdirectories recursively.
-S	-	sort by file size, display largest first

The `reboot` Command

The `reboot` command is used to restart a linux system. However, it requires elevated permission using the sudo command. Necessity to use this command usually arises after significant system or network updates have been made to the system.

Syntax

214

```
reboot [OPTIONS...]
```

Options

- **–help** : This option prints a short help text and exit.
- **-halt** : This command will stop the machine.
- **-w**, **–wtmp-only** : This option only writes wtmp shutdown entry, it do not actually halt, power-off, reboot.

Examples

1. Basic Usage. Mainly used to restart without any further details

```
$ sudo reboot
```

However, alternatively the shutdown command with the `-r` option

```
$ sudo shutdown -r now
```

Note that the usage of the reboot, halt and power off is almost similar in syntax and effect. Run each of these commands with –help to see the details.

2. The `reboot` command has limited usage, and the `shutdown` command is being used instead of reboot command to fulfill much more advance reboot and shutdown requirements. One of those situations is a scheduled restart. Syntax is as follows

```
$ sudo shutdown —r [TIME] [MESSAGE]
```

Here the TIME has various formats. The simplest one is now, already been listed in the previous section, and tells the system to restart immediately. Other valid formats we have are +m, where m is the number of minutes we need to wait until restart and HH:MM which specifies the TIME in a 24hr clock.

Example to reboot the system in 2 minutes

```
$ sudo shutdown —r +2
```

Example of a scheduled restart at 03:00 A.M

```
$ sudo shutdown —r 03:00
```

3. Cancelling a Reboot. Usually happens in case one wants to cancel a scheduled restart

Syntax

```
$ sudo shutdown —c [MESSAGE]
```

Usage

```
$sudo shutdown -c "Scheduled reboot cancelled because the chicken crossed the road"
```

4. Checking your reboot logs

```
$ last reboot
```

The `sort` command

the `sort` command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts a file assuming the contents are ASCII. Using options in the sort command can also be used to sort numerically.

Examples:

Suppose you create a data file with name file.txt:

```
Command :
$ cat > file.txt
abhishek
chitransh
satish
rajan
naveen
divyam
harsh
```

Sorting a file: Now use the sort command

Syntax :

```
sort filename.txt
```

```
Command:
$ sort file.txt

Output :
abhishek
chitransh
divyam
harsh
naveen
rajan
satish
```

Note: This command does not actually change the input file, i.e. file.txt.

The sort function on a file with mixed case content

i.e. uppercase and lower case: When we have a mix file with both uppercase and lowercase letters then first the upper case letters would be sorted following with the lower case letters.

Example:

Create a file mix.txt

```
Command :
$ cat > mix.txt
abc
apple
BALL
Abc
bat
```

Now use the sort command

Command :
$ sort mix.txt
Output :
Abc
BALL
abc
apple
bat

The `paste` command

The `paste` command writes lines of two or more files, sequentially and separated by TABs, to the standard output

Syntax:

```
paste [OPTIONS]... [FILE]...
```

Examples:

1. To paste two files

```
paste file1 file2
```

2. To paste two files using new line as delimiter

```
paste -d '\n' file1 file2
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-d	--delimiter	use charater of TAB
-s	--serial	paste one file at a time instead of in parallel

220

Short Flag	Long Flag	Description
-z	--zero-terminated	set line delimiter to NUL, not newline
	--help	print command help
	--version	print version information

The `exit` command

The `exit` command is used to terminate (close) an active shell session

Syntax:

```
exit
```

Shortcut: *Instead of typing `exit`, press `ctrl + D`, it will do the same Functionality.*

The `diff/sdiff` command

This command is used to display the differences in the files by comparing the files line by line.

Syntax:

```
diff [options] File1 File2
```

Example

1. Lets say we have two files with names a.txt and b.txt containing 5 Indian states as follows-:

```
$ cat a.txt
Gujarat
Uttar Pradesh
Kolkata
Bihar
Jammu and Kashmir

$ cat b.txt
Tamil Nadu
Gujarat
Andhra Pradesh
Bihar
Uttar pradesh
```

On typing the diff command we will get below output.

```
$ diff a.txt b.txt
0a1
> Tamil Nadu
2,3c3
< Uttar Pradesh
 Andhra Pradesh
5c5
  Uttar pradesh
```

Flags and their Functionalities

Short Flag	Description
-c	To view differences in context mode, use the -c option.
-u	To view differences in unified mode, use the -u option. It is similar to context mode
-i	By default this command is case sensitive. To make this command case in-sensitive use -i option with diff.
-version	This option is used to display the version of diff which is currently running on your system.

The `tar` command

The `tar` command stands for tape archive, is used to create Archive and extract the Archive files. This command provides archiving functionality in Linux. We can use tar command to create compressed or uncompressed Archive files and also maintain and modify them.

Examples:

1. To create a tar file in abel directory:

```
tar -cvf file-14-09-12.tar /home/abel/
```

2. To un-tar a file in the current directory:

```
tar -xvf file-14-09-12.tar
```

Syntax:

```
tar [options] [archive-file] [file or directory to be archived
```

Additional Flags and their Functionalities:

Use Flag	Description
-c	Creates Archive
-x	Extract the archive

Use Flag	Description
`-f`	Creates archive with given filename
`-t`	Displays or lists files in archived file
`-u`	Archives and adds to an existing archive file
`-v`	Displays Verbose Information
`-A`	Concatenates the archive files
`-z`	zip, tells tar command that creates tar file using gzip
`-j`	Filter archive tar file using tbzip
`w`	Verify a archive file
`r`	update or add file or directory in already existed .tar file
`-?`	Displays a short summary of the project
`-d`	Find the difference between an archive and file system
`--usage`	shows available tar options
`--version`	Displays the installed tar version
`--show-defaults`	Shows default enabled options

Option Flag	Description
`--check-device`	Check device numbers during incremental archive
`-g`	Used to allow compatibility with GNU-format incremental ackups
`--hole-detection`	Used to detect holes in the sparse files
`-G`	Used to allow compatibility with old GNU-format incremental backups
`--ignore-failed-read`	Don't exit the program on file read errors
`--level`	Set the dump level for created archives
`-n`	Assume the archive is seekable
`--no-check-device`	Do not check device numbers when creating archives
`--no-seek`	Assume the archive is not seekable

Option Flag	Description
`--occurrence=N`	`Process only the Nth occurrence of each file
`--restrict`	`Disable use of potentially harmful options
`--sparse-version=MAJOR,MINOR`	Set version of the sparce format to use
`-S`	Handle sparse files efficiently.

Overwright control Flag	Description
`-k`	Don't replace existing files
`--keep-newer-files`	Don't replace existing files that are newer than the archives version
`--keep-directory-symlink`	Don't replace existing symlinks
`--no-overwrite-dir`	Preserve metadata of existing directories
`--one-top-level=DIR`	Extract all files into a DIR
`--overwrite`	Overwrite existing files
`--overwrite-dir`	Overwrite metadata of directories
`--recursive-unlink`	Recursivly remove all files in the directory before extracting
`--remove-files`	Remove files after adding them to a directory
`--skip-old-files`	Don't replace existing files when extracting
`-u`	Remove each file before extracting over it
`-w`	Verify the archive after writing it

The `gunzip` command

The `gunzip` command is an antonym command of <u>gzip command</u>. In other words, it decompresses files deflated by the `gzip` command.

`gunzip` takes a list of files on its command line and replaces each file whose name ends with *.gz*, *-gz*, *.z*, *-z*, or *_z* (ignoring case) and which begins with the correct magic number with an uncompressed file without the original extension. `gunzip` also recognizes the special extensions *.tgz* and *.taz* as shorthands for *.tar.gz* and *.tar.Z* respectively.

Examples:

1. Uncompress a file

   ```
   gunzip filename.gz
   ```

2. Recursively uncompress content inside a directory, that match extension (suffix) compressed formats accepted by `gunzip`:

   ```
   gunzip -r directory_name/
   ```

3. Uncompress all files in the current/working directory whose suffix match *.tgz*:

```
gunzip -S .tgz *
```

4. List compressed and uncompressed sizes, compression ratio and uncompressed name of input compressed file/s:

```
gunzip -l file_1 file_2
```

Syntax:

```
gunzip [ -acfhklLnNrtvV ] [-S suffix] [ name ...  ]
```

Video tutorial about using gzip, gunzip and tar commands:

This video shows how to compress and decompress in a Unix shell. It uses `gunzip` as decompression command.

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-c	--stdout	write on standard output, keep original files unchanged
-h	--help	give help information
-k	--keep	keep (don't delete) input files
-l	--list	list compressed file contents
-q	--quiet	suppress all warnings
-r	--recursive	operate recursively on directories
-S	--suffix=SUF	use suffix SUF on compressed files

Short Flag	Long Flag	Description
	--synchronous	synchronous output (safer if system crashes, but slower)
-t	--test	test compressed file integrity
-v	--verbose	verbose mode
-V	--version	display version number

The `hostnamectl` command

The `hostnamectl` command provides a proper API used to control Linux system hostname and change its related settings. The command also helps to change the hostname without actually locating and editing the `/etc/hostname` file on a given system.

Syntax

```
$ hostnamectl [OPTIONS...] COMMAND ...
```

where **COMMAND** can be any of the following

status: Used to check the current hostname settings

set-hostname NAME: Used to set system hostname

set-icon-name NAME: Used to set icon name for host

Example

233

1. Basic usage to view the current hostnames

```
$ hostnamectl
```

or

```
$ hostnamectl status
```

2. To change the static host name to *myhostname*. It may or may not require root access

```
$ hostnamectl set-hostname myhostname --static
```

3. To set or change a transient hostname

```
$ hostnamectl set-hostname myotherhostname --transient
```

4. To set the pretty hostname. The name that is to be set needs to be in the double quote(" ").

```
$ hostname set-hostname "prettyname" --pretty
```

The `iptables` command

The `iptables` command is a powerful firewall administration tool for Linux systems. It allows you to configure the Linux kernel firewall (netfilter) by setting up, maintaining, and inspecting the tables of IP packet filter rules.

Syntax

235

```
iptables [options] [chain] [rule-specification] [target]
```

Basic Concepts

236

Tables

- **filter**: Default table for packet filtering (INPUT, OUTPUT, FORWARD)
- **nat**: Network Address Translation (PREROUTING, POSTROUTING, OUTPUT)
- **mangle**: Packet alteration (PREROUTING, POSTROUTING, INPUT, OUTPUT, FORWARD)
- **raw**: Connection tracking exemption (PREROUTING, OUTPUT)

Chains

- **INPUT**: Incoming packets to local system
- **OUTPUT**: Outgoing packets from local system
- **FORWARD**: Packets routed through the system
- **PREROUTING**: Packets before routing decision
- **POSTROUTING**: Packets after routing decision

Targets

- **ACCEPT**: Allow the packet
- **DROP**: Silently discard the packet
- **REJECT**: Discard and send error message
- **LOG**: Log the packet and continue processing
- **DNAT**: Destination NAT
- **SNAT**: Source NAT
- **MASQUERADE**: Dynamic source NAT

Basic Commands

237

Listing Rules

```
# List all rules
sudo iptables -L

# List rules with line numbers
sudo iptables -L --line-numbers

# List rules in specific table
sudo iptables -t nat -L
sudo iptables -t mangle -L

# Show packet and byte counters
sudo iptables -L -v

# Show rules in iptables-save format
sudo iptables -S
```

Basic Rule Operations

```
# Add rule to end of chain
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Insert rule at specific position
sudo iptables -I INPUT 1 -p tcp --dport 80 -j ACCEPT

# Delete specific rule
sudo iptables -D INPUT -p tcp --dport 22 -j ACCEPT

# Delete rule by line number
sudo iptables -D INPUT 3

# Replace rule at specific position
sudo iptables -R INPUT 1 -p tcp --dport 443 -j ACCEPT
```

Common Rule Examples

1. Allow/Block Specific Ports

```
# Allow SSH (port 22)
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Allow HTTP (port 80)
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT

# Allow HTTPS (port 443)
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# Block specific port
sudo iptables -A INPUT -p tcp --dport 8080 -j DROP
```

2. Allow/Block by IP Address

```
# Allow specific IP
sudo iptables -A INPUT -s 192.168.1.100 -j ACCEPT

# Block specific IP
sudo iptables -A INPUT -s 192.168.1.50 -j DROP

# Allow subnet
sudo iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT

# Block IP range
sudo iptables -A INPUT -m iprange --src-range
192.168.1.100-192.168.1.200 -j DROP
```

3. Allow/Block by Interface

```
# Allow traffic on loopback
sudo iptables -A INPUT -i lo -j ACCEPT

# Allow on specific interface
sudo iptables -A INPUT -i eth0 -p tcp --dport 80 -j ACCEPT

# Block on specific interface
sudo iptables -A INPUT -i eth1 -j DROP
```

Advanced Rules

1. Stateful Connections

```
# Allow established and related connections
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j
ACCEPT

# Allow new connections on specific ports
sudo iptables -A INPUT -m state --state NEW -p tcp --dport 22
-j ACCEPT
```

2. Rate Limiting

```
# Limit SSH connections (6 per minute)
sudo iptables -A INPUT -p tcp --dport 22 -m limit --limit
6/min -j ACCEPT

# Limit ping requests
sudo iptables -A INPUT -p icmp --icmp-type echo-request -m
limit --limit 1/sec -j ACCEPT
```

3. Time-based Rules

```
# Allow access during business hours
sudo iptables -A INPUT -p tcp --dport 80 -m time --timestart
09:00 --timestop 17:00 -j ACCEPT

# Allow access on weekdays
sudo iptables -A INPUT -p tcp --dport 22 -m time --weekdays
Mon,Tue,Wed,Thu,Fri -j ACCEPT
```

4. Multiport Rules 242

```
# Allow multiple ports
sudo iptables -A INPUT -p tcp -m multiport --dports 22,80,443
-j ACCEPT

# Block multiple ports
sudo iptables -A INPUT -p tcp -m multiport --dports
135,445,1433 -j DROP
```

NAT Configuration

243

1. Source NAT (SNAT)

```
# Static SNAT
sudo iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth0
-j SNAT --to-source 203.0.113.1

# Dynamic SNAT (Masquerading)
sudo iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth0
-j MASQUERADE
```

2. Destination NAT (DNAT)

```
# Port forwarding
sudo iptables -t nat -A PREROUTING -p tcp --dport 8080 -j DNAT
--to-destination 192.168.1.100:80

# Forward to different IP
sudo iptables -t nat -A PREROUTING -d 203.0.113.1 -j DNAT --
to-destination 192.168.1.100
```

Policy Configuration

244

Default Policies

```
# Set default policies
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT ACCEPT

# View current policies
sudo iptables -L | grep "policy"
```

Chain Management

```
# Create custom chain
sudo iptables -N CUSTOM_CHAIN

# Delete custom chain (must be empty)
sudo iptables -X CUSTOM_CHAIN

# Flush specific chain
sudo iptables -F INPUT

# Flush all chains
sudo iptables -F
```

Logging

```
# Log dropped packets
sudo iptables -A INPUT -j LOG --log-prefix "DROPPED: " --log-
level 4

# Log before dropping
sudo iptables -A INPUT -p tcp --dport 23 -j LOG --log-prefix
"TELNET_ATTEMPT: "
sudo iptables -A INPUT -p tcp --dport 23 -j DROP

# View logs
sudo tail -f /var/log/syslog | grep "DROPPED:"
```

Common Firewall Configurations

1. Basic Desktop Firewall

```bash
#!/bin/bash
# Clear existing rules
sudo iptables -F
sudo iptables -X
sudo iptables -t nat -F
sudo iptables -t nat -X

# Default policies
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT ACCEPT

# Allow loopback
sudo iptables -A INPUT -i lo -j ACCEPT

# Allow established connections
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j
ACCEPT

# Allow SSH
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Allow ping
sudo iptables -A INPUT -p icmp --icmp-type echo-request -j
ACCEPT
```

2. Web Server Firewall

```bash
#!/bin/bash
# Basic web server configuration
sudo iptables -F

# Default policies
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT ACCEPT

# Allow loopback and established connections
sudo iptables -A INPUT -i lo -j ACCEPT
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j
ACCEPT

# Allow SSH (limit attempts)
sudo iptables -A INPUT -p tcp --dport 22 -m limit --limit
6/min -j ACCEPT

# Allow HTTP/HTTPS
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# Allow ping
sudo iptables -A INPUT -p icmp --icmp-type echo-request -j
ACCEPT
```

3. Router/Gateway Configuration

```bash
#!/bin/bash
# Enable IP forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

# NAT for internal network
sudo iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth0
-j MASQUERADE

# Allow forwarding for established connections
sudo iptables -A FORWARD -m state --state ESTABLISHED,RELATED
-j ACCEPT

# Allow forwarding from internal network
sudo iptables -A FORWARD -s 192.168.1.0/24 -j ACCEPT
```

Persistence

249

1. Save/Restore Rules

```
# Save current rules
sudo iptables-save > /etc/iptables/rules.v4

# Restore rules
sudo iptables-restore < /etc/iptables/rules.v4

# Install persistence package (Ubuntu/Debian)
sudo apt install iptables-persistent
```

2. Automatic Loading

```
# Create systemd service
sudo vim /etc/systemd/system/iptables-restore.service

[Unit]
Description=Restore iptables firewall rules
Before=network-pre.target

[Service]
Type=oneshot
ExecStart=/sbin/iptables-restore /etc/iptables/rules.v4

[Install]
WantedBy=multi-user.target

# Enable service
sudo systemctl enable iptables-restore.service
```

Troubleshooting

1. Testing Rules

```
# Test connectivity
telnet target-ip port
nc -zv target-ip port

# Check if rule matches
sudo iptables -L -v -n | grep "rule-description"

# Monitor rule usage
watch "sudo iptables -L -v -n"
```

2. Debugging

```
# Enable all logging temporarily
sudo iptables -A INPUT -j LOG --log-prefix "INPUT: "
sudo iptables -A OUTPUT -j LOG --log-prefix "OUTPUT: "
sudo iptables -A FORWARD -j LOG --log-prefix "FORWARD: "

# Monitor logs
sudo tail -f /var/log/syslog | grep
"INPUT:\|OUTPUT:\|FORWARD:"
```

3. Emergency Access

```
# Temporary rule to allow all (emergency)
sudo iptables -I INPUT 1 -j ACCEPT

# Flush all rules (removes all protection)
sudo iptables -F
sudo iptables -X
sudo iptables -P INPUT ACCEPT
sudo iptables -P OUTPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
```

Security Best Practices

1. Default Deny Policy

```
# Always start with deny-all policy
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
# Keep OUTPUT as ACCEPT for normal operation
```

2. Order Matters

```
# More specific rules should come first
sudo iptables -I INPUT 1 -s 192.168.1.100 -p tcp --dport 22 -j
ACCEPT
sudo iptables -A INPUT -p tcp --dport 22 -j DROP
```

3. Rate Limiting Critical Services

```
# Protect SSH from brute force
sudo iptables -A INPUT -p tcp --dport 22 -m recent --set --
name SSH
sudo iptables -A INPUT -p tcp --dport 22 -m recent --update --
seconds 60 --hitcount 3 --name SSH -j DROP
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

Performance Considerations

```
# Use connection tracking for better performance
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j
ACCEPT

# Place frequently matched rules first
sudo iptables -I INPUT 1 -m state --state ESTABLISHED,RELATED
-j ACCEPT

# Use specific matches to reduce processing
sudo iptables -A INPUT -p tcp --dport 80 -s 192.168.1.0/24 -j
ACCEPT
```

Important Notes

254

- Always test rules before making them permanent
- Keep a way to access the system if rules block you out
- Use specific protocols and ports rather than blanket rules
- Monitor logs to understand traffic patterns
- Document your firewall rules for future reference
- Regular backup of working configurations
- Consider using UFW for simpler firewall management

The `iptables` command provides comprehensive firewall capabilities but requires careful planning and testing to avoid security issues or system lockouts.

For more details, check the manual: `man iptables`

The `netstat` command

The term `netstat` stands for Network Statistics. In layman's terms, netstat command displays the current network connections, networking protocol statistics, and a variety of other interfaces.

Check if you have `netstat` on your PC:

```
netstat —v
```

If you don't have `netstat` installed on your PC, you can install it with the following command:

```
sudo apt install net-tools
```

You can use `netstat` command for some use cases given below:

- `Netstat` command with `-nr` flag shows the routing table detail on the terminal.

Example:

```
netstat  -nr
```

- `Netstat` command with `-i` flag shows statistics for the currently configured network interfaces. This command will display the first

10 lines of file `foo.txt` .

Example:

```
netstat  -i
```

- `Netstat` command with `-tunlp` will gives a list of networks, their current states, and their associated ports.

Example:

```
netstat -tunlp
```

- You can get the list of all TCP port connection by using `-at` with `netstat`.

```
netstat  -at
```

- You can get the list of all UDP port connection by using `-au` with `netstat`.

```
netstat  -au
```

- You can get the list of all active connection by using `-l` with `netstat`.

```
netstat  -l
```

The `lsof` command

The `lsof` command shows **file infomation** of all the files opened by a running process. It's name is also derived from the fact that, list open files > `lsof`

An open file may be a regular file, a directory, a block special file, a character special file, an executing text reference, a library , a stream or a network file (Internet socket, NFS file or UNIX domain socket). A specific file or all the files in a file system may be selected by path.

Syntax:

```
lsof [-OPTION] [USER_NAME]
```

Examples:

1. To show all the files opened by all active processes:

```
lsof
```

2. To show the files opened by a particular user:

```
lsof -u [USER_NAME]
```

3. To list the processes with opened files under a specified directory:

```
lsof +d [PATH_TO_DIR]
```

Options and their Functionalities:

Option Additional Options Description

Option	Additional Options	Description
-i	tcp/ udp/ :port	List all network connections running, Additionally, on udp/tcp or on specified port.
-i4	-	List all processes with ipv4 connections.
-i6	-	List all processes with ipv6 connections.
-c	[PROCESS_NAME]	List all the files of a particular process with given name.
-p	[PROCESS_ID]	List all the files opened by a specified process id.
-p	^[PROCESS_ID]	List all the files that are not opened by a specified process id.
+d	[PATH]	List the processes with opened files under a specified directory
+R	-	List the files opened by parent process Id.

Help Command

Run below command to view the complete guide to lsof command.

```
man lsof
```

The `bzip2` command

The `bzip2` command lets you compress and decompress the files i.e. it helps in binding the files into a single file which takes less storage space as the original file use to take.

Syntax:

```
bzip2 [OPTIONS] filenames ...
```

Note : Each file is replaced by a compressed version of itself, with the name original name of the file followed by extension bz2.

Options and their Functionalities:

Option	Alias	Description
-d	--decompress	to decompress compressed file
-f	--force	to force overwrite an existing output file
-h	--help	to display the help message and exit
-k	--keep	to enable file compression, doesn't deletes the original input file
-L	--license	to display the license terms and conditions
-q	--quiet	to suppress non-essential warning messages
-t	--test	to check integrity of the specified .bz2 file, but don't want to decompress them
-v	--erbose	to display details for each compression operation

259

Option	Alias	Description
-V	--version	to display the software version
-z	--compress	to enable file compression, but deletes the original input file

By default, when bzip2 compresses a file, it deletes the original (or input) file. However, if you don't want that to happen, use the -k command line option.

Examples:

1. To force compression:

```
bzip2 -z input.txt
```

Note: This option deletes the original file also

2. To force compression and also retain original input file:

```
bzip2 -k input.txt
```

3. To force decompression:

```
bzip2 -d input.txt.bz2
```

4. To test integrity of compressed file:

```
bzip2 -t input.txt.bz2
```

5. To show the compression ratio for each file processed:

```
bzip2 -v input.txt
```

The `service` command

Service runs a System V init script in as predictable environment as possible, removing most environment variables and with current working directory set to /.

The SCRIPT parameter specifies a System V init script, located in /etc/init.d/SCRIPT. The supported values of COMMAND depend on the invoked script, service passes COMMAND and OPTIONS it to the init script unmodified. All scripts should support at least the start and stop commands. As a special case, if COMMAND is --full-restart, the script is run twice, first with the stop command, then with the start command.

The COMMAND can be at least start, stop, status, and restart.

service --status-all runs all init scripts, in alphabetical order, with the `status` command

Examples :

1. To check the status of all the running services:

```
service --status-all
```

2. To run a script

```
service SCRIPT-Name start
```

3. A more generalized command:

```
service [SCRIPT] [COMMAND] [OPTIONS]
```

The `vmstat` command

The `vmstat` command lets you monitor the performance of your system. It shows you information about your memory, disk, processes, CPU scheduling, paging, and block IO. This command is also referred to as **virtual memory statistic report**.

The very first report that is produced shows you the average details since the last reboot and after that, other reports are made which report over time.

vmstat

As you can see it is a pretty useful little command. The most important things that we see above are the `free`, which shows us the free space that is not being used, `si` shows us how much memory is swapped in every second in kB, and `so` shows how much memory is swapped out each second in kB as well.

vmstat -a

If we run `vmstat -a`, it will show us the active and inactive memory of the system running.

vmstat -d

The `vmstat -d` command shows us all the disk statistics.

As you can see this is a pretty useful little command that shows you different statistics about your virtual memory

The `mpstat` command

The `mpstat` command is used to report processor related statistics. It accurately displays the statistics of the CPU usage of the system and information about CPU utilization and performance.

Syntax:

```
mpstat [options] [<interval> [<count>]]
```

Note : It initializes the first processor with CPU 0, the second one with CPU 1, and so on.

Options and their Functionalities:

Option	Description
-A	to display all the detailed statistics
-h	to display mpstat help
-I	to display detailed interrupts statistics
-n	to report summary CPU statistics based on NUMA node placement
-N	to indicate the NUMA nodes for which statistics are to be reported
-P	to indicate the processors for which statistics are to be reported
-o	to display the statistics in JSON (Javascript Object Notation) format
-T	to display topology elements in the CPU report

Option	Description
`-u`	to report CPU utilization
`-v`	to display utilization statistics at the virtual processor level
`-V`	to display mpstat version
`-ALL`	to display detailed statistics about all CPUs

Examples:

1. To display processor and CPU statistics:

```
mpstat
```

2. To display processor number of all CPUs:

```
mpstat -P ALL
```

3. To get all the information which the tool may collect:

```
mpstat -A
```

4. To display CPU utilization by a specific processor:

```
mpstat -P 0
```

5. To display CPU usage with a time interval:

```
mpstat 1 5
```

Note: This command will print 5 reports with 1 second time

interval

268

The ncdu Command

ncdu (NCurses Disk Usage) is a curses-based version of the well-known du command. It provides a fast way to see what directories are using your disk space.

Example

1. Quiet Mode

```
ncdu -q
```

2. Omit mounted directories

```
ncdu -q -x
```

Syntax

271

```
ncdu [-hqvx] [--exclude PATTERN] [-X FILE] dir
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
`-h`	-	Print a small help message
`-q`	-	Quiet mode. While calculating disk space, ncdu will update the screen 10 times a second by default, this will be decreased to once every 2 seconds in quiet mode. Use this feature to save bandwidth over remote connections.
`-v`	-	Print version.
`-x`	-	Only count files and directories on the same filesystem as the specified dir.
-	`--exclude PATTERN`	Exclude files that match PATTERN. This argument can be added multiple times to add more patterns.
`-X FILE`	`--exclude-from FILE`	Exclude files that match any pattern in FILE. Patterns should be separated by a newline.

The `uniq` command

The `uniq` command in Linux is a command line utility that reports or filters out the repeated lines in a file. In simple words, `uniq` is the tool that helps you to detect the adjacent duplicate lines and also deletes the duplicate lines. It filters out the adjacent matching lines from the input file(that is required as an argument) and writes the filtered data to the output file .

Examples:

In order to omit the repeated lines from a file, the syntax would be the following:

```
uniq kt.txt
```

In order to tell the number of times a line was repeated, the syntax would be the following:

```
uniq -c kt.txt
```

In order to print repeated lines, the syntax would be the following:

```
uniq -d kt.txt
```

In order to print unique lines, the syntax would be the following:

```
uniq -u kt.txt
```

In order to allows the N fields to be skipped while comparing uniqueness of the lines, the syntax would be the following:

```
uniq -f 2 kt.txt
```

In order to allows the N characters to be skipped while comparing uniqueness of the lines, the syntax would be the following:

```
uniq -s 5 kt.txt
```

In order to to make the comparison case-insensitive, the syntax would be the following:

```
uniq -i kt.txt
```

Syntax:

```
uniq [OPTION] [INPUT[OUTPUT]]
```

Possible options:

Flag	Description	Params
-c	It tells how many times a line was repeated by displaying a number as a prefix with the line.	-
-d	It only prints the repeated lines and not the lines which aren't repeated.	-

Flag	Description	Params
-i	By default, comparisons done are case sensitive but with this option case insensitive comparisons can be made.	-
-f	It allows you to skip N fields(a field is a group of characters, delimited by whitespace) of a line before determining uniqueness of a line.	N
-s	It doesn't compares the first N characters of each line while determining uniqueness. This is like the -f option, but it skips individual characters rather than fields.	N
-u	It allows you to print only unique lines.	-
-z	It will make a line end with 0 byte(NULL), instead of a newline.	-
-w	It only compares N characters in a line.	N
--help	It displays a help message and exit.	-
--version	It displays version information and exit.	-

The `RPM` command

`rpm` - RPM Package Manager

`rpm` is a powerful **Package Manager**, which can be used to build, install, query, verify, update, and erase individual software packages. A **package** consists of an archive of files and meta-data used to install and erase the archive files. The meta-data includes helper scripts, file attributes, and descriptive information about the package. Packages come in two varieties: binary packages, used to encapsulate software to be installed, and source packages, containing the source code and recipe necessary to produce binary packages.

One of the following basic modes must be selected: **Query, Verify, Signature Check, Install/Upgrade/Freshen, Uninstall, Initialize Database, Rebuild Database, Resign, Add Signature, Set Owners/Groups, Show Querytags, and Show Configuration.**

General Options

These options can be used in all the different modes.

Short Flag	Long Flag	Description
-?	--help	Print a longer usage message then normal.
-	--version	Print a single line containing the version number of rpm being used.
-	--quiet	Print as little as possible - normally only error messages will be displayed.
-v	-	Print verbose information - normally routine progress messages will be displayed.
-vv	-	Print lots of ugly debugging information.

276

Short Flag	Long Flag	Description
-	--rcfile FILELIST	Each of the files in the colon separated FILELIST is read sequentially by rpm for configuration information. Only the first file in the list must exist, and tildes will be expanded to the value of $HOME. The default FILELIST is /usr/lib/rpm/rpmrc:/usr/lib/rpm/redhat/rpmrc:/etc/rpmrc:~/.rpmrc.
-	--pipe CMD	Pipes the output of rpm to the command CMD.
-	--dbpath DIRECTORY	Use the database in DIRECTORY rather than the default path /var/lib/rpm
-	--root DIRECTORY	Use the file system tree rooted at DIRECTORY for all operations. Note that this means the database within DIRECTORY will be used for dependency checks and any scriptlet(s) (e.g. %post if installing, or %prep if building, a package) will be run after a chroot(2) to DIRECTORY.
-D	--define='MACRO EXPR'	Defines MACRO with value EXPR.
-E	--eval='EXPR'	Prints macro expansion of EXPR.

Short Flag	Long Flag	Description

Synopsis

Querying and Verifying Packages:

```
rpm {-q|--query} [select-options] [query-options]

rpm {-V|--verify} [select-options] [verify-options]

rpm --import PUBKEY ...

rpm {-K|--checksig} [--nosignature] [--nodigest] PACKAGE_FILE
...
```

Installing, Upgrading, and Removing Packages:

```
rpm {-i|--install} [install-options] PACKAGE_FILE ...

rpm {-U|--upgrade} [install-options] PACKAGE_FILE ...

rpm {-F|--freshen} [install-options] PACKAGE_FILE ...

rpm {-e|--erase} [--allmatches] [--nodeps] [--noscripts] [--
notriggers] [--test] PACKAGE_NAME ...
```

Miscellaneous:

```
rpm {--initdb|--rebuilddb}

rpm {--addsign|--resign} PACKAGE_FILE...

rpm {--querytags|--showrc}

rpm {--setperms|--setugids} PACKAGE_NAME .
```

query-options

```
[--changelog] [-c,--configfiles] [-d,--docfiles] [--dump]
[--filesbypkg] [-i,--info] [--last] [-l,--list]
[--provides] [--qf,--queryformat QUERYFMT]
[-R,--requires] [--scripts] [-s,--state]
[--triggers,--triggerscripts]
```

verify-options

```
[--nodeps] [--nofiles] [--noscripts]
[--nodigest] [--nosignature]
[--nolinkto] [--nofiledigest] [--nosize] [--nouser]
[--nogroup] [--nomtime] [--nomode] [--nordev]
[--nocaps]
```

install-options

```
[--aid] [--allfiles] [--badreloc] [--excludepath OLDPATH]
[--excludedocs] [--force] [-h,--hash]
[--ignoresize] [--ignorearch] [--ignoreos]
[--includedocs] [--justdb] [--nodeps]
[--nodigest] [--nosignature] [--nosuggest]
[--noorder] [--noscripts] [--notriggers]
[--oldpackage] [--percent] [--prefix NEWPATH]
[--relocate OLDPATH=NEWPATH]
[--replacefiles] [--replacepkgs]
[--test]
```

The `scp` command

SCP (secure copy) is a command-line utility that allows you to securely copy files and directories between two locations.

Both the files and passwords are encrypted so that anyone snooping on the traffic doesn't get anything sensitive.

Different ways to copy a file or directory:

- From local system to a remote system.
- From a remote system to a local system.
- Between two remote systems from the local system.

Examples:

1. To copy the files from a local system to a remote system:

```
scp /home/documents/local-file root@{remote-ip-address}:/home/
```

2. To copy the files from a remote system to the local system:

```
scp root@{remote-ip-address}:/home/remote-file
/home/documents/
```

3. To copy the files between two remote systems from the local system.

```
scp root@{remote1-ip-address}:/home/remote-file root@{remote2-
ip-address}/home/
```

4. To copy file though a jump host server.

```
scp /home/documents/local-file -oProxyJump=<jump-host-ip>
root@{remote-ip-address}/home/
```

On newer version of scp on some machines you can use the above command with a `-J` flag.

```
scp /home/documents/local-file -J <jump-host-ip> root@{remote-
ip-address}/home/
```

Syntax:

```
scp [OPTION] [user@]SRC_HOST:]file1 [user@]DEST_HOST:]file2
```

- `OPTION` - scp options such as cipher, ssh configuration, ssh port, limit, recursive copy …etc.
- `[user@]SRC_HOST:]file1` - Source file
- `[user@]DEST_HOST:]file2` - Destination file

Local files should be specified using an absolute or relative path, while remote file names should include a user and host specification.

scp provides several that control every aspect of its behaviour. The most widely used options are:

Short Flag	Long Flag	Description
`-P`	-	Specifies the remote host ssh port.
`-p`	-	Preserves files modification and access times.
`-q`	-	Use this option if you want to suppress the progress meter and non-error messages.
`-C`	-	This option forces scp to compresses the data as it is sent to the destination machine.
`-r`	-	This option tells scp to copy directories recursively.

Before you begin

The `scp` command relies on `ssh` for data transfer, so it requires an `ssh key` or `password` to authenticate on the remote systems.

The `colon (:)` is how scp distinguish between local and remote locations.

To be able to copy files, you must have at least read permissions on the source file and write permission on the target system.

Be careful when copying files that share the same name and location on both systems, `scp` will overwrite files without warning.

When transferring large files, it is recommended to run the scp command inside a `screen` or `tmux` session.

The `sleep` command

The `sleep` command is used to create a dummy job. A dummy job helps in delaying the execution. It takes time in seconds by default but a small suffix(s, m, h, d) can be added at the end to convert it into any other format. This command pauses the execution for an amount of time which is defined by NUMBER.

Note: If you will define more than one NUMBER with sleep command then this command will delay for the sum of the values.

Examples :

1. To sleep for 10s

```
sleep 10s
```

2. A more generalized command:

```
sleep NUMBER[SUFFIX]...
```

Options

287

It accepts the following options:

1. --help
   display this help and exit

2. --version
   output version information and exit

The `split` command

The `split` command in Linux is used to split a file into smaller files.

Examples

1. Split a file into a smaller file using file name

```
split filename.txt
```

2. Split a file named filename into segments of 200 lines beginning with prefix file

```
split -l 200 filename file
```

This will create files of the name fileaa, fileab, fileac, filead, etc. of 200 lines.

3. Split a file named filename into segments of 40 bytes with prefix file

```
split -b 40 filename file
```

This will create files of the name fileaa, fileab, fileac, filead, etc. of 40 bytes.

4. Split a file using --verbose to see the files being created.

```
split filename.txt --verbose
```

Syntax:

```
split [options] filename [prefix]
```

Additional Flags and their Functionalities

Short Flag	Long Flag	Description
-a	--suffix-length=N	Generate suffixes of length N (default 2)
	--additional-suffix=SUFFIX	Append an additional SUFFIX to file names
-b	--bytes=SIZE	Put SIZE bytes per output file
-C	--line-bytes=SIZE	Put at most SIZE bytes of records per output file
-d		Use numeric suffixes starting at 0, not alphabetic
	--numeric-suffixes[=FROM]	Same as -d, but allow setting the start value
-x		Use hex suffixes starting at 0, not alphabetic
	--hex-suffixes[=FROM]	Same as -x, but allow setting the start value
-e	--elide-empty-files	Do not generate empty output files with '-n'
	--filter=COMMAND	Write to shell COMMAND; file name is $FILE
-l	--lines=NUMBER	Put NUMBER lines/records per output file

Short Flag	Long Flag	Description
-n	--number=CHUNKS	Generate CHUNKS output files; see explanation below
-t	--separator=SEP	Use SEP instead of newline as the record separator; '\0' (zero) specifies the NUL character
-u	--unbuffered	Immediately copy input to output with '-n r/...'
	--verbose	Print a diagnostic just before each output file is opened
	--help	Display this help and exit
	--version	Output version information and exit

The SIZE argument is an integer and optional unit (example: 10K is 10*1024). Units are K,M,G,T,P,E,Z,Y (powers of 1024) or KB,MB,... (powers of 1000).

CHUNKS may be:

CHUNKS	Description
N	Split into N files based on size of input
K/N	Output Kth of N to stdout
l/N	Split into N files without splitting lines/records
l/K/N	Output Kth of N to stdout without splitting lines/records
r/N	Like 'l' but use round robin distribution
r/K/N	Likewise but only output Kth of N to stdout

The `stat` command

The `stat` command lets you display file or file system status. It gives you useful information about the file (or directory) on which you use it.

Examples:

1. Basic command usage

```
stat file.txt
```

2. Use the `-c` (or `--format`) argument to only display information you want to see (here, the total size, in bytes)

```
stat file.txt -c %s
```

Syntax:

```
stat [OPTION] [FILE]
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-L	--dereference	Follow links

Short Flag	Long Flag	Description
-f	--file-system	Display file system status instead of file status
-c	--format=FORMAT	Specify the format (see below)
-t	--terse	Print the information in terse form
-	--cached=MODE	Specify how to use cached attributes. Can be: always, never, or default
-	--printf=FORMAT	Like --format, but interpret backslash escapes (\n, \t, ...)
-	--help	Display the help and exit
-	--version	Output version information and exit

Example of Valid Format Sequences for Files:

Format	Description
%a	Permission bits in octal
%A	Permission bits and file type in human readable form
%d	Device number in decimal
%D	Device number in hex
%F	File type
%g	Group ID of owner
%G	Group name of owner
%h	Number of hard links
%i	Inode number
%m	Mount point
%n	File name
%N	Quoted file name with dereference if symbolic link
%s	Total size, in bytes
%u	User ID of owner
%U	User name of owner
%w	Time of file birth, human-readable; - if unknown
%x	Time of last access, human-readable

Format	Description
%y	Time of last data modification, human-readable
%z	Time of last status change, human-readable

The `useradd` command

The `useradd` command is used to add or update user accounts to the system.

Examples:

To add a new user with the `useradd` command the syntax would be the following:

```
useradd NewUser
```

To add a new user with the `useradd` command and give a home directory path for this new user the syntax would be the following:

```
useradd -d /home/NewUser NewUser
```

To add a new user with the `useradd` command and give it a specific id the syntax would be the following:

```
useradd -u 1234 NewUser
```

Syntax:

```
useradd [OPTIONS] NameOfUser
```

Possible options:

Flag	Description	Params
-d	The new user will be created using /path/to/directory as the value for the user's login directory	/path/to/directory
-u	The numerical value of the user's ID	ID
-g	Create a user with specific group id	GroupID
-M	Create a user without home directory	-
-e	Create a user with expiry date	DATE (format: YYYY-MM-DD)
-c	Create a user with a comment	COMMENT
-s	Create a user with changed login shell	/path/to/shell
-p	Set an unencrypted password for the user	PASSWORD

The `userdel` command

The `userdel` command is used to delete a user account and related files

Examples:

To delete a user with the `userdel` command the syntax would be the following:

```
userdel userName
```

To force the removal of a user account even if the user is still logged in, using the `userdel` command the syntax would be the following:

```
userdel -f userName
```

To delete a user along with the files in the user's home directory using the `userdel` command the syntax would be the following:

```
userdel -r userName
```

Syntax:

```
userdel [OPTIONS] userName
```

Possible options:

Flag Description

- f Force the removal of the specified user account even if the user is logged in

- r Remove the files in the user's home directory along with the home directory itself and the user's mail spool

- Z Remove any SELinux(Security-Enhanced Linux) user mapping for the user's login.

The `usermod` command

The `usermod` command lets you change the properties of a user in Linux through the command line. After creating a user we sometimes have to change their attributes, like their password or login directory etc. So in order to do that we use the `usermod` command.

Syntax:

```
usermod [options] USER
```

Note : Only superuser (root) is allowed to execute `usermod` command

Options and their Functionalities:

Option Description

-a	to add anyone of the group to a secondary group
-c	to add comment field for the useraccount
-d	to modify the directory for any existing user account
-g	change the primary group for a User
-G	to add supplementary groups
-l	to change existing user login name
-L	to lock system user account
-m	to move the contents of the home directory from existing home dir to new dir
-p	to create an un-encrypted password
-s	to create a specified shell for new accounts

Option Description

-u to assigned UID for the user account

-U to unlock any locked user

Examples:

1. To add a comment/description for a user:

```
sudo usermod -c "This is test user" test_user
```

2. To change the home directory of a user:

```
sudo usermod -d /home/sam test_user
```

3. To change the expiry date of a user:

```
sudo usermod -e 2021-10-05 test_user
```

4. To change the group of a user:

```
sudo usermod -g sam test_user
```

5. To change user login name:

```
sudo usermod -l test_account test_user
```

6. To lock a user:

```
sudo usermod -L test_user
```

7. To unlock a user:

```
sudo usermod -U test_user
```

8. To set an unencrypted password for the user:

```
sudo usermod -p test_password test_user
```

9. To create a shell for the user:

```
sudo usermod -s /bin/sh test_user
```

10. To change the user id of a user:

```
sudo usermod -u 1234 test_user
```

The `ionice` command

The `ionice` command is used to set or get process I/O scheduling class and priority.

If no arguments are given , `ionice` will query the current I/O scheduling class and priority for that process.

Usage

302

```
ionice [options] -p <pid>
```

```
ionice [options] -P <pgid>
```

```
ionice [options] -u <uid>
```

```
ionice [options] <command>
```

A process can be of three scheduling classes:

- **Idle**

  A program with idle I/O priority will only get disk time when `no other program has asked for disk I/O for a defined grace period`.

  The impact of idle processes on normal system actively should be `zero`.

  This scheduling class `doesn't take priority` argument.

  Presently this scheduling class is permitted for an `ordinary user (since kernel 2.6.25)`.

- **Best Effort**

  This is `effective` scheduling class for any process that has `not asked for a specific I/O priority`.

  This class `takes priority argument from 0-7`, with `lower` number being `higher priority`.

  Programs running at the same best effort priority are served in `round- robbin fashion`.

  Note that before kernel 2.6.26 a process that has not asked for an I/O priority formally uses "None" as scheduling class , but the io schedular will treat such processes as if it were in the best effort class.

The priority within best effort class will be dynamically derived form the CPU nice level of the process : io_priority = (cpu_nice + 20) / 5/ for kernels after 2.6.26 with CFQ I/O schedular a process that has not asked for sn io priority inherits CPU scheduling class.

The I/O priority is derived from the CPU nice level of the process (smr sd before kernel 2.6.26).

• **Real Time**

The real time schedular class is given first access to disk, regardless of what else is going on in the system.

Thus the real time class needs to be used with some care, as it cans tarve other processes .

As with the best effort class, 8 priority levels are defined denoting how big a time slice a given process will receive on each scheduling window.

This scheduling class is not permitted for an ordinary user(non-root).

Options

305

Options	Description
-c, --class	name or number of scheduling class, 0: none, 1: realtime, 2: best-effort, 3: idle
-n, --classdata	priority (0..7) in the specified scheduling class,only for the realtime and best-effort classes
-p, --pid ...	act on these already running processes
-P, --pgid ...	act on already running processes in these groups
-t, --ignore	ignore failures
-u, --uid ...	act on already running processes owned by these users
-h, --help	display this help
-V, --version	display version

For more details see ionice(1).

Examples

Command	O/P	Explanation
$ ionice	*none: prio 4*	Running alone ionice will give the class and priority of current process
$ ionice -p 101	*none : prio 4*	Give the details(*class : priority*) of the process specified by given process id
$ ionice -p 2	*none: prio 4*	Check the class and priority of process with pid 2 it is none and 4 resp.
$ ionice -c2 -n0 -p2	2 (best-effort) priority 0 process 2	Now lets set process(pid) 2 as a best-effort program with highest priority
$ ionice -p 2	best-effort : prio 0	Now if I check details of Process 2 you can see the updated one
$ ionice /bin/ls		get priority and class info of bin/ls
$ ionice -n4 -p2		set priority 4 of process with pid 2
$ ionice -p 2	best-effort: prio 4	Now observe the difference between the command ran above and this one we have changed priority from 0 to 4
$ ionice -c0 -n4 -p2	ionice: ignoring given class data for none class	(Note that before kernel 2.6.26 a process that has not asked for an I/O priority formally uses "None" as scheduling class ,
		but the io schedular will treat such processes as if it were in the best effort class.)
		-t option : ignore failure
$ ionice -c0 -n4 -p2 -t		For ignoring the warning shown above we can use -t option so it will ignore failure

Conclusion

307

Thus we have successfully learnt about `ionice` command.

The du command

The `du` command, which is short for `disk usage` lets you retrieve information about disk space usage information in a specified directory. In order to customize the output according to the information you need, this command can be paired with the appropriate options or flags.

Examples:

1. To show the estimated size of sub-directories in the current directory:

```
du
```

2. To show the estimated size of sub-directories inside a specified directory:

```
du {PATH_TO_DIRECTORY}
```

Syntax:

```
du [OPTION]... [FILE]...
du [OPTION]... --files0-from=F
```

Additional Flags and their Functionalities:

Note: This does not include an exhaustive list of options.

Short Flag	Long Flag	Description
`-a`	`--all`	Includes information for both files and directories
`-c`	`--total`	Provides a grand total at the end of the list of files/directories
`-d`	`--max-depth=N`	Provides information up to `N` levels from the directory where the command was executed
`-h`	`--human-readable`	Displays file size in human-readable units, not in bytes
`-s`	`--summarize`	Display only the total filesize instead of a list of files/directories

The `ping` command

The `ping` (Packet Internet Groper) command is a network utility used to check network connectivity between a host and a server or another host. It sends ICMP (Internet Control Message Protocol) echo requests to a specified IP address or URL and measures the time it takes to receive a response. This time delay is referred to as "latency." Ping is a fundamental tool for network troubleshooting and monitoring.

Understanding Latency

Latency, in the context of networking, is the time delay between sending a packet and receiving a response.

When you use the `ping` command, it measures the latency by sending a series of packets to the target host and calculating the time it takes for each packet to complete the round trip. The latency is typically measured in milliseconds (ms). Understanding latency is essential because:

- **Network Performance**: Lower latency means faster data transmission and more responsive network connections, which is critical for real-time applications.

- **Troubleshooting**: High latency can indicate network congestion, packet loss, or connectivity issues that need attention.

- **Quality of Service (QoS)**: Service providers and network administrators use latency metrics to ensure that network services meet quality standards.

The basic ping syntax includes ping followed by a hostname, a name of a website, or the exact IP address.

```
ping [option] [hostname] or [IP address]
```

Examples:

1. To get ping version installed on your system.

```
sudo ping -v
```

2. To check whether a remote host is up, in this case, google.com, type in your terminal:

```
ping google.com
```

3. Controlling the number of packets to send: Earlier we did not define the number of packets to send to the server/host by using -c option we can do so.

```
ping -c 5 google.com
```

4. Controlling the size of the packet: Earlier a default sized packets were sent to a host but we can send light and heavy packet by using -s option.

```
ping -s 40 -c 5 google.com
```

5. Changing the time interval between ping packets: By default ping wait for 1 sec to send next packet we can change this time by using -i option.

```
ping -i 2 google.com
```

The `rsync` command

The `rsync` command is probably one of the most used commands out there. It is used to securely copy files from one server to another over SSH.

Compared to the `scp` command, which does a similar thing, `rsync` makes the transfer a lot faster, and in case of an interruption, you could restore/resume the transfer process.

In this tutorial, I will show you how to use the `rsync` command and copy files from one server to another and also share a few useful tips!

Before you get started, you would need to have 2 Linux servers. I will be using DigitalOcean for the demo and deploy 2 Ubuntu servers.

You can use my referral link to get a free $100 credit that you could use to deploy your virtual machines and test the guide yourself on a few DigitalOcean servers:

DigitalOcean $100 Free Credit

Transfer Files from local server to remote

This is one of the most common causes. Essentially this is how you would copy the files from the server that you are currently on (the source server) to remote/destination server.

What you need to do is SSH to the server that is holding your files, cd to the directory that you would like to transfer over:

```
cd /var/www/html
```

And then run:

```
rsync -avz user@your-remote-server.com:/home/user/dir/
```

The above command would copy all the files and directories from the current folder on your server to your remote server.

Rundown of the command:

- `-a`: is used to specify that you want recursion and want to preserve the file permissions and etc.
- `-v`: is verbose mode, it increases the amount of information you are given during the transfer.
- `-z`: this option, rsync compresses the file data as it is sent to the destination machine, which reduces the amount of data being transmitted -- something that is useful over a slow connection.

I recommend having a look at the following website which explains the commands and the arguments very nicely:

https://explainshell.com/explain?cmd=rsync+-avz

In case that the SSH service on the remote server is not running on the standard 22 port, you could use `rsync` with a special SSH port:

```
rsync -avz -e 'ssh -p 1234' user@your-remote-
server.com:/home/user/dir/
```

Transfer Files remote server to local

In some cases you might want to transfer files from your remote server to your local server, in this case, you would need to use the following syntax:

```
rsync -avz your-user@your-remote-server.com:/home/user/dir/
/home/user/local-dir/
```

Again, in case that you have a non-standard SSH port, you can use the following command:

```
rsync -avz -e 'ssh -p 2510' your-user@your-remote-
server.com:/home/user/dir/ /home/user/local-dir/
```

Transfer only missing files

If you would like to transfer only the missing files you could use the `--ignore-existing` flag.

This is very useful for final sync in order to ensure that there are no missing files after a website or a server migration.

Basically the commands would be the same apart from the appended --ignore-existing flag:

```
  rsync -avz --ignore-existing  user@your-remote-
server.com:/home/user/dir/
```

Conclusion

Using `rsync` is a great way to quickly transfer some files from one machine over to another in a secure way over SSH.

For more cool Linux networking tools, I would recommend checking out this tutorial here:

Top 15 Linux Networking tools that you should know!

Hope that this helps!

Initially posted here: How to Transfer Files from One Linux Server to Another Using rsync

The `dig` command

dig - DNS lookup utility

The `dig` is a flexible tool for interrogating DNS name servers. It performs DNS lookups and displays the answers that are returned from the name server(s) that were queried.

Examples:

1. Dig is a network administration command-line tool for querying the Domain Name System.

```
dig google.com
```

2. The system will list all google.com DNS records that it finds, along with the IP addresses.

```
dig google.com ANY
```

Syntax:

```
dig [server] [name] [type] [q-type] [q-class] {q-opt}
          {global-d-opt} host [@local-server] {local-d-opt}
          [ host [@local-server] {local-d-opt} [...]]
```

Additional Flags and their Functionalities:

```
domain     is in the Domain Name System
        q-class  is one of (in,hs,ch,...) [default: in]
        q-type   is one of
(a,any,mx,ns,soa,hinfo,axfr,txt,...) [default:a]
                 (Use ixfr=version for type ixfr)
        q-opt    is one of:
                 -4                     (use IPv4 query transport
only)
                 -6                     (use IPv6 query transport
only)
                 -b address[#port]     (bind to source
address/port)
                 -c class               (specify query class)
                 -f filename            (batch mode)
                 -k keyfile             (specify tsig key file)
                 -m                     (enable memory usage
debugging)
                 -p port                (specify port number)
                 -q name                (specify query name)
                 -r                     (do not read ~/.digrc)
                 -t type                (specify query type)
                 -u                     (display times in usec
instead of msec)
                 -x dot-notation        (shortcut for reverse
lookups)
                 -y [hmac:]name:key  (specify named base64
tsig key)
        d-opt    is of the form +keyword[=value], where
keyword is:
                 +[no]aaflag            (Set AA flag in query
(+[no]aaflag))
                 +[no]aaonly            (Set AA flag in query
(+[no]aaflag))
                 +[no]additional        (Control display of
additional section)
                 +[no]adflag            (Set AD flag in query
(default on))
                 +[no]all               (Set or clear all display
flags)
                 +[no]answer            (Control display of
```

```
answer section)
                +[no]authority      (Control display of
authority section)
                +[no]badcookie      (Retry BADCOOKIE
responses)
                +[no]besteffort     (Try to parse even
illegal messages)
                +bufsize[=###]      (Set EDNS0 Max UDP packet
size)
                +[no]cdflag         (Set checking disabled
flag in query)
                +[no]class          (Control display of class
in records)
                +[no]cmd            (Control display of
command line -
                                     global option)
                +[no]comments       (Control display of
packet header
                                     and section name
comments)
                +[no]cookie         (Add a COOKIE option to
the request)
                +[no]crypto         (Control display of
cryptographic
                                     fields in records)
                +[no]defname        (Use search list
(+[no]search))
                +[no]dnssec         (Request DNSSEC records)
                +domain=###         (Set default domainname)
                +[no]dscp[=###]     (Set the DSCP value to
### [0..63])
                +[no]edns[=###]     (Set EDNS version) [0]
                +ednsflags=###      (Set EDNS flag bits)
                +[no]ednsnegotiation (Set EDNS version
negotiation)
                +ednsopt=###[:value] (Send specified EDNS
option)
                +noednsopt          (Clear list of +ednsopt
options)
                +[no]expandaaaa     (Expand AAAA records)
                +[no]expire         (Request time to expire)
                +[no]fail           (Don't try next server on
SERVFAIL)
                +[no]header-only    (Send query without a
question section)
```

```
                +[no]identify      (ID responders in short
answers)
                +[no]idnin         (Parse IDN names
[default=on on tty])
                +[no]idnout        (Convert IDN response
[default=on on tty])
                +[no]ignore        (Don't revert to TCP for
TC responses.)
                +[no]keepalive     (Request EDNS TCP
keepalive)
                +[no]keepopen      (Keep the TCP socket open
between queries)
                +[no]mapped        (Allow mapped IPv4 over
IPv6)
                +[no]multiline     (Print records in an
expanded format)
                +ndots=###         (Set search NDOTS value)
                +[no]nsid          (Request Name Server ID)
                +[no]nssearch      (Search all authoritative
nameservers)
                +[no]onesoa        (AXFR prints only one soa
record)
                +[no]opcode=###    (Set the opcode of the
request)
                +padding=###       (Set padding block size
[0])
                +[no]qr            (Print question before
sending)
                +[no]question      (Control display of
question section)
                +[no]raflag        (Set RA flag in query
(+[no]raflag))
                +[no]rdflag        (Recursive mode
(+[no]recurse))
                +[no]recurse       (Recursive mode
(+[no]rdflag))
                +retry=###         (Set number of UDP
retries) [2]
                +[no]rrcomments    (Control display of per-
record comments)
                +[no]search        (Set whether to use
searchlist)
                +[no]short         (Display nothing except
short

                                    form of answers - global
```

```
option)
                +[no]showsearch      (Search with intermediate
results)
                +[no]split=##        (Split hex/base64 fields
into chunks)
                +[no]stats           (Control display of
statistics)
                +subnet=addr         (Set edns-client-subnet
option)
                +[no]tcflag          (Set TC flag in query
(+[no]tcflag))
                +[no]tcp             (TCP mode (+[no]vc))
                +timeout=###         (Set query timeout) [5]
                +[no]trace           (Trace delegation down
from root [+dnssec])
                +tries=###           (Set number of UDP
attempts) [3]
                +[no]ttlid           (Control display of ttls
in records)
                +[no]ttlunits        (Display TTLs in human-
readable units)
                +[no]unexpected      (Print replies from
unexpected sources
                                      default=off)
                +[no]unknownformat   (Print RDATA in RFC 3597
"unknown" format)
                +[no]vc              (TCP mode (+[no]tcp))
                +[no]yaml            (Present the results as
YAML)
                +[no]zflag           (Set Z flag in query)
       global d-opts and servers (before host name) affect
all queries.
       local d-opts and servers (after host name) affect only
that lookup.
       -h                            (print help and exit)
       -v                            (print version and exit)
```

The `whois` command

The `whois` command in Linux to find out information about a domain, such as the owner of the domain, the owner's contact information, and the nameservers that the domain is using.

Examples:

1. Performs a whois query for the domain name:

   ```
   whois {Domain_name}
   ```

2. -H option omits the lengthy legal disclaimers that many domain registries deliver along with the domain information.

   ```
   whois -H {Domain_name}
   ```

Syntax:

```
whois [ -h HOST ] [ -p PORT ] [ -aCFHlLMmrRSVx ] [ -g
SOURCE:FIRST-LAST ]
       [ -i ATTR ] [ -S SOURCE ] [ -T TYPE ] object
```

```
whois -t TYPE
```

```
whois -v TYPE
```

```
whois -q keyword
```

Additional Flags and their Functionalities:

Flag	Description
`-h HOST`, `--host HOST`	Connect to HOST.
`-H`	Do not display the legal disclaimers some registries like to show you.
`-p`, `--port PORT`	Connect to PORT.
`--verbose`	Be verbose.
`--help`	Display online help.
`--version`	Display client version information. Other options are flags understood by whois.ripe.net and some other RIPE-like servers.
`-a`	Also search all the mirrored databases.
`-b`	Return brief IP address ranges with abuse contact.
`-B`	Disable object filtering *(show the e-mail addresses)*
`-c`	Return the smallest IP address range with a reference to an irt object.
`-d`	Return the reverse DNS delegation object too.
`-g SOURCE:FIRST-LAST`	Search updates from SOURCE database between FIRST and LAST update serial number. It's useful to obtain Near Real Time Mirroring stream.
`-G`	Disable grouping of associated objects.

Flag	Description
-i ATTR[,ATTR]...	Search objects having associated attributes. ATTR is attribute name. Attribute value is positional OBJECT argument.
-K	Return primary key attributes only. Exception is members attribute of set object which is always returned. Another exceptions are all attributes of objects organisation, person, and role that are never returned.
-l	Return the one level less specific object.
-L	Return all levels of less specific objects.
-m	Return all one level more specific objects.
-M	Return all levels of more specific objects.
-q KEYWORD	Return list of keywords supported by server. KEYWORD can be version for server version, sources for list of source databases, or types for object types.
-r	Disable recursive look-up for contact information.
-R	Disable following referrals and force showing the object from the local copy in the server.
-s SOURCE[,SOURCE]...	Request the server to search for objects mirrored from SOURCES. Sources are delimited by comma and the order is significant. Use -q sources option to obtain list of valid sources.
-t TYPE	Return the template for a object of TYPE.
-T TYPE[,TYPE]...	Restrict the search to objects of TYPE. Multiple types are separated by a comma.
-v TYPE	Return the verbose template for a object of TYPE.
-x	Search for only exact match on network address prefix.

The ssh command

The `ssh` command in Linux stands for "Secure Shell". It is a protocol used to securely connect to a remote server/system. ssh is more secure in the sense that it transfers the data in encrypted form between the host and the client. ssh runs at TCP/IP port 22.

Examples:

1. Use a Different Port Number for SSH Connection:

```
ssh test.server.com -p 3322
```

2. -i ssh to remote server using a private key?

```
ssh -i private.key user_name@host
```

3. -l ssh specifying a different username

```
ssh -l alternative-username sample.ssh.com
```

Syntax:

```
ssh user_name@host(IP/Domain_Name)
```

```
ssh -i private.key user_name@host
```

```
ssh sample.ssh.com  ls /tmp/doc
```

Additional Flags and their Functionalities:

Flag	Description
-1	Forces ssh to use protocol SSH-1 only.
-2	Forces ssh to use protocol SSH-2 only.
-4	Allows IPv4 addresses only.
-A	Authentication agent connection forwarding is enabled..
-a	Authentication agent connection forwarding is disabled.
-B bind_interface	Bind to the address of bind_interface before attempting to connect to the destination host. This is only useful on systems with more than one address.
-b bind_address	Use bind_address on the local machine as the source address of the connection. Only useful on systems with more than one address.

Flag	Description
`-C`	Compresses all data (including stdin, stdout, stderr, and data for forwarded X11 and TCP connections) for a faster transfer of data.
`-c cipher_spec`	Selects the cipher specification for encrypting the session.
`-D [bind_address:]port`	Dynamic application-level port forwarding. This allocates a socket to listen to port on the local side. When a connection is made to this port, the connection is forwarded over the secure channel, and the application protocol is then used to determine where to connect to from the remote machine.
`-E log_file`	Append debug logs instead of standard error.
`-e escape_char`	Sets the escape character for sessions with a pty (default: '~'). The escape character is only recognized at the beginning of a line. The escape character followed by a dot ('.') closes the connection; followed by control-Z suspends the connection; and followed by itself sends the escape character once. Setting the character to "none" disables any escapes and makes the session fully transparent.

Flag	Description
`-F configfile`	Specifies a per-user configuration file. The default for the per-user configuration file is ~/.ssh/config.
`-f`	Requests ssh to go to background just before command execution.
`-G`	Causes ssh to print its configuration after evaluating Host and Match blocks and exit.
`-g`	Allows remote hosts to connect to local forwarded ports.
`-I pkcs11`	Specify the PKCS#11 shared library ssh should use to communicate with a PKCS#11 token providing keys.
`-i identity_file`	A file from which the identity key (private key) for public key authentication is read.
`-J [user@]host[:port]`	Connect to the target host by first making a ssh connection to the pjump host[(/iam/jump-host) and then establishing a TCP forwarding to the ultimate destination from there.
`-K`	Enables GSSAPI-based authentication and forwarding (delegation) of GSSAPI credentials to the server.
`-k`	Disables forwarding (delegation) of GSSAPI credentials to the server.

Flag	Description
`-L [bind_address:]port:host:hostport`, `-L [bind_address:]port:remote_socket`, `-L local_socket:host:hostport`, `-L local_socket:remote_socket`	Specifies that connections to the given TCP port or Unix socket on the local (client) host are to be forwarded to the given host and port, or Unix socket, on the remote side. This works by allocating a socket to listen to either a TCP port on the local side, optionally bound to the specified bind_address, or to a Unix socket. Whenever a connection is made to the local port or socket, the connection is forwarded over the secure channel, and a connection is made to either host port hostport, or the Unix socket remote_socket, from the remote machine.
`-l login_name`	Specifies the user to log in as on the remote machine.
`-M`	Places the ssh client into "master" mode for connection sharing. Multiple -M options places ssh into "master" mode but with confirmation required using ssh-askpass before each operation that changes the multiplexing state (e.g. opening a new session).
`-m mac_spec`	A comma-separated list of MAC (message authentication code) algorithms, specified in order of preference.

Flag	Description
`-N`	Do not execute a remote command. This is useful for just forwarding ports.
`-n`	Prevents reading from stdin.
`-O ctl_cmd`	Control an active connection multiplexing master process. When the -O option is specified, the ctl_cmd argument is interpreted and passed to the master process. Valid commands are: "check" (check that the master process is running), "forward" (request forwardings without command execution), "cancel" (cancel forwardings), "exit" (request the master to exit), and "stop" (request the master to stop accepting further multiplexing requests).
`-o`	Can be used to give options in the format used in the configuration file. This is useful for specifying options for which there is no separate command-line flag.
`-p, --port PORT`	Port to connect to on the remote host.

Flag	Description
`-Q query_option`	Queries ssh for the algorithms supported for the specified version 2. The available features are: cipher (supported symmetric ciphers), cipher-auth (supported symmetric ciphers that support authenticated encryption), help (supported query terms for use with the -Q flag), mac (supported message integrity codes), kex (key exchange algorithms), kex-gss (GSSAPI key exchange algorithms), key (keytypes), key-cert (certificate key types), key-plain (non-certificate key types), key-sig (all keytypes and signature algorithms), protocol-version (supported SSH protocol versions), and sig (supported signature algorithms). Alternatively, any keyword from ssh_config(5) or sshd_config(5) thattakes an algorithm list may be used as an alias for the corresponding query_option.
`-q`	Qiet mode. Causes most warning and diagnostic messages to be suppressed.
`-R [bind_address:]port:host:hostport, -R [bind_address:]port:local_socket, -R remote_socket:host:hostport, -R remote_socket:local_socket, -R [bind_address:]port`	Specifies that connections to the given TCP port or Unix socket on the remote (server) host are to be forwarded to the local side.

Flag	Description
`-S ctl_path`	Specifies the location of a control socket for connection sharing, or the string "none" to disable connection sharing.
`-s`	May be used to request invocation of a subsystem on the remote system. Subsystems facilitate the use of SSH as a secure transport for other applications (e.g. sftp(1)). The subsystem is specified as the remote command.
`-T`	Disable pseudo-terminal allocation.
`-t`	Force pseudo-terminal allocation. This can be used to execute arbitrary screen-based programs on a remote machine, which can be very useful, e.g. when implementing menu services. Multiple -t options force tty allocation, even if ssh has no local tty.
`-V`	Display the version number.
`-v`	Verbose mode. It echoes everything it is doing while establishing a connection. It is very useful in the debugging of connection failures.

Flag

Flag	Description
`-W host:port`	Requests that standard input and output on the client be forwarded to host on port over the secure channel. Implies -N, -T, ExitOnForwardFailure and ClearAllForwardings, though these can be overridden in the configuration file or using -o command line options.
`-w local_tun[remote_tun]`	Requests tunnel device forwarding with the specified tun devices between the client (local_tun) and the server (remote_tun). The devices may be specified by numerical ID or the keyword "any", which uses the next available tunnel device. If remote_tun is not specified, it defaults to "any". If the Tunnel directive is unset, it will be set to the default tunnel mode, which is "point-to-point". If a different Tunnel forwarding mode it desired, then it should be specified before -w.
`-X`	Enables X11 forwarding (GUI Forwarding).
`-x`	Disables X11 forwarding (GUI Forwarding).
`-Y`	Enables trusted X11 Forwarding.
`-y`	Send log information using the syslog system module. By default this information is sent to stderr.

The ᴀᴡᴋ command

Awk is a general-purpose scripting language designed for advanced text processing. It is mostly used as a reporting and analysis tool.

WHAT CAN WE DO WITH AWK?

1. AWK Operations: (a) Scans a file line by line (b) Splits each input line into fields (c) Compares input line/fields to pattern (d) Performs action(s) on matched lines

2. Useful For: (a) Transform data files (b) Produce formatted reports

3. Programming Constructs: (a) Format output lines (b) Arithmetic and string operations (c) Conditionals and loops

Syntax

```
awk options 'selection _criteria {action }' input-file >
output-file
```

Example

Consider the following text file as the input file for below example:

```
$cat > employee.txt
```

```
ajay manager account 45000
sunil clerk account 25000
varun manager sales 50000
amit manager account 47000
tarun peon sales 15000
```

1. Default behavior of Awk: By default Awk prints every line of data from the specified file.

```
$ awk '{print}' employee.txt
```

```
ajay manager account 45000
sunil clerk account 25000
varun manager sales 50000
amit manager account 47000
tarun peon sales 15000
```

In the above example, no pattern is given. So the actions are applicable to all the lines. Action print without any argument prints the whole line by default, so it prints all the lines of the file without failure.

2. Print the lines which match the given pattern.

```
awk '/manager/ {print}' employee.txt
```

```
ajay manager account 45000
varun manager sales 50000
amit manager account 47000
```

In the above example, the awk command prints all the line which matches with the 'manager'.

3. Splitting a Line Into Fields : For each record i.e line, the awk command splits the record delimited by whitespace character by default and stores it in the $n variables. If the line has 4 words, it will be stored in $1, $2, $3 and $4 respectively. Also, $0 represents the whole line.

```
$ awk '{print $1,$4}' employee.txt
```

```
ajay 45000
sunil 25000
varun 50000
amit 47000
tarun 15000
```

Built-In Variables In Awk

Awk's built-in variables include the field variables—$1, $2, $3, and so on ($0 is the entire line) — that break a line of text into individual words or pieces called fields.

NR: NR command keeps a current count of the number of input records. Remember that records are usually lines. Awk command performs the pattern/action statements once for each record in a file. NF: NF command keeps a count of the number of fields within the current input record. FS: FS command contains the field separator character which is used to divide fields on the input line. The default is "white space",

meaning space and tab characters. FS can be reassigned to another character (typically in BEGIN) to change the field separator. RS: RS command stores the current record separator character. Since, by default, an input line is the input record, the default record separator character is a newline. OFS: OFS command stores the output field separator, which separates the fields when Awk prints them. The default is a blank space. Whenever print has several parameters separated with commas, it will print the value of OFS in between each parameter. ORS: ORS command stores the output record separator, which separates the output lines when Awk prints them. The default is a newline character. print automatically outputs the contents of ORS at the end of whatever it is given to print.

The `crontab` command

`crontab` is used to maintain crontab files for individual users (Vixie Cron)

crontab is the program used to install, uninstall or list the tables used to drive the cron(8) daemon in Vixie Cron. Each user can have their own crontab, and though these are files in `/var/spool/cron/crontabs`, they are not intended to be edited directly.

Syntax:

```
crontab [ -u user ] file
crontab [ -u user ] [ -i ] { -e | -l | -r }
```

Examples:

1. The -l option causes the current crontab to be displayed on standard output.

```
crontab -l
```

2. The -r option causes the current crontab to be removed.

```
crontab -r
```

3. The -e option is used to edit the current crontab using the editor

specified by the VISUAL or EDITOR environment variables. After you exit from the editor, the modified crontab will be installed automatically. If neither of the environment variables is defined, then the default editor /usr/bin/editor is used.

```
crontab -e
```

4. You can specify the user you want to edit the crontab for. Every user has its own crontab. Assume you have a `www-data` user, which is in fact the user Apache is default running as. If you want to edit the crontab for this user you can run the following command

```
crontab -u www-data -e
```

Help Command

Run below command to view the complete guide to `crontab` command.

```
man crontab
```

The `xargs` command

`xargs` is used to build and execute command lines from standard input

Some commands like grep can accept input as parameters, but some commands accepts arguments, this is place where xargs came into picture.

Syntax:

```
xargs [options] [command [initial-arguments]]
```

Options:

```
-0, --null
```

Input items are terminated by a null character instead of by whitespace, and the quotes and backslash are not special (every character is taken literal-ly). Disables the end of file string, which is treated like any other argument. Useful when input items might contain white space, quote marks, or back-slashes.

```
-a file, --arg-file=file
```

Read items from file instead of standard input. If you use this option, stdin remains unchanged when commands are run. Otherwise, stdin is

redirected from /dev/null.

```
-o, --open-tty
```

Reopen stdin as /dev/tty in the child process before executing the command. This is useful if you want xargs to run an interactive application.

```
--delimiter=delim, -d delim
```

Input items are terminated by the specified character. The specified delimiter may be a single character, a C-style character escape such as \n, or an octal or hexadecimal escape code. Octal and hexadecimal escape codes are understood as for the printf command. Multibyte characters are not supported. When processing the input, quotes and backslash are not special; every character in the input is taken literally. The -d option disables any end-of-file string, which is treated like any other argument. You can use this option when the input consists of simply newline-separated items, although it is al- most always better to design your program to use --null where this is possible.

```
-p, --interactive
```

Prompt the user about whether to run each command line and read a line from the terminal. Only run the command line if the response starts with `y'` `or` Y'. Implies -t.

Examples:

```
find /tmp -name core -type f -print | xargs /bin/rm -f
```

Find files named core in or below the directory /tmp and delete them. Note that this will work incorrectly if there are any filenames containing newlines or spaces.

```
find /tmp -name core -type f -print0 | xargs -0 /bin/rm -f
```

Find files named core in or below the directory /tmp and delete them, processing filenames in such a way that file or directory names containing spaces or new- lines are correctly handled.

```
find /tmp -depth -name core -type f -delete
```

Find files named core in or below the directory /tmp and delete them, but more efficiently than in the previous example (because we avoid the need to use fork(2) and exec(2) to launch rm and we don't need the extra xargs process).

```
cut -d: -f1 < /etc/passwd | sort | xargs echo
```

Generates a compact listing of all the users on the system.

Help Command

Run below command to view the complete guide to xargs command.

```
man xargs
```

The nohup command

When a shell exits (maybe while logging out of an SSH session), the HUP ('hang up') signal is send to all of its child processes, causing them to terminate. If you require a long-running process to continue after exiting shell, you'll need the nohup command. Prefixing any command with nohup causes the command to become *immune* to HUP signals. Additionally, STDIN is being ignored and all output gets redirected to local file `./nohup.out`.

Examples:

1. Applying nohup to a long-running debian upgrade:

```
nohup apt-get -y upgrade
```

Syntax:

```
nohup COMMAND [ARG]...
nohup OPTION
```

The `pstree` command

The `pstree` command is similar to `ps`, but instead of listing the running processes, it shows them as a tree. The tree-like format is sometimes more suitable way to display the processes hierarchy which is a much simpler way to visualize running processes. The root of the tree is either init or the process with the given pid.

Examples

1. To display a hierarchical tree structure of all running processes:

```
pstree
```

2. To display a tree with the given process as the root of the tree:

```
pstree [pid]
```

3. To show only those processes that have been started by a user:

```
pstree [USER]
```

4. To show the parent processes of the given process:

```
pstree -s [PID]
```

5. To view the output one page at a time, pipe it to the `less` command:

```
pstree | less
```

Syntax

```
ps [OPTIONS] [USER or PID]
```

Additional Flags and their Functionalities

Short Flag	Long Flag	Description
-a	--arguments	Show command line arguments
-A	--ascii	use ASCII line drawing characters
-c	--compact	Don't compact identical subtrees
-h	--highlight-all	Highlight current process and its ancestors
-H PID	--highlight-pid=PID	highlight this process and its ancestors
-g	--show-pgids	show process group ids; implies `-c`
-G	--vt100	use VT100 line drawing characters
-l	--long	Don't truncate long lines
-n	--numeric-sort	Sort output by PID
-N type	--ns-sort=type	Sort by namespace type (cgroup, ipc, mnt, net, pid, user, uts)
-p	--show-pids	show PIDs; implies -c
-s	--show-parents	Show parents of the selected process
-S	--ns-changes	show namespace transitions
-t	--thread-names	Show full thread names
-T	--hide-threads	Hide threads, show only processes
-u	--uid-changes	Show uid transitions

Short Flag	Long Flag	Description
-U	--unicode	Use UTF-8 (Unicode) line drawing characters
-V	--version	Display version information
-Z	--security-context	Show SELinux security contexts

The `tree` command

The `tree` command in Linux recursively lists directories as tree structures. Each listing is indented according to its depth relative to root of the tree.

Examples:

1. Show a tree representation of the current directory.

```
tree
```

2. -L NUMBER limits the depth of recursion to avoid display very deep trees.

```
tree -L 2 /
```

Syntax:

```
tree  [-acdfghilnpqrstuvxACDFQNSUX]  [-L  level [-R]] [-H
baseHREF] [-T title]
      [-o filename] [--nolinks] [-P pattern] [-I  pattern]  [-
-inodes]
      [--device] [--noreport] [--dirsfirst] [--version] [--
help] [--filelimit #]
      [--si] [--prune] [--du] [--timefmt  format]  [--
matchdirs]  [--from-file]
      [--] [directory ...]
```

Additional Flags and their Functionalities:

Flag	Description
-a	Print all files, including hidden ones.
-d	Only list directories.
-l	Follow symbolic links into directories.
-f	Print the full path to each listing, not just its basename.
-x	Do not move across file-systems.
-L #	Limit recursion depth to #.
-P REGEX	Recurse, but only list files that match the REGEX.
-I REGEX	Recurse, but do not list files that match the REGEX.
--ignore-case	Ignore case while pattern-matching.
--prune	Prune empty directories from output.
--filelimit #	Omit directories that contain more than # files.
-o FILE	Redirect STDOUT output to FILE.
-i	Do not output indentation.

The `whereis` command

The `whereis` command is used to find the location of source/binary file of a command and manuals sections for a specified file in Linux system. If we compare `whereis` command with find command they will appear similar to each other as both can be used for the same purposes but `whereis` command produces the result more accurately by consuming less time comparatively.

Points to be kept on mind while using the whereis command:

Since the `whereis` command uses chdir(change directory 2V) to give you the result in the fastest possible way, the pathnames given with the -M, -S, or -B must be full and well-defined i.e. they must begin with a `/` and should be a valid path that exist in the system's directories, else it exits without any valid result. `whereis` command has a hard-coded(code which is not dynamic and changes with specification) path, so you may not always find what you're looking for.

Syntax

```
whereis [options] [filename]
```

Options

-b : This option is used when we only want to search for binaries. -m : This option is used when we only want to search for manual sections. -s : This option is used when we only want to search for source files. -u:

This option search for unusual entries. A source file or a binary file is said to be unusual if it does not have any existence in system as per [-bmsu] described along with "–u". Thus `whereis -m -u *' asks for those files in the current directory which have unsual entries.

-B : This option is used to change or otherwise limit the places where whereis searches for binaries. -M : This option is used to change or otherwise limit the places where whereis searches for manual sections. -S : This option is used to change or otherwise limit the places where whereis searches for source files.

-f : This option simply terminate the last directory list and signals the start of file names. This must be used when any of the -B, -M, or -S options are used. -V: Displays version information and exit. -h: Displays the help and exit.

The `printf` command

This command lets you print the value of a variable by formatting it using rules. It is pretty similar to the printf in C language.

Syntax:

```
$printf [-v variable_name] format [arguments]
```

Options:

OPTION	Description
FORMAT	FORMAT controls the output, and defines the way that the ARGUMENTs will be expressed in the output
ARGUMENT	An ARGUMENT will be inserted into the formatted output according to the definition of FORMAT
--help	Display help and exit
--version	Output version information adn exit

Formats:

The anatomy of the FORMAT string can be extracted into three different parts,

- *ordinary characters*, which are copied exactly the same characters as were used originally to the output.
- *interpreted character* sequences, which are escaped with a backslash ("\").

- *conversion specifications*, this one will define the way the ARGUMENTs will be expressed as part of the output.

You can see those parts in this example,

```
printf " %s is where over %d million developers shape \"the
future of sofware.\" " Github 65
```

The output:

```
Github is where over 65 million developers shape "the future
of sofware."
```

There are two conversion specifications `%s` and `%d`, and there are two escaped characters which are the opening and closing double-quotes wrapping the words of *the future of software*. Other than that are the ordinary characters.

Conversion Specifications:

Each conversion specification begins with a `%` and ends with a `conversion character`. Between the `%` and the `conversion character` there may be, in order:

`-`	A minus sign. This tells printf to left-adjust the conversion of the argument
number	An integer that specifies field width; printf prints a conversion of ARGUMENT in a field at least number characters wide. If necessary it will be padded on the left (or right, if left-adjustment is called for) to make up the field width
`.`	A period, which separates the field width from the precision

number	An integer, the precision, which specifies the maximum number of characters to be printed from a string, or the number of digits after the decimal point of a floating-point value, or the minimum number of digits for an integer
`h` or `l`	These differentiate between a short and a long integer, respectively, and are generally only needed for computer programming

The conversion characters tell `printf` what kind of argument to print out, are as follows:

Conversion char	Argument type
`s`	A string
`c`	An integer, expressed as a character corresponds ASCII code
`d`, `i`	An integer as a decimal number
`o`	An integer as an unsigned octal number
`x`, `X`	An integer as an unsigned hexadecimal number
`u`	An integer as an unsigned decimal number
`f`	A floating-point number with a default precision of 6
`e`, `E`	A floating-point number in scientific notation
`p`	A memory address pointer
`%`	No conversion

Here is the list of some examples of the `printf` output the ARGUMENT. we can put any word but in this one we put a 'linuxcommand` word and enclosed it with quotes so we can see easier the position related to the whitespaces.

FORMAT string	ARGUMENT string	Output string
`"%s"`	`"linuxcommand"`	"linuxcommand"
`"%5s"`	`"linuxcommand"`	"linuxcommand"
`"%.5s"`	`"linuxcommand"`	"linux"
`"%-8s"`	`"linuxcommand"`	"linuxcommand"

FORMAT string	ARGUMENT string	Output string
"%-15s"	"linuxcommand"	"linuxcommand "
"%12.5s"	"linuxcommand"	" linux"
"%-12.5"	"linuxcommand"	"linux "
"%-12.4"	"linuxcommand"	"linu "

Notes:

- `printf` requires the number of conversion strings to match the number of ARGUMENTs
- `printf` maps the conversion strings one-to-one, and expects to find exactly one ARGUMENT for each conversion string
- Conversion strings are always interpreted from left to right.

Here's the example:

The input

```
printf "We know %f is %s %d" 12.07 "larger than" 12
```

The output:

```
We know 12.070000 is larger than 12
```

The example above shows 3 arguments, *12.07*, *larger than*, and *12*. Each of them interpreted from left to right one-to-one with the given 3 conversion strings (`%f`, `%d`, `%s`).

Character sequences which are interpreted as special characters by `printf`:

Escaped char	Description
\a	issues an alert (plays a bell). Usually ASCII BEL characters
\b	prints a backspace
\c	instructs `printf` to produce no further output
\e	prints an escape character (ASCII code 27)
\f	prints a form feed
\n	prints a newline
\r	prints a carriage return
\t	prints a horizontal tab
\v	prints a vertical tab
\"	prints a double-quote (")
\\	prints a backslash ()
\NNN	prints a byte with octal value NNN (1 to 3 digits)
\xHH	prints a byte with hexadecimal value HH (1 to 2 digits)
\uHHHH	prints the unicode character with hexadecimal value HHHH (4 digits)
\UHHHHHHHH	prints the unicode character with hexadecimal value HHHHHHHH (8 digits)
%b	prints ARGUMENT as a string with "\" escapes interpreted as listed above, with the exception that octal escapes take the form \0 or \0NN

Examples:

The format specifiers usually used with printf are stated in the examples below:

- %s

```
$printf "%s\n" "Printf command documentation!"
```

This will print `Printf command documentation!` in the shell.

357

Other important attributes of printf command:

- `%b` - Prints arguments by expanding backslash escape sequences.
- `%q` - Prints arguments in a shell-quoted format which is reusable as input.
- `%d` , `%i` - Prints arguments in the format of signed decimal integers.
- `%u` - Prints arguments in the format of unsigned decimal integers.
- `%o` - Prints arguments in the format of unsigned octal(base 8) integers.
- `%x`, `%X` - Prints arguments in the format of unsigned hexadecimal(base 16) integers. %x prints lower-case letters and %X prints upper-case letters.
- `%e`, `%E` - Prints arguments in the format of floating-point numbers in exponential notation. %e prints lower-case letters and %E prints upper-case.
- `%a`, `%A` - Prints arguments in the format of floating-point numbers in hexadecimal(base 16) fractional notation. %a prints lower-case letters and %A prints upper-case.
- `%g`, `%G` - Prints arguments in the format of floating-point numbers in normal or exponential notation, whichever is more appropriate for the given value and precision. %g prints lower-case letters and %G prints upper-case.
- `%c` - Prints arguments as single characters.
- `%f` - Prints arguments as floating-point numbers.
- `%s` - Prints arguments as strings.
- `%%` - Prints a "%" symbol.

More Examples:

The input:

```
printf 'Hello\nyoung\nman!'
```

The output:

```
hello
young
man!
```

The two `\n` break the sentence into 3 parts of words.

The input:

```
printf "%f\n" 2.5 5.75
```

The output

```
2.500000
5.750000
```

The `%f` specifier combined with the `\n` interpreted the two arguments in the form of floating point in the seperated new lines.

The cut command

The cut command lets you remove sections from each line of files. Print selected parts of lines from each FILE to standard output. With no FILE, or when FILE is -, read standard input.

Usage and Examples:

1. Selecting specific fields in a file

```
cut -d "delimiter" -f (field number) file.txt
```

2. Selecting specific characters:

```
cut -c [(k)-(n)/(k),(n)/(n)] filename
```

Here, **k** denotes the starting position of the character and **n** denotes the ending position of the character in each line, if *k* and *n* are separated by "-" otherwise they are only the position of character in each line from the file taken as an input.

3. Selecting specific bytes:

```
cut -b 1,2,3 filename                    //select bytes 1,2
and 3
cut -b 1-4 filename                      //select bytes 1
through 4
cut -b 1- filename                       //select bytes
1 through the end of file
cut -b -4 filename                       //select bytes
from the beginning till the 4th byte
```

Tabs and backspaces are treated like as a character of 1 byte.

Syntax:

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

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-b	--bytes=LIST	select only these bytes
-c	--characters=LIST	select only these characters
-d	--delimiter=DELIM	use DELIM instead of TAB for field delimiter
-f	--fields	select only these fields; also print any line that contains no delimiter character, unless the -s option is specified
-s	--only-delimited	do not print lines not containing delimiters
-z	--zero-terminated	line delimiter is NUL, not newline

The `sed` command

`sed` command stands for stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). For instance, it can perform lot's of functions on files like searching, find and replace, insertion or deletion. While in some ways it is similar to an editor which permits scripted edits (such as `ed`), `sed` works by making only one pass over the input(s), and is consequently more efficient. But it is sed's ability to filter text in a pipeline that particularly distinguishes it from other types of editors.

The most common use of `sed` command is for a substitution or for find and replace. By using sed you can edit files even without opening it, which is a much quicker way to find and replace something in the file. It supports basic and extended regular expressions that allow you to match complex patterns. Most Linux distributions come with GNU and `sed` is pre-installed by default.

Examples:

1. To Find and Replace String with `sed`

```
sed -i 's/{search_regex}/{replace_value}/g' input-file
```

2. For Recursive Find and Replace *(along with `find`)*

Sometimes you may want to recursively search directories for files containing a string and replace the string in all files. This can be done using commands such as find to recursively find files in the directory and piping the file names to `sed`. The following command will recursively search for files in the current working directory and pass the file names to `sed`. It will recursively search for files in the current working directory and pass the file names to `sed`.

```
find . -type f -exec sed -i
's/{search_regex}/{replace_value}/g' {} +
```

Syntax:

```
sed [OPTION]... {script-only-if-no-other-script} [INPUT-
FILE]...
```

- `OPTION` - sed options in-place, silent, follow-symlinks, line-length, null-data ...etc.
- `{script-only-if-no-other-script}` - Add the script to command if available.
- `INPUT-FILE` - Input Stream, A file or input from a pipeline.

If no option is given, then the first non-option argument is taken as the sed script to interpret. All remaining arguments are names of input files; if no input files are specified, then the standard input is read.

GNU sed home page: http://www.gnu.org/software/sed/

Short Flag	Long Flag	Description
`-i[SUFFIX]`	--in-place[=SUFFIX]	Edit files in place (makes backup if SUFFIX supplied).

Short Flag	Long Flag	Description
`-n`	--quiet, --silent	Suppress automatic printing of pattern space.
`-e script`	--expression=script	Add the script to the commands to be executed.
`-f script-file`	--file=script-file	Add the contents of script-file to the commands to be executed.
`-l N`	--line-length=N	Specify the desired line-wrap length for the `l` command.
`-r`	--regexp-extended	Use extended regular expressions in the script.
`-s`	--separate	Consider files as separate rather than as a single continuous long stream.
`-u`	--unbuffered	Load minimal amounts of data from the input files and flush the output buffers more often.
`-z`	--null-data	Separate lines by NULL characters.

Before you begin

It may seem complicated and complex at first, but searching and replacing text in files with sed is very simple.

To find out more: https://www.gnu.org/software/sed/manual/sed.html

The `vim` command

The [vim](#) is a text editor for Unix that comes with Linux, BSD, and macOS. It is known to be fast and powerful, partly because it is a small program that can run in a terminal (although it has a graphical interface). Vim text editor is developed by [Bram Moolenaar](#). It supports most file types and the vim editor is also known as a programmer's editor. It is mainly because it can be managed entirely without menus or a mouse with a keyboard.

Note: Do not confuse `vim` with `vi`. `vi`, which stands for "Visual", is a text editor that was developed by [Bill Joy](#) in 1976. `vim` stands for "Vi Improved", and is an improved clone of the `vi` editor.

The most searched question about `vim` :

How to exit vim editor?

The most searched question about vim editor looks very funny but it's true that the new user gets stuck at the very beginning when using vim editor.

The command to save the file and exit vim editor: `:wq` or `:x`

The command to exit vim editor without saving the file: `:q!`

Fun reading:

Here's a [survey](#) for the same question, look at this and do not think to quit the vim editor.

Installation:

First check if vim is already installed or not, enter the following command:

```
vim --version
```

If it is already installed it will show its version, else we can run the below commands for the installations:

On Ubuntu/Debian:

```
sudo apt-get install vim
```

On Arch:

```
sudo pacman -S vim
```

On CentOS/Fedora:

```
sudo yum install vim
```

If you want to use advanced features on CentOS/Fedora, you'll need to install enhanced vim editor, to do this run the following command:

```
sudo yum install -y vim-enhanced
```

On macOS:

```
brew install vim
```

Syntax:

```
vim [FILE_PATH/FILE_NAME]
```

Examples:

1. To open the file named "demo.txt" from your current directory:

```
vim demo.txt
```

2. To open the file in a specific directory:

```
vim {File_Path/filename}
```

3. To open the file starting on a specific line in the file:

```
vim {File_Path/filename} +LINE_NUMBER
```

Modes in vim editor:

There are some arguments as to how many modes that vim has, but the modes you're most likely to use are `command mode` and `insert mode`. These modes will allow you to do just about anything you need, including creating your document, saving your document, and doing advanced editing, including taking advantage of search and replace

functions.

Workflow of `vim` editor:

1. Open a new or existing file with `vim filename`.
2. Type `i` to switch into insert mode so that you can start editing the file.
3. Enter or modify the text of your file.
4. When you're done, press the `Esc` key to exit insert mode and back to command mode.
5. Type :w or :wq to save the file or save and exit from the file respectively.

Navigate in vim

Some common commands:

- `j` : move down one line
- `k` : move up one line
- `h` : Move left one character
- `l` : move right one character
- `w` : move forward one word
- `b` : move backward one word
- `e` : Move to the end of your word
- `0` : move to begining of line
- `$` : move to end of line
- `gg` : go to begining of file
- `G` : go to end of file
- `:linenumber` : go to a specific line number

Copy, Paste and Delete

1. Copy(Yank): Copying in vim is called "yanking":

   - `yy` : yank (copy) the current line
   - `2yy` : yank 2 lines
   - `y$` : yank from cursor to end of line
   - `y^` : yank from cursor to begining of line
   - `yw` : yank one word
   - `y}` : yank until end of paragraph

2. Paste:

   - `p` : paste after the cursor
   - `P` : paste before the cursor

3. Delete:

   - `x` : delete a single character
   - `dd` : delete the whole current line
   - `2dd` : delete 2 lines (or use any number `n  dd`)
   - `d$` : delete from cursor to end of line
   - `d^` : delete from cursor to begining of line
   - `dG` : delete from cursor to end of file
   - `dgg`: delete from cursor to begining of file
   - `dw` : delete from cursor to end of word
   - `di"` : delete inside double quotes
   - `diw` : delete inner word (without spaces)
   - `dip` : delete inner paragraph (no newline)

Selection (visual mode)

   - `v` : start character-wise selection
   - `V` : start line-wise selection

- `ctrl + v` : start block-wise selection

Interactive training

In this interactive tutorial, you will learn the different ways to use the `vim` command:

[The Open vim Tutorial](#)

Additional Flags and their Functionalities:

Flags/Options	Description
`-e`	Start in Ex mode (see [Ex-mode](#))
`-R`	Start in read-only mode
`-R`	Start in read-only mode
`-g`	Start the [GUI](#)
`-eg`	Start the GUI in Ex mode
`-Z`	Like "vim", but in restricted mode
`-d`	Start in diff mode [diff-mode](#)
`-h`	Give usage (help) message and exit
`+NUMBER`	Open a file and place the cursor on the line number specified by NUMBER

Read more about vim:

vim can not be learned in a single day, use in day-to-day tasks to get hands-on in vim editor.

To learn more about `vim` follow the given article:

[Article By Daniel Miessler](#)

The chown command

The chown command makes it possible to change the ownership of a file or directory. Users and groups are fundamental in Linux, with chown you can change the owner of a file or directory. It's also possible to change ownership on folders recursively

Examples:

1. Change the owner of a file

```
chown user file.txt
```

2. Change the group of a file

```
chown :group file.txt
```

3. Change the user and group in one line

```
chown user:group file.txt
```

4. Change to ownership on a folder recursively

```
chown -R user:group folder
```

Syntax:

372

```
chown [-OPTION] [DIRECTORY_PATH]
```

The `find` command

The `find` command lets you **search for files in a directory hierarchy**

- Search a file with specific name.
- Search a file with pattern
- Search for empty files and directories.

Examples:

1. Search a file with specific name:

```
find ./directory1 -name sample.txt
```

2. Search a file with pattern:

```
find ./directory1 -name '*.txt'
```

3. To find all directories whose name is test in / directory.

```
find / -type d -name test
```

4. Searching empty files in current directory

```
find . -size 0k
```

Syntax:

374

```
find [options] [paths] [expression]
```

In Simple words

```
find [where to start searching from]
  [expression determines what to find] [-options] [what to
  find]
```

Additional Flags and their Functionalities:

Commonly-used primaries include:

- `name` pattern - tests whether the file name matches the shell-glob pattern given.
- `type` type - tests whether the file is a given type. Unix file types accepted include:

options	Description
b	block device (buffered)
d	directory
f	regular file
l	Symbolic link
-print	always returns true; prints the name of the current file plus a newline to the stdout.
-mtime n	find's all the files which are modified n days back.
-atime n	find's all the files which are accessed 50 days back.
-cmin n	find's all the files which are modified in the last 1 hour.
-newer file	find's file was modified more recently than file.
-size n	File uses n units of space, rounding up.

Help Command

375

Run below command to view the complete guide to `find` command or
<u>click here</u>.

```
man find
```

The `rmdir` command

The **rmdir** command is used to remove empty directories from the filesystem in Linux. The rmdir command removes each and every directory specified in the command line only if these directories are empty.

Usage and Examples:

1. remove directory and its ancestors

```
rmdir -p a/b/c                          // is similar to 'rmdir
a/b/c a/b a'
```

2. remove multiple directories

```
rmdir a b c                             // removes empty
directories a,b and c
```

Syntax:

```
rmdir [OPTION]... DIRECTORY...
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-	--ignore-fail-on-non-empty	ignore each failure that is solely because a directory is non-empty
-p	--parents	remove DIRECTORY and its ancestors
-d	--delimiter=DELIM	use DELIM instead of TAB for field delimiter
-v	--verbose	output a diagnostic for every directory processed

The `lsblk` command

Summary

379

The `lsblk` command displays the block and loop devices on the system. It is especially useful when you want to format disks, write filesystems, check the filesystem and know the mount point of a device.

Examples

1. Basic usage is fairly simple - just execute 'lsblk' sans any option.

```
lsblk
```

2. Make lsblk display empty devices as well

```
lsblk -a
```

3. Make lsblk print size info in bytes

```
lsblk -b
```

4. Make lsblk print zone model for each device

```
lsblk -z
```

5. Make lsblk skip entries for slaves

```
lsblk -d
```

6. Make lsblk use ascii characters for tree formatting

```
lsblk -i
```

7. Make lsblk display info about device owner, group, and mode

```
lsblk -m
```

8. Make lsblk output select columns

```
lsblk -o NAME,SIZE
```

Syntax

382

```
lsblk [options] [<device> ...]
```

Reading information given by `lsblk`

On running `lsblk` with no flags or command-line arguments, it writes general disk information to the STDOUT. Here is a table that interpretes that information:

Column Name	Meaning	Interpretation
NAME	Name of the device.	Shows name of the device.
RM	Removable.	Shows 1 if the device is removable, 0 if not.
SIZE	Size of the device.	Shows size of the device.
RO	Read-Only.	Shows 1 if read-only, 0 if not.
TYPE	The type of block or loop device.	Shows `disk` for entire disk and `part` for partitions.
MOUNTPOINTS	Where the device is mounted.	Shows where the device is mounted. Empty if not mounted.

Reading information of a specific device

`lsblk` can display information of a specific device when the device's absolute path is passed to it. For example, `lsblk` command for displaying the information of the `sda` disk is:

```
lsblk /dev/sda
```

Useful flags for `lsblk`

Here is a table that show some of the useful flags that can be used with lsblk

Short Flag	Long Flag	Description
-a	--all	`lsblk` does not list empty devices by default. This option disables this restriction.
-b	--bytes	Print the SIZE column in bytes rather than in human-readable format.
-d	--nodeps	Don't print device holders or slaves.
-D	--discard	Print information about the discard (TRIM, UNMAP) capabilities for each device.
-E	--dedup column	Use column as a de-duplication key to de-duplicate output tree. If the key is not available for the device, or the device is a partition and parental whole-disk device provides the same key than the device is always printed.
-e	--exclude list	xclude the devices specified by a comma-separated list of major device numbers. Note that RAM disks (major=1) are excluded by default. The filter is applied to the top-level devices only.
-f	--fs	Displays information about filesystem.
-h	--help	Print a help text and exit.
-l	--include list	Displays all the information in List Format.
-J	--json	Displays all the information in JSON Format.
-l	--list	Displays all the information in List Format.
-m	--perms	Displays info about device owner, group and mode.
-M	--merge	Group parents of sub-trees to provide more readable output for RAIDs and Multi-path devices. The tree-like output is required.
-n	--noheadings	Do not print a header line.
-o	--output list	Specify which output columns to print. Use `--help` to get a list of all supported columns.
-O	--output-all	Displays all available columns.
-p	--paths	Displays absolute device paths.
-P	--pairs	Use key="value" output format. All potentially unsafe characters are hex-escaped (`\x`)
-r	--raw	Use the raw output format. All potentially unsafe characters are hex-escaped (`\x`) `in NAME, KNAME, LABEL, PARTLABEL and MOUNTPOINT columns.`
-S	--scsi	Output info about SCSI devices only. All partitions, slaves and holder devices are ignored.
-s	--inverse	Print dependencies in inverse order.
-t	--topology	Output info about block device topology. This option is equivalent to "-o NAME,ALIGNMENT,MIN-IO,OPT-IO,PHY-SEC,LOG-SEC,ROTA,SCHED,RQ-SIZE".
-T	--tree[=column]	Displays all the information in Tree Format.
-V	--version	Output version information and exit.

Short Flag	Long Flag	Description
-w	--width	pecifies output width as a number of characters. The default is the number of the terminal columns, and if not executed ona terminal, then output width is not restricted at all by default.
-x	--sort [column]	Sort output lines by column. This option enables `--list` output format by default. It is possible to use the option `--tree` to force tree-like output and than the tree branches are sorted by the column.
-z	--zoned	Print the zone model for each device.
-	--sysroot directory	Gather data for a Linux instance other than the instance from which the lsblk command is issued. The specified directory is the system root of the Linux instance to be inspected.

Short Flag	Long Flag	Description

Exit Codes

387

Like every Unix / Linux Program, `lslbk` returns an exit code to the environment. Here is a table of all the exit codes.

Exit Code	Meaning
0	Exit with success.
1	Exit with failure.
32	Specified device(s) not found.
64	Some of the specified devices were found while some not.

The `cmatrix` command

This command doesn't come by default in Linux. It has to be installed, and as seen in chapter 052 we need to run the following command:

```
sudo apt-get install cmatrix
```

And after everything is installed, you have become a 'legit hacker'. In order to use this command, just type in `cmatrix` and press enter:

```
cmatrix
```

And this is what you should see:

As you can see you have access to the matrix now. Well, not really.

What this actually is just a fun little command to goof around with. There are actually a few options you can use. For examle you can change the text colour. You can choose from **green, red, blue, white, yellow, cyan, magenta and black**.

```
cmatrix -C red
```


And the falling characters will be red. This command isn't really something that will help you with your job or anything, but it is fun to

know that you can have some fun in Linux.

The `chmod` command

The `chmod` command allows you to change the permissions on a file using either a symbolic or numeric mode or a reference file.

Examples:

1. Change the permission of a file using symbolic mode:

```
chmod u=rwx,g=rx,o=r myfile
```

The command above means :

- user can read, write, execute `myfile`
- group can read, execute `myfile`
- other can read `myfile`

2. Change the permission of a file using numeric mode

```
chmod 754 myfile user:group file.txt
```

The command above means :

- user can read, write, execute `myfile`
- group can read, execute `myfile`
- other can read `myfile`

3. Change the permission of a folder recursively

```
chmod -R 754 folder
```

Syntax:

```
chmod [OPTIONS] MODE FILE(s)
```

- [OPTIONS] : -R: recursive, mean all file inside directory

- MODE: different way to set permissions:

- **Symbolic mode explained**

  - u: user
  - g: group
  - o: other
  - =: set the permission
  - r: read
  - w: write
  - x: execute
  - example u=rwx means user can read write and execute

- **Numeric mode explained**:

The **numeric mode** is based off of a binary representation of the permissions for user, group, and others, for more information please look at this explanation from Digital Ocean's community section:

- 4 stands for "read",
- 2 stands for "write",

- 1 stands for "execute", and
- 0 stands for "no permission."
- example 7 mean read + write + execute

The grep command

The `grep` filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. grep stands for globally search for regular expression and print out. The pattern that is searched in the file is referred to as the regular expression.

Examples:

1. To search the contents of the destination.txt file for a string("KeY") case insensitively.

```
grep -i "KeY" destination.txt
```

2. Displaying the count of number of matches

```
grep -c "key" destination.txt
```

3. We can search multiple files and only display the files that contains the given string/pattern.

```
grep -l "key" destination1.txt destination2.txt
destination3.xt destination4.txt
```

4. To show the line number of file with the line matched.

```
grep -n "key" destination.txt
```

5. If you want to grep the monitored log files, you can add the `--line-buffered` to search them in real time.

```
tail -f destination.txt | grep --line-buffered "key"
```

Syntax:

The general syntax for the grep command is as follows:

```
grep [options] pattern [files]
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-c	--count	print a count of matching lines for each input file
-h	--no-filename	Display the matched lines, but do not display the filenames
-i	--ignore-case	Ignores, case for matching
-l	--files-with-matches	Displays list of a filenames only.
-n	--line-number	Display the matched lines and their line numbers.
-v	--invert-match	This prints out all the lines that do not matches the pattern
-e	--regexp=	Specifies expression with this option. Can use multiple times
-f	--file=	Takes patterns from file, one per line.

Short Flag	Long Flag	Description
-F	--fixed-strings=	Interpret patterns as fixed strings, not regular expressions.
-E	--extended-regexp	Treats pattern as an extended regular expression (ERE)
-w	--word-regexp	Match whole word
-o	--only-matching	Print only the matched parts of a matching line, with each such part on a separate output line.
	--line-buffered	Force output to be line buffered.

The `screen` command

`screen` - With screen you can start a screen session and then open any number of windows (virtual terminals) inside that session. Processes running in Screen will continue to run when their window is not visible even if you get disconnected. This is very handy for running long during session such as bash scripts that run very long.

To start a screen session you type `screen`, this will open a new screen session with a virtual terminal open.

Below are some most common commands for managing Linux Screen Windows:

Command	Description
`Ctrl+a+ c`	Create a new window (with shell).
`Ctrl+a+ "`	List all windows.
`Ctrl+a+ 0`	Switch to window 0 (by number).
`Ctrl+a+ A`	Rename the current window.
`Ctrl+a+ S`	Split current region horizontally into two regions.
`Ctrl+a+ '`	Split current region vertically into two regions.
`Ctrl+a+ tab`	Switch the input focus to the next region.
`Ctrl+a+ Ctrl+a`	Toggle between the current and previous windows
`Ctrl+a+ Q`	Close all regions but the current one.
`Ctrl+a+ X`	Close the current region.

Restore a Linux Screen

To restore to a screen session you type `screen -r`, if you have more than one open screen session you have to add the session id to the command to connect to the right session.

Listing all open screen sessions

To find the session ID you can list the current running screen sessions
with:

`screen -ls`

There are screens on:

```
18787.pts-0.your-server   (Detached)
15454.pts-0.your-server   (Detached)
2 Sockets in /run/screens/S-yourserver.
```

If you want to restore screen 18787.pts-0, then type the following
command:

`screen -r 18787`

The nc command

The nc (or netcat) command is used to perform any operation involving TCP (Transmission Control Protocol, connection oriented), UDP (User Datagram Protocol, connection-less, no guarantee of data delivery) or UNIX-domain sockets. It can be thought of as swiss-army knife for communication protocol utilities.

Syntax:

```
nc [options] [ip] [port]
```

Examples:

1. Open a TCP connection to port 80 of host, using port 1337 as source port with timeout of 5s:

```
$ nc -p 1337 -w 5 host.ip 80
```

2. Open a UDP connection to port 80 on host:

```
$ nc -u host.ip 80
```

3. Create and listen on UNIX-domain stream socket:

```
$ nc -lU /var/tmp/dsocket
```

4. Create a basic server/client model:

This creates a connection, with no specific server/client sides with respect to nc, once the connection is established.

```
$ nc -l 1234 # in one console

$ nc 127.0.0.1 1234 # in another console
```

5. Build a basic data transfer model:

After the file has been transferred, sequentially, the connection closes automatically

```
$ nc -l 1234 > filename.out # to start listening in one console and collect data

$ nc host.ip 1234 < filename.in
```

6. Talk to servers:

Basic example of retrieving the homepage of the host, along with headers.

```
$ printf "GET / HTTP/1.0\r\n\r\n" | nc host.ip 80
```

7. Port scanning:

Checking which ports are open and running services on target machines. -z flag commands to inform about those rather than initiate

a connection.

```
$ nc -zv host.ip 20-2000 # range of ports to check for
```

Flags and their Functionalities:

Short Flag	Description
-4	Forces nc to use IPv4 addresses
-6	Forces nc to use IPv6 addresses
-b	Allow broadcast
-D	Enable debugging on the socket
-i	Specify time interval delay between lines sent and received
-k	Stay listening for another connection after current is over
-l	Listen for incoming connection instead of initiate one to remote
-T	Specify length of TCP
-p	Specify source port to be used
-r	Specify source and/or destination ports randomly
-s	Specify IP of interface which is used to send the packets
-U	Use UNIX-domain sockets
-u	Use UDP instead of TCP as protocol
-w	Declare a timeout threshold for idle or unestablished connections
-x	Should use specified protocol when talking to proxy server
-z	Specify to scan for listening daemons, without sending any data

The `make` command

The `make` command is used to automate the reuse of multiple commands in certain directory structure.

An example for that would be the use of `terraform init`, `terraform plan`, and `terraform validate` while having to change different subscriptions in Azure. This is usually done in the following steps:

```
az account set --subscription "Subscription - Name"
terraform init
```

How the `make` command can help us is it can automate all of that in just one go: `make tf-init`

Syntax:

```
make [ -f makefile ] [ options ] ... [ targets ] ...
```

Example use (guide):

1. Create `Makefile` in your guide directory

2. Include the following in your `Makefile` :

```
hello-world:
        echo "Hello, World!"

hello-bobby:
        echo "Hello, Bobby!"

touch-letter:
        echo "This is a text that is being inputted into our
letter!" > letter.txt

clean-letter:
        rm letter.txt
```

3. Execute `make hello-world` - this echoes "Hello, World" in our terminal.

4. Execute `make hello-bobby` - this echoes "Hello, Bobby!" in our terminal.

5. Execute `make touch-letter` - This creates a text file named `letter.txt` and populates a line in it.

6. Execute `make clean-letter`

References to lenghtier and more contentful tutorials:

(linoxide - linux make command examples)[https://linoxide.com/linux-make-command-examples/] (makefiletutorial.com - the name itself gives it out)[https://makefiletutorial.com/]

The `basename` command

The `basename` is a command-line utility that strips directory from given file names. Optionally, it can also remove any trailing suffix. It is a simple command that accepts only a few options.

Examples

The most basic example is to print the file name with the leading directories removed:

```
basename /etc/bar/foo.txt
```

The output will include the file name:

```
foo.txt
```

If you run basename on a path string that points to a directory, you will get the last segment of the path. In this example, /etc/bar is a directory.

```
basename /etc/bar
```

Output

```
bar
```

The basename command removes any trailing / characters:

```
basename /etc/bar/foo.txt/
```

Output

```
foo.txt
```

Options

1. By default, each output line ends in a newline character. To end the lines with NUL, use the -z (--zero) option.

```
$ basename -z /etc/bar/foo.txt
foo.txt$
```

2. The `basename` command can accept multiple names as arguments. To do so, invoke the command with the `-a` (`--multiple`) option, followed by the list of files separated by space. For example, to get the file names of `/etc/bar/foo.txt` and `/etc/spam/eggs.docx` you would run:

```
basename -a /etc/bar/foo.txt /etc/spam/eggs.docx
```

```
foo.txt
eggs.docx
```

Syntax

The basename command supports two syntax formats:

```
basename NAME [SUFFIX]
basename OPTION... NAME...
```

Additional functionalities

Removing a Trailing Suffix: To remove any trailing suffix from the file name, pass the suffix as a second argument:

```
basename /etc/hostname name
host
```

Generally, this feature is used to strip file extensions

Help Command

Run the following command to view the complete guide to basename command.

```
man basename
```

The `banner` command

The `banner` command writes ASCII character Strings to standard output in large letters. Each line in the output can be up to 10 uppercase or lowercase characters in length. On output, all characters appear in uppercase, with the lowercase input characters appearing smaller than the uppercase input characters.

Note: If you will define more than one NUMBER with sleep command then this command will delay for the sum of the values.

Examples :

1. To display a banner at the workstation, enter:

```
banner LINUX!
```

2. To display more than one word on a line, enclose the text in quotation marks, as follows:

```
banner "Intro to" Linux
```

This displays Intro to on one line and Linux on the next

3. Printing "101LinuxCommands" in large letters.

```
banner 101LinuxCommands
```

It will print only 101LinuxCo as banner has a default capacity of 10

The `alias` command

The `alias` command lets you create shortcuts for commands or define your own commands.
This is mostly used to avoid typing long commands.

Examples:

1. To show the list of all defined aliases in the reusable form `alias NAME=VALUE` :

```
alias -p
```

2. To make `ls -A` shortcut:

```
alias la='ls -A'
```

Syntax:

```
alias [-p] [name[=value]]
```

Setting Persistent Options:

As with most Linux custom settings for the terminal, any alias you defined is only applied to the current opening terminal session.

For any alias to be active for all new sessions you need to add that command to your rc file to be executed in the startup of every new terminal. this file can be as follows:

- **Bash**: ~/.bashrc
- **ZSH**: ~/.zshrc
- **Fish** – ~/.config/fish/config.fish

you can open that file with your favorite editor as follows:

```
vim ~/.bashrc
```

type your commands one per line, then save the file and exit. the commands will be automatically applied in the next session.

If you want to apply it in the current session, run the following command:

```
source ~/.bashrc
```

Opposite command:

To remove predefined alias you can use `unalias` command as follows:

```
unalias alias_name
```

to remove all aliases

```
unalias -a
```

The `which` command

`which` command identifies the executable binary that launches when you issue a command to the shell. If you have different versions of the same program on your computer, you can use which to find out which one the shell will use.

It has 3 return status as follows:

```
0 : If all specified commands are found and executable.
1 : If one or more specified commands is nonexistent or not
executable.
2 : If an invalid option is specified.
```

Examples

1. To find the full path of the ls command, type the following:

```
which ls
```

2. We can provide more than one arguments to the which command:

```
which netcat uptime ping
```

The which command searches from left to right, and if more than one matches are found in the directories listed in the PATH path variable, which will print only the first one.

3. To display all the paths for the specified command:

```
which [filename] -a
```

4. To display the path of node executable files, execute the command:

```
which node
```

5. To display the path of Java executable files, execute:

```
which java
```

Syntax

```
which [filename1] [filename2] ...
```

You can pass multiple programs and commands to which, and it will check them in order.

For example:

```
which ping cat uptime date head
```

Options

-a : List all instances of executables found (instead of just the first one of each).

-s : No output, just return 0 if all the executables are found, or 1 if some

were not found

413

The `date` command

The `date` command is used to print the system current date and time.

`date` command is also used to set the date and time of the system, but you need to be the super-user *(root)* to do it.

Examples:

1. To show the current date and time:

```
date
```

2. You can use -u option to show the date and time in UTC *(Coordinated Universal Time)* time zone

```
date -u
```

1. To display any given date string in formatted date:

```
date --date="2/02/2010"
date --date="2 years ago"
```

Syntax:

```
date [OPTION]... [+FORMAT]
date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-d	--date=STRING	convert the provided string into formatted date
-f	--file=DATEFILE	like --date but for files
-I[FMT]	--iso-8601[=FMT]	Display date and time in ISO 8601 format
-r	--reference=FILE	Display the last modification time of FILE
-s	--set=STRING	sets the time to the one described by STRING
-u	--universal	show the date and time in UTC *(Coordinated Universal Time)* time zone
-R	--rfc-email	Display date and time in ISO 8601 format Example: (Fri, 22 Oct 2021 05:18:42 +0200)
-	rfc-3339=FMT	Display date and time in RFC 3339 format
-	--debug	Usually used with --date to annotate the parsed date and warn about questionable usage to stderr

Control The output:

You can use Format specifiers to control the output date and time.

Examples:

Command	Output
`$ date "+%D"`	`10/22/21`
`$ date "+%D %T"`	`10/22/21 05:33:51`
`$ date "+%A %B %d %T %y"`	`Friday October 22 05:34:47 21`

Syntax:

```
date "+%[format-options ...]"
```

List of Format specifiers to control the output:

Specifiers	Description
%a	abbreviated weekday name *(e.g., Sun)*
%A	full weekday name *(e.g., Sunday)*
%b	abbreviated month name *(e.g., Jan)*
%B	full month name *(e.g., January)*
%c	date and time *(e.g., Thu Mar 3 23:05:25 2005)*
%C	century; like %Y, except omit last two digits (e.g., 20)
%d	day of month (e.g., 01)
%D	date; same as %m/%d/%y
%e	day of month, space padded; same as %_d
%F	full date; same as %Y-%m-%d
%g	last two digits of year of ISO week number (see %G)
%G	year of ISO week number (see %V); normally useful only with %V
%h	same as %b
%H	hour (00..23)
%I	hour (01..12)
%j	day of year (001..366)
%k	hour, space padded (0..23); same as %_H

Specifiers Description

%l	hour, space padded (1..12); same as %_I
%m	month (01..12)
%M	minute (00..59)
%n	a newline
%N	nanoseconds (000000000..999999999)
%p	locale's equivalent of either AM or PM; blank if not known
%P	like %p, but lower case
%q	quarter of year (1..4)
%r	locale's 12-hour clock time (e.g., 11:11:04 PM)
%R	24-hour hour and minute; same as %H:%M
%s	seconds since 1970-01-01 00:00:00 UTC
%S	second (00..60)
%t	a tab
%T	time; same as %H:%M:%S
%u	day of week (1..7); 1 is Monday
%U	week number of year, with Sunday as first day of week (00..53)
%V	ISO week number, with Monday as first day of week (01..53)
%w	day of week (0..6); 0 is Sunday
%W	week number of year, with Monday as first day of week (00..53)
%x	locale's date representation (e.g., 12/31/99)
%X	locale's time representation (e.g., 23:13:48)
%y	last two digits of year (00..99)
%Y	year
%z	+hhmm numeric time zone (e.g., -0400)
%:z	+hh:mm numeric time zone (e.g., -04:00)
%::z	+hh:mm:ss numeric time zone (e.g., -04:00:00)
%:::z	numeric time zone with : to necessary precision (e.g., -04, +05:30)
%Z	alphabetic time zone abbreviation (e.g., EDT)

The `mount` command

The `mount` command is used to mount 'attach' a filesystem and make it accessible by an existing directory structure tree.

Examples:

1. Displays version information:

```
mount -V
```

2. Attaching filesystem found on device and of type type at the directory dir:

```
mount -t type device dir
```

Syntax Forms:

```
mount [-lhV]
```

```
mount -a [-fFnrsvw] [-t vfstype] [-O optlist]
```

```
mount [-fnrsvw] [-t fstype] [-o options] device dir
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-h	--help	Dispaly a help message and exists
-n	--no-mtab	Mount without writing in /etc/mtab
-a	--all	Mount all filesystems (of the given types) mentioned in fstab
-r	--read-only	Mount the filesystem read-only
-w	--rw	Mount the filesystem as read/write.
-M	--move	Move a subtree to some other place.
-B	--bind	Remount a subtree somewhere else *(so that its contents are available in both places)*.

The `nice/renice` command

The `nice/renice` commands is used to modify the priority of the program to be executed. The priority range is between -20 and 19 where 19 is the lowest priority.

Examples:

1. Running cc command in the background with a lower priority than default (slower):

```
nice -n 15 cc -c *.c &
```

2. Increase the priority to all processes belonging to group "test":

```
renice --20 -g test
```

Syntax:

```
nice [  -Increment|  -n Increment ] Command [ Argument ... ]
```

Flags :

Short Flag	Long Flag	Description

Short Flag	Long Flag	Description
`-Increment`	-	Increment is the value of priority you want to assign.
`-n Increment`	-	Same as `-Increment`

The wc command

the `wc` command stands for word count. It's used to count the number of lines, words, and bytes *(characters)* in a file or standard input then prints the result to the standard output.

Examples:

1. To count the number of lines, words and characters in a file in order:

```
wc file.txt
```

2. To count the number of directories in a directory:

```
ls -F | grep / | wc -l
```

Syntax:

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

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-c	--bytes	print the byte counts

Short Flag	Long Flag	Description
-m	--chars	print the character counts
-l	--lines	print the newline counts
-	--files0-from=F	read input from the files specified by NUL-terminated names in file F. If F is - then read names from standard input
-L	--max-line-length	print the maximum display width
-w	--words	print the word counts

Additional Notes:

- Passing more than one file to `wc` command prints the counts for each file and the total conuts of them.
- you can combine more whan one flag to print the result as you want.

The tr command

The tr command in UNIX is a command line utility for translating or deleting characters. It supports a range of transformations including uppercase to lowercase, squeezing repeating characters, deleting specific characters and basic find and replace. It can be used with UNIX pipes to support more complex translation. tr stands for translate.

Examples:

1. Convert all lowercase letters in file1 to uppercase.

```
$ cat file1
foo
bar
baz
tr a-z A-Z < file1
FOO
BAR
BAZ
```

2. Make consecutive line breaks into one.

```
$ cat file1
foo


bar


baz
$ tr -s "\n" < file1
foo
bar
baz
```

3. Remove the newline code.

```
$ cat file1
foo
bar
baz
$ tr -d "\n" < file1
foobarbaz%
```

Syntax:

The general syntax for the tr command is as follows:

```
tr [options] string1 [string2]
```

Additional Flags and their Functionalities:

Short Flag	Long Flag	Description
-C		Complement the set of characters in string1, that is -C ab includes every character except for a and b.

Short Flag	Long Flag	Description
-c		Same as -C.
-d		Delete characters in string1 from the input.
-s		If there is a sequence of characters in string1, combine them into one.

The `fdisk` command

The `fdisk` command is used for controlling the disk partition table and making changes to it and this is a list of some of options provided by it :

- Organize space for new drives.
- Modify old drives.
- Create space for new partitions.
- Move data to new partitions.

Examples:

1. To view basic details about all available partitions on the system:

```
fdisk -l
```

2. To show the size of the partition:

```
fdisk -s /dev/sda
```

3. To view the help message and all options of the command:

```
fdisk -h
```

Syntax:

```
fdisk [options] device
```

Some of the command options:

On writing the following command

```
fdisk /dev/sdb
```

the following window appears :

```
linux@ubuntu:~$ sudo fdisk /dev/sda

Welcome to fdisk (util-linux 2.31.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help):
```

and then you type m which will show you all options you need such as
creating new partition and deleting a partition as in the following
picture :

```
Command (m for help): m

Help:

  DOS (MBR)
   a   toggle a bootable flag
   b   edit nested BSD disklabel
   c   toggle the dos compatibility flag

  Generic
   d   delete a partition
   F   list free unpartitioned space
   l   list known partition types
   n   add a new partition
   p   print the partition table
   t   change a partition type
   v   verify the partition table
   i   print information about a partition

  Misc
   m   print this menu
   u   change display/entry units
   x   extra functionality (experts only)

  Script
   I   load disk layout from sfdisk script file
   O   dump disk layout to sfdisk script file

  Save & Exit
   w   write table to disk and exit
   q   quit without saving changes

  Create a new label
   g   create a new empty GPT partition table
   G   create a new empty SGI (IRIX) partition table
   o   create a new empty DOS partition table
   s   create a new empty Sun partition table


Command (m for help):
```

The `Wait` commands

The wait command is a shell builtin that pauses script execution until a specific background process, or all running child processes, have finished.

Its primary purpose is to synchronize tasks, ensuring that a script doesn't continue to the next step until prerequisite background jobs are complete. A background process is a command that is run with an ampersand (&) at the end, which tells the shell to run it without waiting for it to finish.

Syntax

430

```
$ wait [PID]
```

[PID] - An optional Process ID to wait for. If no PID is given, wait will wait for all active child processes to complete.

Examples

431

1. Waiting for a Specific Process

This example shows how to launch a single background process and wait for it specifically.

Script:

```bash
#!/bin/bash
echo "This process will run in the background..." &
process_id=$!

echo "Script is now waiting for process ID: $process_id"
wait $process_id
echo "Process $process_id has finished."
echo "The script exited with status: $?"
```

Explanation:

- &: The ampersand runs the echo command in the background, allowing the script to immediately continue to the next line.

- $!: This is a special shell variable that holds the Process ID (PID) of the most recently executed background command. We save it to the process_id variable.

- wait $process_id: This is the key command. The script pauses here until the process with that specific ID is complete.

- **$?:** This variable holds the exit status of the last command that finished. An exit status of 0 means success.

Output:

```
$ bash wait_example.sh
Script is now waiting for process ID: 12345
This process will run in the background...
Process 12345 has finished.
The script exited with status: 0
```

2. Waiting for All Background Processes

This is the most common use case. Here, we launch several background tasks and then use a single wait command to pause until all of them are done.

Script:

```bash
#!/bin/bash

echo "Starting multiple background jobs..."
sleep 3 &
sleep 1 &
sleep 2 &

echo "Waiting for all sleep commands to finish."
wait
echo "All jobs are done. Continuing with the rest of the
script."
```

Output:

```
$ bash wait_all_example.sh
Starting multiple background jobs...
Waiting for all sleep commands to finish.
(after about 3 seconds)
All jobs are done. Continuing with the rest of the script.
```

The `zcat` command

The `zcat` allows you to look at a compressed file.

Examples:

1. To view the content of a compressed file:

```
~$ zcat test.txt.gz
Hello World
```

2. It can also Works with multiple files:

```
~$ zcat test2.txt.gz test.txt.gz
hello
Hello world
```

Syntax:

The general syntax for the `zcat` command is as follows:

```
zcat [  -n ] [  -V ] [  File ... ]
```

The `fold` command

The `fold` command in Linux wraps each line in an input file to fit a specified width and prints it to the standard output.

By default, it wraps lines at a maximum width of 80 columns but this is configurable.

To fold input using the fold command pass a file or standard input to the command.

Syntax:

```
fold [OPTION]... [FILE]...
```

Options

-w : By using this option in fold command, we can limit the width by number of columns.

By using this command we change the column width from default width of 80. Syntax:

```
fold -w[n] [FILE]
```

Example: wrap the lines of file1.txt to a width of 60 columns

```
fold -w60 file1.txt
```

-b : This option of fold command is used to limit the width of the output by the number of bytes rather than the number of columns.

By using this we can enforce the width of the output to the number of bytes.

```
fold -b[n] [FILE]
```

Example: limit the output width of the file to 40 bytes and the command breaks the output at 40 bytes.

```
fold -b40 file1.txt
```

-s : This option is used to break the lines on spaces so that words are not broken.

If a segment of the line contains a blank character within the first width column positions, break the line after the last such blank character meeting the width constraints.

```
fold -w[n] -s [FILE]
```

The `quota` command

The `quota` display disk usage and limits.

Installation:

You can simply go ahead and install quota on ubuntu systems by running:

```
sudo apt-get install quota
```

for Debian use the install command without sudo:

```
apt-get install quota
```

Syntax:

The general syntax for the `quota` command is as follows:

```
quota [ -u [ User ] ] [ -g [ Group ] ] [ -v | -q ]
```

The `aplay` command

`aplay` is a command-line audio player for ALSA(Advanced Linux Sound Architecture) sound card drivers. It supports several file formats and multiple soundcards with multiple devices. It is basically used to play audio on command-line interface. aplay is much the same as arecord only it plays instead of recording. For supported soundfile formats, the sampling rate, bit depth, and so forth can be automatically determined from the soundfile header.

Syntax:

439

```
$ aplay [flags] [filename [filename]] ...
```

Options:

440

```
-h, —help : Show the help information.
-d, —duration=# : Interrupt after # seconds.
-r, —rate=# : Sampling rate in Hertz. The default rate is 8000
Hertz.
—version : Print current version.
-l, —list-devices : List all soundcards and digital audio
devices.
-L, —list-pcms : List all PCMs(Pulse Code Modulation) defined.
-D, —device=NAME : Select PCM by name.
```

Note: This command contain various other options that we normally don't need. If you want to know more about you can simply run following command on your terminal.

```
aplay --help
```

Examples :

441

1. To play audio for only 10 secs at 2500hz frequency.

   ```
   $ aplay -d 10 -r 2500hz sample.mp3
   ```

   Plays sample.mp3 file for only 10 secs at 2500hz frequency.

2. To play full audio clip at 2500hz frezuency.

   ```
   $ aplay -r 2500hz sample.mp3
   ```

   Plays sample.mp3 file at 2500hz frezuency.

3. To Display version information.

   ```
   $ aplay --version
   ```

   Displays version information. For me it shows aplay: vesrion 1.1.0

The `spd-say` command

`spd-say` sends text-to-speech output request to speech-dispatcher process which handles it and ideally outputs the result to the audio system.

Syntax:

```
$ spd-say [options] "some text"
```

Options:

```
-r, --rate
       Set the rate of the speech (between -100 and +100,
default: 0)

-p, --pitch
       Set the pitch of the speech (between -100 and +100,
default: 0)

-i, --volume
       Set the volume (intensity) of the speech (between -100
and +100, default: 0)

-o, --output-module
       Set the output module

-l, --language
       Set the language (iso code)

-t, --voice-type
       Set the preferred voice type  (male1,  male2,  male3,
female1,  female2,  female3,
       child_male, child_female)

-m, --punctuation-mode
       Set the punctuation mode (none, some, all)

-s, --spelling
       Spell the message

-x, --ssml
       Set SSML mode on (default: off)

-e, --pipe-mode
       Pipe from stdin to stdout plus Speech Dispatcher

-P, --priority
       Set  priority  of  the  message  (important, message,
text, notification, progress;
       default: text)
```

```
-N, --application-name
        Set the application name used to establish the
connection to specified string value
        (default: spd-say)

-n, --connection-name
        Set  the connection name used to establish the
connection to specified string value
        (default: main)

-w, --wait
        Wait till the message is spoken or discarded

-S, --stop
        Stop speaking the message being spoken in Speech
Dispatcher

-C, --cancel
        Cancel all messages in Speech Dispatcher

-v, --version
        Print version and copyright info

-h, --help
        Print this info
```

Examples :

446

1. To Play the given text as the sound.

```
$ spd-say "Hello"
```

Plays "Hello" in sound.

The xeyes command

Xeyes is a graphical user interface program that creates a set of eyes on the desktop that follow the movement of the mouse cursor. It seems much of a funny command, than of any useful use. Being funny is as much useful, is another aspect.

Syntax:

```
xeyes
```

What is the purpose of xeyes?

xeyes is not for fun, at least not only. The purpose of this program is to let you follow the mouse pointer which is sometimes hard to see. It is very useful on multi-headed computers, where monitors are separated by some distance, and if someone (say teacher at school) wants to present something on the screen, the others on their monitors can easily follow the mouse with xeyes.

447

The `parted` command

The `parted` command is used to manage hard disk partitions on Linux. It can be used to add, delete, shrink and extend disk partitions along with the file systems located on them. You will need root access to the system to run `parted` commands.

NOTE: Parted writes the changes immediately to your disk, be careful when you are modifying the disk partitions.

Examples:

1. Displays partition layout of all block devices:

```
sudo parted -l
```

2. Display partition table of a specific `disk`

```
sudo parted disk print
```

Examples of `disk` are /dev/sda, /dev/sdb

3. Create a new disk label of `label-type` for a specific disk

```
sudo parted mklabel disk label-type
```

`label-type` can take values "aix", "amiga", "bsd", "dvh", "gpt", "loop",

"mac", "msdos", "pc98", or "sun"

449

4. Create a new partition in a specific `disk` of type `part-time`, file system is `fs-type` and of size `size` Mb.

```
sudo parted disk mkpart part-time fs-type 1 size
```

`part-time` can take values "primary", "logical", "extended".
`fs-type` is optional. It can take values "btrfs", "ext2", "ext3", "ext4", "fat16", "fat32", "hfs", "hfs+", "linux-swap", "ntfs", "reiserfs", "udf", or "xfs"
`size` has to less than the total size of the specified disk. To create a partition of size 50Mb, will take the value of 50

5. `parted` can also be run in an interactive format. Operations to manage the disk partitions can be performed by entering appropriate commands in the interactive session. `help` command in the interactive session shows a list of all possible disk management operations which can be performed.

```
$ sudo parted
GNU Parted 3.3
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of
commands.
(parted) print  # prints the partition table of the default
selected disk - /dev/sda
Model: ATA VBOX HARDDISK (scsi)
Disk /dev/sda: 53.7GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start    End      Size     Type     File system  Flags
 1      1049kB   53.7GB   53.7GB   primary  ext4         boot

(parted) select /dev/sdb  # change the current disk on which
operations have to be performed
Using /dev/sdb
(parted) quit  # exit the interactive session
```

Syntax Forms:

```
parted [options] [device [command [options...]...]]
```

Options:

Short Flag	Long Flag	Description
-h	--help	displays a help message listing all possible `commands [options]`
-l	--list	lists partition layout on all block devices
-m	--machine	displays machine parseable output
-v	--version	displays the version

450

Short Flag	Long Flag	Description
-a	--align	set alignment type for newly created partition. It can take the following values: `none`: Use the minimum alignment allowed by the disk type `cylinder`: Align partitions to cylinders `minimal`: Use minimum alignment as given by the disk topology information `optimal`: Use optimum alignment as given by the disk topology information

The nl command

The "nl" command enumerates lines in a file. A different way of viewing the contents of a file, the "nl" command can be very useful for many tasks.

Syntax

```
nl [ -b Type ] [ -f Type ] [ -h Type ] [ -l Number ] [ -d
Delimiter ] [ -i Number ] [ -n Format ] [ -v Number ] [ -w
Number ] [ -p ] [ -s Separator ] [ File ]
```

Examples:

454

1. To number all lines:

```
nl  -ba  chap1
```

2. Displays all the text lines:

```
[server@ssh ~]$ nl states
1 Alabama
2 Alaska
3 Arizona
4 Arkansas
5 California
6 Colorado
7 Connecticut.
8 Delaware
```

3. Specify a different line number format

```
nl  -i10  -nrz  -s::  -v10  -w4  chap1
```

You can name only one file on the command line. You can list the flags and the file name in any order.

The `pidof` command

The `pidof` is a command-line utility that allows you to find the process ID of a running program.

Syntax

```
pidof [OPTIONS] PROGRAM_NAME
```

To view the help message and all options of the command:

```
[user@home ~]$ pidof -h

 -c           Return PIDs with the same root directory
 -d <sep>     Use the provided character as output separator
 -h           Display this help text
 -n           Avoid using stat system function on network
shares
 -o <pid>     Omit results with a given PID
 -q           Quiet mode. Do not display output
 -s           Only return one PID
 -x           Return PIDs of shells running scripts with a
matching name
 -z           List zombie and I/O waiting processes. May cause
pidof to hang.
```

Examples:

To find the PID of the SSH server, you would run:

```
pidof sshd
```

If there are running processes with names matching `sshd`, their PIDs will be displayed on the screen. If no matches are found, the output will be empty.

```
# Output
4382 4368 811
```

`pidof` returns `0` when at least one running program matches with the requested name. Otherwise, the exit code is `1`. This can be useful when writing shell scripts.

To be sure that only the PIDs of the program you are searching for are displayed, use the full pathname to the program as an argument. For example, if you have two running programs with the same name located in two different directories pidof will show PIDs of both running programs.

By default, all PIDs of the matching running programs are displayed. Use the `-s` option to force pidof to display only one PID:

```
pidof -s program_name
```

The `-o` option allows you to exclude a process with a given PID from the command output:

```
pidof -o pid program_name
```

When pidof is invoked with the `-o` option, you can use a special PID named %PPID that represents the calling shell or shell script.

To return only the PIDs of the processes that are running with the same root directory, use the `-c` option. This option works only pidof is run as `root` or `sudo` user:

```
pidof -c pid program_name
```

Conclusion

459

The `pidof` command is used to find out the PIDs of a specific running program.

`pidof` is a simple command that doesn't have a lot of options. Typically you will invoke pidof only with the name of the program you are searching for.

The `shuf` command

The `shuf` command in Linux writes a random permutation of the input lines to standard output. It pseudo randomizes an input in the same way as the cards are shuffled. It is a part of GNU Coreutils and is not a part of POSIX. This command reads either from a file or standard input in bash and randomizes those input lines and displays the output.

Syntax

```
# file shuf
shuf [OPTION] [FILE]

# list shuf
shuf -e [OPTION]... [ARG]

# range shuf
shuf -i LO-HI [OPTION]
```

Like other Linux commands, shuf command comes with -—help option:

```
[user@home ~]$ shuf --help
Usage: shuf [OPTION]... [FILE]
  or:  shuf -e [OPTION]... [ARG]...
  or:  shuf -i LO-HI [OPTION]...
Write a random permutation of the input lines to standard
output.

With no FILE, or when FILE is -, read standard input.

Mandatory arguments to long options are mandatory for short
options too.
  -e, --echo                treat each ARG as an input line
  -i, --input-range=LO-HI   treat each number LO through HI as
an input line
  -n, --head-count=COUNT    output at most COUNT lines
  -o, --output=FILE         write result to FILE instead of
standard output
      --random-source=FILE  get random bytes from FILE
  -r, --repeat              output lines can be repeated
  -z, --zero-terminated     line delimiter is NUL, not newline
```

Examples:

shuf command without any option or argument.

```
shuf
```

When `shuf` command is used without any argument in the command line, it takes input from the user until `CTRL-D` is entered to terminate the set of inputs. It displays the input lines in a shuffled form. If `1`, `2`, `3`, `4` and `5` are entered as input lines, then it generates `1`, `2`, `3`, `4` and `5` in random order in the output as seen in the illustration below:

```
[user@home ~]$ shuf
1
2
3
4
5

4
5
1
2
3
```

Consider an example where Input is taken from the pipe:

```
{
seq 5 | shuf
}
```

`seq 5` returns the integers sequentially from `1` to `5` while the `shuf`

command takes it as input and shuffles the content i.e, the integers from 1 to 5. Hence, 1 to 5 is displayed as output in random order.

```
[user@home ~]$ {
> seq 5 | shuf
> }
5
4
2
3
1
```

File shuf

When `shuf` command is used without `-e` or `-i` option, then it operates as a file shuf i.e, it shuffles the contents of the file. The `<file_name>` is the last parameter of the `shuf` command and if it is not given, then input has to be provided from the shell or pipe.

Consider an example where input is taken from a file:

```
shuf file.txt
```

Suppose file.txt contains 6 lines, then the shuf command displays the input lines in random order as output.

```
[user@home ~]$ cat file.txt
line-1
line-2
line-3
line-4
line-5

[user@home ~]$ shuf file.txt
line-5
line-4
line-1
line-3
line-2
```

Any number of lines can be randomized by using -n option.

```
shuf -n 2 file.txt
```

This will display any two random lines from the file.

```
line-5
line-2
```

List shuf

When -e option is used with shuf command, it works as a list shuf. The arguments of the command are taken as the input line for the shuf.

Consider an example:

```
shuf -e A B C D E
```

It will take A, B, C, D, E as input lines, and will shuffle them to

display the output.

```
A
C
B
D
E
```

Any number of input lines can be displayed using the `-n` option along with `-e` option.

```
shuf -e -n 2 A B C D E
```

This will display any two of the inputs.

```
E
A
```

Range shuf

When `-i` option is used along with `shuf` command, it acts as a `range shuf`. It requires a range of input as input where `L0` is the lower bound while `HI` is the upper bound. It displays integers from `L0-HI` in shuffled form.

```
[user@home ~]$ shuf -i 1-5
4
1
3
2
5
```

Conclusion

466

The `shuf` command helps you randomize input lines. And there are features to limit the number of output lines, repeat lines and even generate random positive integers. Once you're done practicing whatever we've discussed here, head to the tool's <u>man page</u> to know more about it.

The `less` command

The less command is a Linux terminal pager which shows a file's content one screen at a time. Useful when dealing with a large text file because it doesn't load the entire file but accesses it page by page, resulting in fast loading speeds.

Syntax

```
less [options] file_path
```

Options

469

Some popular option flags include:

```
-E  less automatically exits upon reaching the end of file.
-f      Forces less to open non-regular files (a directory or
a device-special file).
-F      Exit less if the entire file can be displayed on the
first screen.
-g      Highlights the string last found using search. By
default, less highlights all strings matching the last search
command.
-G      Removes all highlights from strings found using
search.
```

For a complete list of options, refer to the less help file by running:

```
less --help
```

Few Examples:

1. Open a Text File

```
less /etc/updatedb.conf
```

2. Show Line Numbers

```
less -N /etc/init/mysql.conf
```

3. Open File with Pattern Search

```
less -pERROR /etc/init/mysql.conf
```

4. Remove Multiple Blank Lines

```
less welcome.txt
```

Here I showed you how to use the less command in Linux. Although there are other terminal pagers, such as most and more, but less could be a better choice as it is a powerful tool present in almost every system.

For more details:
https://phoenixnap.com/kb/less-command-in-linux#:~:text=The%20less%20command%20is%20a,resulting%20in%20fast%20loading%20speeds.

The `nslookup` command

The `nslookup` command is a network administration command-line tool for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or any other specific DNS record.

Syntax

472

```
nslookup [options] [host]
```

Options

Some popular option flags include:

```
-domain=[domain-name]    Change the default DNS name.
-debug  Show debugging information.
-port=[port-number]    Specify the port for queries. The
default port number is 53.
-timeout=[seconds]      Specify the time allowed for the
server to respond.
-type=a      View information about the DNS A address records.
-type=any    View all available records.
-type=hinfo  View hardware-related information about the host.
-type=mx     View Mail Exchange server information.
-type=ns     View Name Server records.
-type=ptr    View Pointer records. Used in reverse DNS
lookups.
-type=soa    View Start of Authority records.
```

Few Examples:

1. Query DNS Server

```
nslookup www.google.com
```

2. Specify a port to query

```
nslookup -port=53 www.google.com
```

3. Get the MX Record

```
nslookup -type=mx google.com
```

Here I showed you how to use the nslookup command in Linux. Although there are other DNS lookup tools, such as dig, nslookup could be a better choice as it is a powerful tool present in almost every system.

For more details: Nslookup on Wikipedia

The `cmp` command

The `cmp` command is a simple utility used to compare two files byte by byte.

If the files are identical, `cmp` produces no output and returns a successful exit status. If the files differ, it reports the byte and line number where the first difference occurred.

Syntax

476

```
$ cmp [OPTION]... FILE1 [FILE2]
```

Examples :

For the following examples, let's assume we have three files:

file1.txt:

```
hello world
```

file2.txt:

```
hello world
```

file3.txt:

```
hello World
```

1. Comparing two identical files

When the files are the same, cmp will produce no output. This is the standard way to confirm that two files are identical.

```
$ cmp file1.txt file2.txt
```

(No output is shown)

2. Comparing two different files

When a difference is found, cmp reports the location of the first

differing byte.

```
$ cmp file1.txt file3.txt
file1.txt file3.txt differ: byte 7, line 1
```

3. Displaying all differing bytes (--verbose or -l)

The -l (lowercase L) flag is very powerful. It prints the byte number (in decimal) and the value of the differing bytes (in octal) for every difference in the files.

```
$ cmp -l file1.txt file3.txt
 7 167 127
```

Explanation:

This output means that at byte position 7, file1.txt has the octal value 167 (the letter 'w'), while file3.txt has the octal value 127 (the letter 'W').

4. Comparing only the first "n" bytes (--bytes or -n)

You can limit the comparison to a certain number of bytes. Here, we only compare the first 5 bytes. Since "hello" is the same in both files, cmp finds no difference.

```
$ cmp -n 5 file1.txt file3.txt
```

(No output is shown)

5. Ignoring the initial "n" bytes (--ignore-initial or -i)

You can tell cmp to skip a certain number of bytes at the beginning of the files before starting its comparison. Here, we skip the first 6 bytes, so the comparison starts at the letter 'w'.

```
$ cmp -i 6 file1.txt file3.txt
file1.txt file3.txt differ: byte 1, line 1
```

Explanation:

The output now says the difference is at "byte 1" because the comparison started after the initial 6 bytes were ignored.

Common Options

Short Flag	Long Flag	Description
-b	--print-bytes	Print the differing bytes.
-i SKIP	--ignore-initial=SKIP	Skip the first SKIP bytes of both files.
-l	--verbose	Output the byte number and the values of all differing bytes.
-n LIMIT	--bytes=LIMIT	Compare at most LIMIT bytes.
-s	--quiet, --silent	Suppress all output. Only return an exit status.

The `expr` command

The `expr` command evaluates a given expression and displays its corresponding output. It is used for basic operations like addition, subtraction, multiplication, division, and modulus on integers and Evaluating regular expressions, string operations like substring, length of strings etc.

Syntax

481

```
expr expression
```

Few Examples:

1. **Perform basic arithmetic operations using expr command**

```
expr 7 + 14
expr 7 * 8
```

2. **Comparing two expressions**

```
x=10
y=20
res=`expr $x = $y`
echo $res
```

3. **Match the numbers of characters in two strings**

```
expr alphabet : alpha
```

4. **Find the modulus value**

```
expr 20 % 30
```

5. **Extract the substring**

```
a=HelloWorld
b=`expr substr $a 6 10`
echo $b
```

Additional Flags and their Functionalities

Flag	Description
--version	output version information and exit
--help	Display this help and exit

For more details: <u>Expr on Wikipedia</u>

The `wall` command

The `wall` command (short for *write all*) is used to send a message to all logged-in users on a Linux system. It is commonly used by system administrators to broadcast important information, such as planned maintenance or urgent announcements.

Syntax

```
$wall [options] [message]
```

If a [message] is not provided on the command line, wall will read from standard input until it receives an end-of-file character (Ctrl+D).

Options

486

Option	Description
-n	Suppress the banner (which shows who sent the message) and only show the message text.
-t [seconds]	Set a timeout, in seconds. `wall` will try to write to a user's terminal for this duration before giving up.

Examples:

1. Broadcast a message to all users

This command sends a message directly to all logged-in users.

```
$ wall "The system will shut down in 10 minutes. Please save
your work."
```

Output (on other users' terminals):

```
Broadcast message from your_username@hostname (pts/0) (Sat Oct
4 19:50:00 2025):

The system will shut down in 10 minutes. Please save your
work.
```

2. Broadcast a message from a text file

You can redirect the contents of a file to be used as the message.

Contents of message.txt:

```
System maintenance will begin shortly.
Connections may be temporarily unstable.
```

Command:

```
$ wall < message.txt
```

3. Send a multi-line message from standard input

If you run wall without a message, you can type a multi-line message directly in the terminal. Press Ctrl+D when you are finished to send it.

```
$ wall
The server is now back online.
All services are running normally.
<Ctrl+D>
```

Output (on other users' terminals):

```
Broadcast message from your_username@hostname (pts/0) (Sat Oct
4 19:52:00 2025):

The server is now back online.
All services are running normally.
```

The `ln` command

The `ln` command is used to create links between files in Linux. It can create both hard links and symbolic (soft) links, which are essential for file system management and organization.

Syntax

490

```
ln [options] target linkname
ln [options] target... directory
```

Types of Links

491

Hard Links

- Point directly to the file's inode
- Cannot span across different filesystems
- Cannot link to directories
- If original file is deleted, hard link still contains the data

Symbolic (Soft) Links

- Point to the file path (like shortcuts)
- Can span across different filesystems
- Can link to directories
- If original file is deleted, symbolic link becomes broken

Options

492

Some popular option flags include:

```
-s  Create symbolic (soft) links instead of hard links
-f      Force creation by removing existing destination files
-v      Verbose output, show what's being linked
-n      Treat destination as normal file if it's a symlink to
a directory
-r      Create relative symbolic links
-t      Specify target directory for links
```

Examples

1. Create a hard link

```
ln file.txt hardlink.txt
```

2. Create a symbolic link

```
ln -s /path/to/original/file.txt symlink.txt
```

3. Create a symbolic link to a directory

```
ln -s /var/log logs
```

4. Create multiple symbolic links in a directory

```
ln -s /usr/bin/python3 /usr/bin/gcc /usr/local/bin/
```

5. Create a relative symbolic link

```
ln -sr ../config/app.conf current_config
```

6. Force create a link (overwrite existing)

```
ln -sf /new/target existing_link
```

7. Create links with verbose output

```
ln -sv /source/file /destination/link
```

Use Cases

495

- Creating shortcuts to frequently used files or directories
- Maintaining multiple versions of configuration files
- Organizing files without duplicating storage space
- Creating backup references to important files
- Setting up development environments with shared libraries

Important Notes

- Use `ls -l` to see if a file is a symbolic link (indicated by `->`)
- Use `ls -i` to see inode numbers for hard links
- Be careful with symbolic links to avoid creating circular references
- Hard links share the same inode and disk space
- Symbolic links take minimal disk space (just the path information)

The `ln` command is essential for efficient file system organization and is widely used in system administration and development workflows.

For more details, check the manual: `man ln`

The `systemctl` command

The `systemctl` command is used to control and manage systemd services and the systemd system and service manager in Linux. It's the primary tool for managing services in modern Linux distributions.

Syntax

498

```
systemctl [options] command [service-name]
```

Common Commands

499

Service Management

```
start [service]      Start a service
stop [service]       Stop a service
restart [service]    Restart a service
reload [service]     Reload service configuration
status [service]     Show service status
enable [service]     Enable service to start at boot
disable [service]    Disable service from starting at boot
```

System Commands

```
reboot       Restart the system
poweroff     Shutdown the system
suspend      Suspend the system
hibernate    Hibernate the system
```

Options

Some popular option flags include:

```
-l          Show full output (don't truncate)
--no-pager  Don't pipe output into a pager
--failed    Show only failed units
--all       Show all units, including inactive ones
-q          Quiet mode, suppress output
-t          Specify unit type (service, socket, etc.)
```

Examples

501

1. Start a service

```
systemctl start nginx
```

2. Stop a service

```
systemctl stop apache2
```

3. Check service status

```
systemctl status ssh
```

4. Enable a service to start at boot

```
systemctl enable mysql
```

5. Disable a service from starting at boot

```
systemctl disable bluetooth
```

6. Restart a service

```
systemctl restart networking
```

7. Reload service configuration without stopping

```
systemctl reload nginx
```

8. List all active services

```
systemctl list-units --type=service
```

9. List all services (active and inactive)

```
systemctl list-units --type=service --all
```

10. List failed services

```
systemctl --failed
```

11. Show service dependencies

```
systemctl list-dependencies nginx
```

12. Check if a service is enabled

```
systemctl is-enabled ssh
```

13. Check if a service is active

```
systemctl is-active mysql
```

14. Restart the system

```
systemctl reboot
```

15. Shutdown the system

```
systemctl poweroff
```

Service Status Information

When checking status, you'll see:

- **Active (running)**: Service is currently running
- **Active (exited)**: Service completed successfully
- **Inactive (dead)**: Service is not running
- **Failed**: Service failed to start

Use Cases

505

- Managing web servers (nginx, apache)
- Controlling database services (mysql, postgresql)
- Managing system services (ssh, networking)
- Troubleshooting service issues
- Automating service management in scripts
- System administration and maintenance

Important Notes

- Requires root privileges for most operations (use `sudo`)
- Services are called "units" in systemd terminology
- Configuration files are located in `/etc/systemd/system/`
- Always check service status after making changes
- Use `journalctl` to view detailed service logs

The `systemctl` command is essential for modern Linux system administration and service management.

For more details, check the manual: `man systemctl`

The `journalctl` command

The `journalctl` command is used to view and query the systemd journal, which collects and stores system logs in a structured, indexed format. It's the primary tool for viewing system logs in modern Linux distributions.

Syntax

```
journalctl [options] [matches]
```

Options

509

Some popular option flags include:

```
-f           Follow journal (like tail -f)
-u [unit]    Show logs for specific unit/service
-p [level]   Filter by priority level (0-7)
-S [time]    Show entries since specified time
-U [time]    Show entries until specified time
-b           Show logs from current boot
-k           Show kernel messages only
-r           Reverse output (newest first)
-n [lines]   Show last N lines
--no-pager   Don't pipe output to pager
-x           Add explanatory help texts
-o [format]  Output format (json, short, verbose, etc.)
--disk-usage Show current disk usage
--vacuum-size=[size] Remove logs to reduce size
--vacuum-time=[time] Remove logs older than time
```

Priority Levels

510

```
0  Emergency (emerg)
1  Alert (alert)
2  Critical (crit)
3  Error (err)
4  Warning (warning)
5  Notice (notice)
6  Informational (info)
7  Debug (debug)
```

Examples

1. View all journal entries

```
journalctl
```

2. Follow live journal entries

```
journalctl -f
```

3. Show logs for a specific service

```
journalctl -u nginx
```

4. Show logs since last boot

```
journalctl -b
```

5. Show logs from previous boot

```
journalctl -b -1
```

6. Show kernel messages

```
journalctl -k
```

7. Show logs from specific time

```
journalctl --since "2024-01-01 00:00:00"
```

8. Show logs from last hour

```
journalctl --since "1 hour ago"
```

9. Show logs between time periods

```
journalctl --since "2024-01-01" --until "2024-01-02"
```

10. Show only error and critical messages

```
journalctl -p err
```

11. Show last 50 lines

```
journalctl -n 50
```

12. Follow logs for specific service

```
journalctl -u ssh -f
```

13. Show logs in JSON format

```
journalctl -o json
```

14. Show disk usage

```
journalctl --disk-usage
```

15. Remove old logs to free space

```
journalctl --vacuum-size=100M
```

16. Remove logs older than 2 weeks

```
journalctl --vacuum-time=2weeks
```

17. Show logs with explanations

```
journalctl -x
```

18. Show logs for specific process ID

```
journalctl _PID=1234
```

19. Show logs for specific user

```
journalctl _UID=1000
```

20. Show reverse chronological order

```
journalctl -r
```

Time Specifications

You can use various time formats:

- `"2024-01-01 12:00:00"`
- `"yesterday"`
- `"today"`
- `"1 hour ago"`
- `"30 minutes ago"`
- `"2 days ago"`

Output Formats

515

```
short      Default format
verbose    All available fields
json       JSON format
json-pretty Pretty-printed JSON
export     Binary export format
cat        Very short format
```

Use Cases

516

- Troubleshooting system issues
- Monitoring service behavior
- Security auditing
- Performance analysis
- Debugging system problems
- Tracking user activities

Important Notes

517

- Journal files are stored in `/var/log/journal/` or `/run/log/journal/`
- Requires appropriate permissions to view system logs
- Can consume significant disk space over time
- Use vacuum options to manage log size
- Persistent logging requires proper configuration

The `journalctl` command is essential for system administration and troubleshooting in systemd-based Linux distributions.

For more details, check the manual: `man journalctl`

The `watch` command

The `watch` command is used to execute a command repeatedly at regular intervals and display the output. It's particularly useful for monitoring changes in system status, file contents, or command output over time.

Syntax

519

```
watch [options] command
```

Options

520

Some popular option flags include:

```
-n [seconds]    Set update interval (default is 2 seconds)
-d              Highlight differences between updates
-t              Turn off header showing interval and command
-b              Beep if command has non-zero exit status
-e              Exit on error (non-zero exit status)
-g              Exit when output changes
-c              Interpret ANSI color sequences
-x              Pass command to shell with exec
-p              Precise timing mode
```

Examples

521

1. Watch system uptime every 2 seconds (default)

```
watch uptime
```

2. Watch disk space with custom interval

```
watch -n 5 df -h
```

3. Monitor memory usage with differences highlighted

```
watch -d free -h
```

4. Watch network connections

```
watch -n 1 'netstat -tuln'
```

5. Monitor specific directory contents

```
watch 'ls -la /var/log'
```

6. Watch CPU information

```
watch -n 2 'cat /proc/cpuinfo | grep "cpu MHz"'
```

7. Monitor active processes

```
watch -d 'ps aux | head -20'
```

8. Watch file size changes

```
watch -n 1 'ls -lh /var/log/syslog'
```

9. Monitor system load

```
watch -n 3 'cat /proc/loadavg'
```

10. Watch with precise timing

```
watch -p -n 0.5 date
```

11. Monitor service status

```
watch 'systemctl status nginx'
```

12. Watch with color support

```
watch -c 'ls --color=always'
```

13. Exit when output changes

```
watch -g 'cat /tmp/status.txt'
```

14. Watch with beep on error

```
watch -b 'ping -c 1 google.com'
```

15. Monitor log file size

```
watch 'wc -l /var/log/messages'
```

16. Watch docker containers

```
watch 'docker ps'
```

17. Monitor temperature sensors

```
watch -n 2 sensors
```

18. Watch git status

```
watch -d 'git status --porcelain'
```

19. Monitor bandwidth usage

```
watch -n 1 'cat /proc/net/dev'
```

20. Watch without header

```
watch -t 'date'
```

Use Cases

524

- System monitoring and performance analysis
- Watching log files for changes
- Monitoring network connectivity
- Tracking file system changes
- Observing process behavior
- Debugging system issues
- Automation and scripting
- Real-time status monitoring

Key Features

525

- **Real-time updates**: Continuously refreshes output
- **Difference highlighting**: Shows what changed between updates
- **Flexible intervals**: Customize update frequency
- **Exit conditions**: Can exit on changes or errors
- **Header information**: Shows command and update interval

Important Notes

- Press `Ctrl+C` to exit watch
- Use quotes around complex commands with pipes or redirections
- The command runs in a subshell each time
- Be careful with resource-intensive commands and short intervals
- Screen will clear and refresh with each update
- Header shows last update time and interval

Tips

- Use `-d` to easily spot changes
- Combine with `grep` to filter output
- Use longer intervals for less critical monitoring
- Consider system load when setting very short intervals

The `watch` command is an essential tool for system administrators and developers who need to monitor changes in real-time.

For more details, check the manual: `man watch`

The `jobs` command

The `jobs` command is used to display information about active jobs in the current shell session. Jobs are processes that have been started from the shell and can be managed using job control commands.

Syntax

529

```
jobs [options] [job_spec]
```

Options

Some popular option flags include:

```
-l      List process IDs along with job information
-p      List only process IDs
-n      List only jobs that have changed status since last
notification
-r      List only running jobs
-s      List only stopped jobs
-x      Replace job specifications with process IDs in command
```

Job States

531

Jobs can be in different states:

- **Running**: Job is currently executing
- **Stopped**: Job is suspended (paused)
- **Done**: Job has completed successfully
- **Terminated**: Job was killed or ended abnormally

Examples

532

1. List all current jobs

```
jobs
```

2. List jobs with process IDs

```
jobs -l
```

3. List only process IDs

```
jobs -p
```

4. List only running jobs

```
jobs -r
```

5. List only stopped jobs

```
jobs -s
```

6. Show status of specific job

```
jobs %1
```

Job Control Examples

1. Start a background job

```
sleep 100 &
```

2. Start multiple background jobs

```
find / -name "*.log" > /tmp/logs.txt 2>/dev/null &
ping google.com > /tmp/ping.txt &
```

3. View all jobs

```
jobs
```

Output might look like:

```
[1]-  Running    find / -name "*.log" > /tmp/logs.txt
2>/dev/null &
[2]+  Running    ping google.com > /tmp/ping.txt &
```

4. Stop a running job (Ctrl+Z)

```
# Start a command
vim myfile.txt
# Press Ctrl+Z to stop it
# Then check jobs
jobs
```

5. Bring job to foreground

```
fg %1
```

6. Send job to background

```
bg %1
```

7. Kill a specific job

```
kill %2
```

Job Specifications

535

You can refer to jobs using different formats:

- `%1` - Job number 1
- `%+` or `%%` - Current job (most recent)
- `%-` - Previous job
- `%string` - Job whose command line starts with string
- `%?string` - Job whose command line contains string

Examples with Job Control

1. Start and manage multiple jobs

```
# Start some background jobs
sleep 300 &
ping localhost > /dev/null &
find /usr -name "*.conf" > /tmp/configs.txt 2>/dev/null &

# List all jobs
jobs -l

# Bring first job to foreground
fg %1

# Put it back to background (after stopping with Ctrl+Z)
bg %1

# Kill second job
kill %2

# Check remaining jobs
jobs
```

2. Working with stopped jobs

```
# Start a text editor
nano myfile.txt

# Stop it with Ctrl+Z
# Check jobs
jobs

# Resume in background
bg

# Resume in foreground
fg
```

Use Cases

538

- **Multitasking**: Running multiple commands simultaneously
- **Long-running processes**: Managing tasks that take time to complete
- **Background processing**: Running tasks while working on other things
- **Job monitoring**: Keeping track of running processes
- **Process management**: Controlling and organizing shell processes

Related Commands

539

- `fg` - Bring job to foreground
- `bg` - Send job to background
- `nohup` - Run command immune to hangups
- `disown` - Remove job from job table
- `kill` - Terminate job or process

Important Notes

540

- Jobs are specific to the current shell session
- Job numbers are assigned sequentially
- Jobs disappear when they complete or when you exit the shell
- Use `&` at the end of a command to run it in background
- Press `Ctrl+Z` to stop (suspend) a running job
- Use `Ctrl+C` to terminate a running job

Advanced Examples

541

1. Run command in background and disown it

```
long_running_script.sh &
disown %1
```

2. Check for completed jobs

```
jobs -n
```

3. Kill all jobs

```
kill $(jobs -p)
```

The `jobs` command is essential for managing multiple processes and implementing effective workflow management in the shell.

For more details, check the manual: `man jobs` or `help jobs`

The bg command

The bg command is used to put stopped jobs in the background, allowing them to continue running while you use the terminal for other tasks. It's part of the job control features in Unix-like shells.

Syntax

543

```
bg [job_spec]
```

If no job specification is provided, bg operates on the current job (most recent job).

Job Specifications

You can refer to jobs using different formats:

- `%1` - Job number 1
- `%+` or `%%` - Current job (most recent)
- `%-` - Previous job
- `%string` - Job whose command line starts with string
- `%?string` - Job whose command line contains string

Examples

545

1. Put the current stopped job in background

```
bg
```

2. Put specific job in background

```
bg %1
```

3. Put multiple jobs in background

```
bg %1 %2 %3
```

Complete Job Control Workflow

Here's a typical workflow demonstrating bg usage:

1. Start a long-running command

```
find / -name "*.log" > /tmp/findlogs.txt 2>/dev/null
```

2. Stop the job with Ctrl+Z

```
^Z
[1]+  Stopped     find / -name "*.log" > /tmp/findlogs.txt
2>/dev/null
```

3. Check jobs

```
jobs
```

Output:

```
[1]+  Stopped     find / -name "*.log" > /tmp/findlogs.txt
2>/dev/null
```

4. Put the stopped job in background

```
bg %1
```

Output:

```
[1]+ find / -name "*.log" > /tmp/findlogs.txt 2>/dev/null &
```

5. Verify the job is running in background

```
jobs
```

Output:

```
[1]+  Running     find / -name "*.log" > /tmp/findlogs.txt 2>/dev/null &
```

Practical Examples

1. Working with a text editor

```
# Start editing a file
vim myfile.txt

# Stop with Ctrl+Z
# Put it in background
bg

# Now you can run other commands while vim runs in background
ls -la

# Bring vim back to foreground when needed
fg %1
```

2. Managing multiple background tasks

```
# Start several tasks and stop them
ping google.com > /tmp/ping1.txt
# Ctrl+Z
sleep 300
# Ctrl+Z
tar czf backup.tar.gz /home/user/documents
# Ctrl+Z

# Check all stopped jobs
jobs

# Put all in background
bg %1
bg %2
bg %3

# Or put specific ones
bg %ping    # Job starting with "ping"
```

3. Starting command directly in background vs using bg

549

```
# Method 1: Start directly in background
find /usr -name "*.conf" > /tmp/configs.txt &

# Method 2: Start normally, stop, then background
find /usr -name "*.conf" > /tmp/configs.txt
# Ctrl+Z
bg
```

Related Commands

550

- `fg` - Bring job to foreground
- `jobs` - List active jobs
- `kill` - Terminate job
- `nohup` - Run command immune to hangups
- `disown` - Remove job from job table

Use Cases

551

- **Multitasking**: Run multiple tasks simultaneously
- **Long processes**: Let time-consuming tasks run while working on other things
- **Interactive programs**: Temporarily background editors or interactive tools
- **Development**: Background compilation while coding
- **System administration**: Background monitoring while performing other tasks

Important Notes

- Jobs put in background with `bg` are still attached to the terminal
- If you close the terminal, background jobs may be terminated
- Use `nohup` or `disown` for persistent background processes
- Background jobs cannot read from stdin (keyboard input)
- You can use `fg` to bring background jobs back to foreground
- Background jobs continue to write to stdout/stderr unless redirected

Error Handling

553

If bg fails, common reasons include:

- Job doesn't exist
- Job is already running
- Job cannot be put in background (some interactive programs)

Tips

- Always check job status with `jobs` before and after using `bg`
- Redirect output for background jobs to avoid cluttering the terminal
- Use job control responsibly to avoid system resource issues
- Consider using terminal multiplexers like `screen` or `tmux` for persistent sessions

The `bg` command is essential for effective multitasking and job management in the shell environment.

For more details, check the manual: `help bg`

The `fg` command

The `fg` command is used to bring background or stopped jobs to the foreground, making them the active process in your terminal. It's an essential part of job control in Unix-like shells.

Syntax

556

```
fg [job_spec]
```

If no job specification is provided, `fg` operates on the current job (most recent job).

Job Specifications

You can refer to jobs using different formats:

- `%1` - Job number 1
- `%+` or `%%` - Current job (most recent)
- `%-` - Previous job
- `%string` - Job whose command line starts with string
- `%?string` - Job whose command line contains string

Examples

558

1. Bring the current job to foreground

```
fg
```

2. Bring specific job to foreground

```
fg %1
```

3. Bring job by partial command name

```
fg %vim
```

4. Bring job containing specific text

```
fg %?backup
```

Complete Job Control Workflow

Here's a typical workflow demonstrating `fg` usage:

1. Start a background job

```
ping google.com > /tmp/ping.txt &
```

2. Start another job and stop it

```
vim myfile.txt
# Press Ctrl+Z to stop
```

3. Check current jobs

```
jobs
```

Output:

```
[1]-  Running    ping google.com > /tmp/ping.txt &
[2]+  Stopped    vim myfile.txt
```

4. Bring vim to foreground

```
fg %2
```

5. Work in vim, then stop again (Ctrl+Z) and bring ping to foreground

```
fg %1
```

Practical Examples

1. Working with editors

```
# Start editing
nano config.txt
# Stop with Ctrl+Z
# Do other work
ls -la
# Return to editor
fg
```

2. Managing multiple development tasks

```
# Start compilation in background
make all > build.log 2>&1 &

# Start editing source code
vim main.c
# Stop editor (Ctrl+Z)

# Check build progress
fg %make
# Stop build monitoring (Ctrl+Z)

# Return to editing
fg %vim
```

3. Interactive debugging session

```
# Start debugger
gdb ./myprogram
# Stop debugger (Ctrl+Z)

# Check core dumps or logs
ls -la core.*

# Return to debugger
fg %gdb
```

4. Working with multiple terminals/sessions

```
# Start SSH session
ssh user@remote-server
# Stop SSH (Ctrl+Z)

# Do local work
ps aux | grep myprocess

# Return to SSH session
fg %ssh
```

Advanced Usage

1. Switching between multiple stopped jobs

```
# Start several editors
vim file1.txt
# Ctrl+Z
vim file2.txt
# Ctrl+Z
nano file3.txt
# Ctrl+Z

# Check all jobs
jobs

# Switch between them
fg %1     # vim file1.txt
# Ctrl+Z
fg %2     # vim file2.txt
# Ctrl+Z
fg %3     # nano file3.txt
```

2. Using with job control in scripts

```
#!/bin/bash
# Start background monitoring
tail -f /var/log/syslog &
MONITOR_PID=$!

# Do main work
./main_script.sh

# Bring monitor to foreground for review
fg %tail

# Or kill it
kill $MONITOR_PID
```

Related Commands

- bg - Put job in background
- jobs - List active jobs
- kill - Terminate job
- Ctrl+Z - Stop (suspend) current job
- Ctrl+C - Terminate current job

Use Cases

565

- **Code editing**: Switch between multiple open editors
- **Development**: Alternate between compilation and editing
- **System monitoring**: Switch between monitoring tools
- **Remote sessions**: Resume SSH or other remote connections
- **Interactive programs**: Return to paused interactive applications
- **Debugging**: Resume debugger sessions

Important Notes

- When a job is brought to foreground, it becomes the active process
- You can only have one foreground job at a time
- Foreground jobs can receive keyboard input
- Use Ctrl+Z to stop (suspend) a foreground job
- Use Ctrl+C to terminate a foreground job
- Background jobs continue running even when not in foreground

Error Handling

567

Common issues with `fg`:

- Job doesn't exist: `fg: %3: no such job`
- No jobs available: `fg: no current job`
- Job already in foreground

Tips for Effective Usage

1. **Use job numbers**: More reliable than partial names
2. **Check jobs first**: Always run `jobs` to see current status
3. **Consistent workflow**: Develop a routine for job switching
4. **Redirect output**: Background jobs should redirect output to avoid interference

```
# Good practice
tail -f /var/log/messages > monitor.out 2>&1 &
vim script.sh
# Ctrl+Z
fg %tail   # Review logs
# Ctrl+Z
fg %vim    # Continue editing
```

Integration with Other Tools

`fg` works well with:

- **Terminal multiplexers**: `screen`, `tmux`
- **Development environments**: IDEs, editors
- **System monitoring**: `top`, `htop`, `tail`
- **Network tools**: `ssh`, `ping`, `netstat`

The `fg` command is crucial for efficient terminal multitasking and provides seamless switching between different tasks.

For more details, check the manual: `help fg`

The `time` command

The `time` command is used to measure the execution time of programs and commands. It provides detailed information about how long a command takes to run, including user time, system time, and real (wall-clock) time.

Syntax

571

```
time [options] command [arguments]
```

Syntax

Types of Time Measurement

Real Time (Wall-clock time)

- Total elapsed time from start to finish
- Includes time spent waiting for I/O, other processes, etc.

User Time

- Time spent executing user-level code
- CPU time used by the process itself

System Time

- Time spent in kernel mode
- CPU time used for system calls

Output Format

The standard output shows three measurements:

```
real    0m2.345s
user    0m1.234s
sys     0m0.567s
```

Options

Some popular option flags include:

```
-p          Use POSIX format output
-f format   Use custom format string
-o file     Write output to file instead of stderr
-a          Append to output file instead of overwriting
-v          Verbose output with detailed statistics
```

Examples

1. Time a simple command

```
time ls -la
```

2. Time a script execution

```
time ./my_script.sh
```

3. Time a compilation process

```
time make all
```

4. Time with POSIX format

```
time -p find /usr -name "*.txt"
```

5. Save timing information to file

```
time -o timing.log -a make clean && make
```

6. Verbose timing information

```
time -v python large_calculation.py
```

Advanced Usage

576

1. Time multiple commands

```
time (command1 && command2 && command3)
```

2. Time with custom format

```
/usr/bin/time -f "Time: %E, Memory: %M KB"
./memory_intensive_program
```

3. Time and redirect output

```
time (find /usr -name "*.log" > found_logs.txt 2>&1)
```

4. Compare execution times

```
echo "Method 1:"
time method1_script.sh

echo "Method 2:"
time method2_script.sh
```

Using GNU time (Advanced)

The GNU version of `time` (usually at `/usr/bin/time`) provides more detailed information:

```
/usr/bin/time -v command
```

This shows additional statistics like:

- Maximum resident set size (memory usage)
- Page faults
- Context switches
- File system inputs/outputs

Format Specifiers for GNU time

```
%E    Elapsed real time (wall clock time)
%U    User CPU time
%S    System CPU time
%M    Maximum resident set size (KB)
%P    Percentage of CPU used
%X    Average size of shared text (KB)
%D    Average size of unshared data (KB)
%c    Number of voluntary context switches
%w    Number of involuntary context switches
%I    Number of file system inputs
%O    Number of file system outputs
```

Practical Examples

1. Profile a Python script

```
time python -c "
import time
for i in range(1000000):
    str(i)
"
```

2. Compare different algorithms

```
echo "Bubble sort:"
time ./bubble_sort < large_dataset.txt

echo "Quick sort:"
time ./quick_sort < large_dataset.txt
```

3. Time database operations

```
time mysql -u user -p database < complex_query.sql
```

4. Time network operations

```
time wget https://large-file.example.com/bigfile.zip
```

5. Time compression operations

```
echo "gzip compression:"
time gzip -c large_file.txt > large_file.gz

echo "bzip2 compression:"
time bzip2 -c large_file.txt > large_file.bz2
```

6. Profile build processes

```
echo "Clean build timing:"
time (make clean && make -j4)
```

Understanding the Output

Example output interpretation:

```
real    0m5.234s    # Total elapsed time (5.234 seconds)
user    0m3.456s    # CPU time in user mode (3.456 seconds)
sys     0m0.789s    # CPU time in system mode (0.789 seconds)
```

Analysis:

- If real > user + sys: Process was I/O bound or waiting
- If real ≈ user + sys: Process was CPU bound
- If user >> sys: Process spent most time in user code
- If sys >> user: Process made many system calls

Benchmarking Best Practices

1. **Multiple runs**: Run several times and average results

```
for i in {1..5}; do
    echo "Run $i:"
    time ./program
done
```

2. **Warm-up runs**: Do a few runs to warm up caches

```
# Warm-up
./program > /dev/null 2>&1

# Actual timing
time ./program
```

3. **Consistent environment**: Control variables

```
# Clear caches
sync && echo 3 > /proc/sys/vm/drop_caches

# Run with consistent priority
nice -n 0 time ./program
```

Use Cases

583

- **Performance optimization**: Identify slow operations
- **Benchmarking**: Compare different implementations
- **System analysis**: Understand resource usage patterns
- **Build optimization**: Time compilation processes
- **Script profiling**: Find bottlenecks in shell scripts
- **Development**: Measure algorithm efficiency

Important Notes

584

- Built-in `time` vs. `/usr/bin/time` may have different features
- Results can vary between runs due to system load
- I/O operations can significantly affect timing
- Use multiple measurements for accurate benchmarking
- Consider system caches when timing file operations

Combining with Other Tools

1. With `nice` for priority control

```
time nice -n 10 ./cpu_intensive_task
```

2. With `timeout` for maximum runtime

```
time timeout 30s ./potentially_slow_command
```

3. With `strace` for system call analysis

```
time strace -c ./program 2> syscalls.log
```

The `time` command is essential for performance analysis, optimization, and understanding program behavior in Linux systems.

For more details, check the manual: `man time`

The `export` command

The `export` command is used to set environment variables that will be available to child processes. It makes variables available to all processes started from the current shell session.

Syntax

587

```
export [options] [variable[=value]]
export [options] [name[=value] ...]
```

How Environment Variables Work

- **Local variables**: Only available in the current shell
- **Environment variables**: Available to current shell and all child processes
- `export` converts local variables to environment variables

Options

589

Some popular option flags include:

```
-f    Export functions instead of variables
-n    Remove variable from environment (unexport)
-p    Display all exported variables
```

Examples

1. Export a simple variable

```
export MY_VAR="Hello World"
```

2. Export multiple variables at once

```
export VAR1="value1" VAR2="value2" VAR3="value3"
```

3. Export an existing local variable

```
LOCAL_VAR="test"
export LOCAL_VAR
```

4. Show all exported variables

```
export -p
```

5. Export PATH modifications

```
export PATH="$PATH:/usr/local/bin"
```

6. Export with command substitution

```
export CURRENT_DATE=$(date)
export HOSTNAME=$(hostname)
```

7. Unexport a variable (remove from environment)

```
export -n MY_VAR
```

8. Export function

```
my_function() {
    echo "Hello from function"
}
export -f my_function
```

Common Environment Variables

1. **PATH** - Executable search paths

```
export PATH="/usr/local/bin:$PATH"
```

2. **HOME** - User's home directory

```
export HOME="/home/username"
```

3. **EDITOR** - Default text editor

```
export EDITOR="vim"
export VISUAL="code"
```

4. **LANG** - System language and locale

```
export LANG="en_US.UTF-8"
```

5. **PS1** - Primary prompt string

```
export PS1="\u@\h:\w\$ "
```

6. **JAVA_HOME** - Java installation directory

```
export JAVA_HOME="/usr/lib/jvm/java-11-openjdk"
```

7. **NODE_ENV** - Node.js environment

```
export NODE_ENV="production"
```

Development Environment Examples

1. **Python development**

```
export PYTHONPATH="$PYTHONPATH:/path/to/modules"
export VIRTUAL_ENV="/path/to/venv"
```

2. **Node.js development**

```
export NODE_PATH="/usr/local/lib/node_modules"
export NPM_CONFIG_PREFIX="$HOME/.npm-global"
```

3. **Go development**

```
export GOPATH="$HOME/go"
export GOROOT="/usr/local/go"
export PATH="$PATH:$GOROOT/bin:$GOPATH/bin"
```

4. **Database configuration**

```
export DB_HOST="localhost"
export DB_PORT="5432"
export DB_NAME="myapp"
export DB_USER="dbuser"
```

Shell Configuration Files

Make exports permanent by adding them to configuration files:

1. **Bash** - `~/.bashrc` or `~/.bash_profile`

```
echo 'export MY_VAR="permanent_value"' >> ~/.bashrc
```

2. **Zsh** - `~/.zshrc`

```
echo 'export MY_VAR="permanent_value"' >> ~/.zshrc
```

3. **System-wide** - `/etc/environment` or `/etc/profile`

```
# /etc/environment
MY_GLOBAL_VAR="system_wide_value"
```

Checking Variables

1. Check if variable is exported

```
env | grep MY_VAR
printenv MY_VAR
echo $MY_VAR
```

2. Check variable scope

```
# Local variable
MY_LOCAL="test"
bash -c 'echo $MY_LOCAL'  # Empty output

# Exported variable
export MY_EXPORTED="test"
bash -c 'echo $MY_EXPORTED'  # Shows "test"
```

Advanced Usage

1. Conditional exports

```
if [ -d "/opt/myapp" ]; then
    export MYAPP_HOME="/opt/myapp"
fi
```

2. Export with default values

```
export EDITOR="${EDITOR:-vim}"
export PORT="${PORT:-3000}"
```

3. Export arrays (Bash 4+)

```
declare -a my_array=("item1" "item2" "item3")
export my_array
```

4. Export with validation

```
validate_and_export() {
    if [ -n "$1" ] && [ -n "$2" ]; then
        export "$1"="$2"
        echo "Exported $1=$2"
    else
        echo "Error: Invalid arguments"
    fi
}

validate_and_export "API_KEY" "your-secret-key"
```

Use Cases

598

- **Development environments**: Setting up language-specific paths
- **Application configuration**: Database URLs, API keys, feature flags
- **System administration**: Custom PATH modifications, proxy settings
- **CI/CD pipelines**: Build configuration, deployment targets
- **Security**: Sensitive data that shouldn't be in scripts

Important Notes

599

- Exported variables are inherited by child processes
- Changes to exported variables in child processes don't affect parent
- Use quotes for values with spaces or special characters
- Environment variables are typically uppercase by convention
- Be careful with sensitive data in environment variables
- Some variables (like PATH) should be appended to, not replaced

Security Considerations

1. Avoid sensitive data in exports

```
# Bad
export PASSWORD="secret123"

# Better - read from secure file or prompt
read -s -p "Enter password: " PASSWORD
export PASSWORD
```

2. Use temporary exports for sensitive operations

```
# Export temporarily
export TEMP_TOKEN="secret"
my_command_that_needs_token
unset TEMP_TOKEN  # Clean up
```

The `export` command is fundamental for shell scripting and system administration, enabling proper environment configuration for applications and processes.

For more details, check the manual: `help export` or `man bash`

The `ufw` command

UFW (Uncomplicated Firewall) is a user-friendly command-line frontend for managing iptables firewall rules on Ubuntu and other Debian-based systems. It provides a simple way to configure firewall rules without dealing with complex iptables syntax.

Syntax

602

```
ufw [options] command [parameters]
```

Installation

603

```
# Ubuntu/Debian
sudo apt update && sudo apt install ufw

# Check if UFW is installed
which ufw
```

Basic Commands

604

Enable/Disable UFW

```
# Enable UFW
sudo ufw enable

# Disable UFW
sudo ufw disable

# Check UFW status
sudo ufw status
sudo ufw status verbose
sudo ufw status numbered
```

Basic Rules

Allow/Deny Traffic

1. **Allow specific ports**

```
# Allow SSH (port 22)
sudo ufw allow 22
sudo ufw allow ssh

# Allow HTTP (port 80)
sudo ufw allow 80
sudo ufw allow http

# Allow HTTPS (port 443)
sudo ufw allow 443
sudo ufw allow https

# Allow custom port
sudo ufw allow 8080
```

2. **Deny specific ports**

```
# Deny port 80
sudo ufw deny 80

# Deny SSH from specific IP
sudo ufw deny from 192.168.1.100 to any port 22
```

3. **Allow/Deny by service name**

```
# Allow common services
sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow https
sudo ufw allow ftp
sudo ufw allow smtp
```

Advanced Rules

Port Ranges

```
# Allow port range
sudo ufw allow 1000:2000/tcp
sudo ufw allow 1000:2000/udp

# Allow specific protocol
sudo ufw allow 53/udp  # DNS
sudo ufw allow 53/tcp  # DNS over TCP
```

IP Address Rules

1. **Allow/Deny specific IP addresses**

```
# Allow from specific IP
sudo ufw allow from 192.168.1.100

# Deny from specific IP
sudo ufw deny from 192.168.1.50

# Allow subnet
sudo ufw allow from 192.168.1.0/24
```

2. **Allow IP to specific port**

```
# Allow specific IP to SSH
sudo ufw allow from 192.168.1.100 to any port 22

# Allow subnet to web server
sudo ufw allow from 10.0.0.0/8 to any port 80
```

Interface-specific Rules

```
# Allow on specific interface
sudo ufw allow in on eth0 to any port 80

# Allow out on specific interface
sudo ufw allow out on eth1 to any port 443
```

Rule Management

List Rules

```
# Show status and rules
sudo ufw status

# Show numbered rules
sudo ufw status numbered

# Show verbose status
sudo ufw status verbose
```

Delete Rules

```
# Delete by rule number
sudo ufw delete 3

# Delete by specifying the rule
sudo ufw delete allow 80
sudo ufw delete allow from 192.168.1.100
```

Insert Rules

```
# Insert rule at specific position
sudo ufw insert 1 allow from 192.168.1.0/24
```

Default Policies

```
# Set default policies
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw default deny forward

# Check current defaults
sudo ufw status verbose
```

Application Profiles

List Available Profiles

```
# List application profiles
sudo ufw app list

# Show profile info
sudo ufw app info OpenSSH
sudo ufw app info "Apache Full"
```

Use Application Profiles

```
# Allow application
sudo ufw allow OpenSSH
sudo ufw allow "Apache Full"
sudo ufw allow "Nginx Full"

# Common application profiles
sudo ufw allow "OpenSSH"
sudo ufw allow "Apache"
sudo ufw allow "Apache Secure"
sudo ufw allow "Nginx HTTP"
sudo ufw allow "Nginx HTTPS"
sudo ufw allow "Nginx Full"
```

Logging

612

```
# Enable logging
sudo ufw logging on

# Set log level
sudo ufw logging low
sudo ufw logging medium
sudo ufw logging high

# Disable logging
sudo ufw logging off

# View logs
sudo tail -f /var/log/ufw.log
```

Reset and Reload

613

```
# Reset all rules to default
sudo ufw --force reset

# Reload UFW
sudo ufw reload
```

Common Use Cases

1. Basic Web Server Setup

```
# Allow SSH, HTTP, and HTTPS
sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow https
sudo ufw enable
```

2. Database Server (MySQL)

```
# Allow MySQL only from application servers
sudo ufw allow from 192.168.1.10 to any port 3306
sudo ufw allow from 192.168.1.11 to any port 3306
```

3. Development Server

```
# Allow common development ports
sudo ufw allow 3000  # Node.js
sudo ufw allow 8000  # Django
sudo ufw allow 5000  # Flask
sudo ufw allow 4200  # Angular
```

4. Mail Server

```
# Allow mail server ports
sudo ufw allow smtp        # Port 25
sudo ufw allow 587/tcp     # SMTP submission
sudo ufw allow 993/tcp     # IMAPS
sudo ufw allow 995/tcp     # POP3S
```

5. DNS Server

```
# Allow DNS traffic
sudo ufw allow 53/tcp
sudo ufw allow 53/udp
```

Security Best Practices

1. Principle of Least Privilege

```
# Start with deny all
sudo ufw default deny incoming
sudo ufw default allow outgoing

# Only allow what's needed
sudo ufw allow ssh
sudo ufw allow from 192.168.1.0/24 to any port 80
```

2. Limit SSH Access

```
# Limit SSH attempts (6 attempts in 30 seconds)
sudo ufw limit ssh

# Allow SSH only from specific networks
sudo ufw allow from 192.168.1.0/24 to any port 22
sudo ufw deny ssh
```

3. Monitor and Log

```
# Enable logging
sudo ufw logging medium

# Monitor logs
sudo tail -f /var/log/ufw.log | grep DPT
```

Troubleshooting

1. Check Current Rules

```
sudo ufw status numbered
sudo iptables -L -n
```

2. Test Connections

```
# Test if port is accessible
telnet your-server-ip 80
nc -zv your-server-ip 22
```

3. Debug UFW

```
# Dry run (show what would happen)
sudo ufw --dry-run allow 80

# Check UFW version
ufw --version
```

Advanced Configuration

1. Custom Rules File

```
# Edit UFW rules directly
sudo vim /etc/ufw/user.rules
sudo vim /etc/ufw/user6.rules
```

2. Rate Limiting

```
# Limit connections per IP
sudo ufw limit ssh
sudo ufw limit 80/tcp
```

3. Port Forwarding

```
# Enable IP forwarding
echo 'net.ipv4.ip_forward=1' | sudo tee -a
/etc/ufw/sysctl.conf

# Add NAT rules to /etc/ufw/before.rules
```

Integration with Services

1. Docker Integration

```
# Allow Docker containers
sudo ufw allow from 172.17.0.0/16

# Block Docker bypass (in /etc/ufw/after.rules)
```

2. Fail2ban Integration

```
# UFW works with fail2ban
sudo apt install fail2ban
# Configure fail2ban to use UFW actions
```

Important Notes

620

- UFW is a frontend for iptables, not a replacement
- Rules are processed in order (first match wins)
- Default policies apply when no specific rule matches
- UFW doesn't interfere with existing iptables rules by default
- Always test rules before enabling in production
- Keep SSH access rule before enabling UFW remotely

Quick Reference

621

```
# Essential commands
sudo ufw enable                    # Enable firewall
sudo ufw status                    # Check status
sudo ufw allow 22                  # Allow SSH
sudo ufw allow from 192.168.1.0/24 # Allow subnet
sudo ufw delete 3                  # Delete rule #3
sudo ufw reset                     # Reset all rules
sudo ufw reload                    # Reload configuration
```

UFW provides an excellent balance between simplicity and functionality, making it ideal for system administrators who need effective firewall management without iptables complexity.

For more details, check the manual: `man ufw`

The `traceroute` command

The `traceroute` command is used to trace the path that packets take from your computer to a destination host across a network. It shows each hop (router) along the path and measures the time it takes to reach each hop.

Syntax

623

```
traceroute [options] destination
```

Installation

624

```
# Ubuntu/Debian
sudo apt update && sudo apt install traceroute

# CentOS/RHEL/Fedora
sudo yum install traceroute
# or
sudo dnf install traceroute

# macOS (usually pre-installed)
traceroute

# Check if installed
which traceroute
```

Basic Usage

625

1. **Trace route to a website**

```
traceroute google.com
traceroute github.com
traceroute 8.8.8.8
```

2. **Trace route to IP address**

```
traceroute 192.168.1.1
traceroute 208.67.222.222
```

Options

626

Some popular option flags include:

```
-n           Don't resolve hostnames (show IP addresses only)
-w [sec]     Set timeout for responses (default 5 seconds)
-q [num]     Set number of probe packets per hop (default 3)
-m [hops]    Set maximum number of hops (default 30)
-p [port]    Set destination port (default 33434)
-f [ttl]     Set first TTL value (starting hop)
-g [addr]    Use loose source route gateway
-I           Use ICMP ECHO instead of UDP
-T           Use TCP SYN instead of UDP
-U           Use UDP (default)
-4           Force IPv4
-6           Force IPv6
-s [addr]    Set source address
-i [iface]   Set network interface
```

Examples

1. **Basic traceroute**

```
traceroute google.com
```

2. **Show IP addresses only (no DNS resolution)**

```
traceroute -n google.com
```

3. **Set custom timeout**

```
traceroute -w 10 google.com
```

4. **Use ICMP instead of UDP**

```
traceroute -I google.com
```

5. **Use TCP traceroute**

```
traceroute -T google.com
```

6. **Set maximum hops**

```
traceroute -m 15 google.com
```

7. **Set number of probes per hop**

```
traceroute -q 1 google.com
```

8. **Force IPv6**

```
traceroute -6 ipv6.google.com
```

9. **Set custom port**

```
traceroute -p 80 google.com
```

10. **Start from specific TTL**

```
traceroute -f 5 google.com
```

Understanding Output

Sample traceroute output:

```
traceroute to google.com (172.217.164.174), 30 hops max, 60
byte packets
 1  router.local (192.168.1.1)  1.234 ms  1.123 ms  1.045 ms
 2  10.0.0.1 (10.0.0.1)  12.345 ms  11.234 ms  10.123 ms
 3  isp-gateway.net (203.0.113.1)  25.678 ms  24.567 ms
23.456 ms
 4  * * *
 5  google-router.net (172.217.164.174)  45.123 ms  44.234 ms
43.345 ms
```

Output Explanation:

- **Hop number**: Sequential number of each router
- **Hostname/IP**: Name and IP address of the router
- **Three times**: Round-trip time for three probe packets
- ***** : Indicates timeout or filtered response

Common Use Cases

1. Network Troubleshooting

```
# Check where packets are being dropped
traceroute -n problematic-server.com

# Compare paths to different destinations
traceroute server1.com
traceroute server2.com
```

2. Performance Analysis

```
# Identify slow hops
traceroute -w 10 slow-website.com

# Check latency to different regions
traceroute eu-server.com
traceroute us-server.com
traceroute asia-server.com
```

3. Network Security Analysis

```
# Check if traffic goes through unexpected countries
traceroute -n suspicious-site.com

# Verify VPN routing
traceroute -n whatismyip.com
```

4. ISP Route Analysis 631

```
# Check ISP routing decisions
traceroute -n 8.8.8.8
traceroute -n 1.1.1.1
traceroute -n 208.67.222.222
```

Advanced Techniques

1. TCP Traceroute (tcptraceroute)

```
# Install tcptraceroute
sudo apt install tcptraceroute

# Trace TCP path to web server
sudo tcptraceroute google.com 80
sudo tcptraceroute -n github.com 443
```

2. MTR (My TraceRoute)

```
# Install mtr
sudo apt install mtr

# Continuous traceroute with statistics
mtr google.com
mtr -n google.com     # No DNS resolution
mtr -r google.com     # Report mode
```

3. Paris Traceroute

```
# More accurate for load-balanced networks
sudo apt install paris-traceroute
paris-traceroute google.com
```

Traceroute Variants

1. IPv6 Traceroute

```
# IPv6 traceroute
traceroute6 ipv6.google.com
traceroute -6 ipv6.google.com
```

2. Visual Traceroute Tools

```
# Web-based visual traceroute
# Visit: traceroute-online.com
# or use: mtr with GUI

# Install mtr-gtk for GUI
sudo apt install mtr-gtk
mtr-gtk
```

Analyzing Results

634

1. Identifying Issues

```
# High latency at specific hop
# Look for sudden jumps in response times

# Packet loss
# Look for * * * responses

# Asymmetric routing
# Different paths for different packets
```

2. Geographic Analysis

```
# Use whois to identify hop locations
whois 203.0.113.1

# Use online IP geolocation services
# to map the route geographically
```

Troubleshooting Common Issues

1. Timeouts and Asterisks

```
# Try different protocols
traceroute -I google.com    # ICMP
traceroute -T google.com    # TCP
traceroute -U google.com    # UDP (default)

# Increase timeout
traceroute -w 10 google.com
```

2. Permission Issues

```
# UDP traceroute might need privileges
sudo traceroute google.com

# ICMP definitely needs privileges
sudo traceroute -I google.com
```

3. Firewall Interference

```
# Some firewalls block traceroute
# Try different ports
traceroute -p 53 google.com   # DNS port
traceroute -p 80 google.com   # HTTP port
```

Security Considerations

1. Information Disclosure

```
# Traceroute reveals network topology
# Be careful when sharing results publicly

# Use -n to avoid revealing internal hostnames
traceroute -n destination
```

2. Firewall Evasion

```
# Try different protocols if blocked
traceroute -T -p 443 target.com
traceroute -I target.com
```

Automation and Scripting

1. Batch Traceroute

```bash
#!/bin/bash
# Trace routes to multiple destinations
destinations=("google.com" "github.com" "stackoverflow.com")

for dest in "${destinations[@]}"; do
    echo "Tracing route to $dest"
    traceroute -n "$dest" > "traceroute_$dest.txt"
done
```

2. Monitoring Script

```bash
#!/bin/bash
# Monitor route changes
while true; do
    traceroute -n google.com > "/tmp/trace_$(date +%s).txt"
    sleep 3600  # Check every hour
done
```

3. Route Comparison

```bash
#!/bin/bash
# Compare routes from different locations
echo "Route from current location:"
traceroute -n $1

echo "Route from VPN:"
# Connect to VPN and run again
```

Alternative Commands

1. pathping (Windows equivalent)

```
# On Windows systems
pathping google.com
```

2. mtr (Better alternative)

```
# Continuous monitoring
mtr --report google.com
mtr --report-cycles 10 google.com
```

3. hping3 (Advanced probing)

```
sudo apt install hping3
sudo hping3 -T -p 80 -c 3 google.com
```

Performance Optimization

1. Faster Traceroute

```
# Reduce probes per hop
traceroute -q 1 google.com

# Reduce max hops
traceroute -m 15 google.com

# Skip DNS resolution
traceroute -n google.com
```

2. Detailed Analysis

```
# More probes for accuracy
traceroute -q 5 google.com

# Longer timeout for slow links
traceroute -w 15 google.com
```

Important Notes

640

- Traceroute may not show the actual path in load-balanced networks
- Some routers don't respond to traceroute probes
- Results can vary between runs due to route changes
- ICMP traceroute often works better than UDP
- Modern networks may use ECMP (Equal Cost Multi-Path) routing
- VPNs and proxies will alter the apparent route

The `traceroute` command is essential for network diagnostics, helping identify routing issues, network performance problems, and understanding network topology.

For more details, check the manual: `man traceroute`

The `nmcli` command

The `nmcli` command is used for managing network connections by controlling the NetworkManager, a daemon that handles networking, through command line. The command stands for Network Manager Command Line Interface.

Installation:

The `nmcli` command is already installed by default on most Linux distros. To check if it is installed on your system type:

```
nmcli --version
```

If you don't have nmcli installed you can do it by using the package manager:

Ubuntu or Debian:

```
sudo apt install network manager
```

This will install NetworkManageron in your system. Now we have to start the network managing service.

```
sudo systemctl start NetworkManager
```

Red Hat-based System(such as Fedora, CentOS, REHL):

```
sudo dnf install NetworkManager
```

This will install NetworkManager in your system. Now we have to start the network managing service.

```
sudo systemctl start NetworkManager
```

Arch Linux:

```
sudo pacman -S networkmanager
```

This will install NetworkManager in your system. Now we have to start the network managing service.

```
sudo systemctl start NetworkManager
```

Examples:

1. List all available Wifi networks

```
nmcli device wifi list
```

2. View Network Status

```
nmcli device status
```

3. Connect to a Wifi Network

```
nmcli device wifi connect "SSID_NAME" password "YOUR_PASSWORD"
```

4. Disconnect from a Wifi Network

```
nmcli connection down "CONNECTION_NAME"
```

5. Turn Wifi On/Off

```
nmcli radio wifi on
nmcli radio wifi off
```

, respectively.

6. Turn Bluetooth On/Off

```
nmcli radio bluetooth on
nmcli radio bluetooth off
```

, respectively.

7. To show all connections

```
nmcli connection show
```

8. To show detailed info about specific connections

```
nmcli connection show "CONNECTION_NAME"
```

Syntax:

The general syntax for the nmcli command is as follows:

```
nmcli [OPTIONS...] { help | general | networking | radio |
connection | device | agent | monitor } [COMMAND]
[ARGUMENTS...]
```

Additional Flags and their Functionalities:

Options

Short Flag	Long Flag	Description
-a	--ask	nmcli will stop and ask for any missing required argument(do not use for non interactive options)
-c	--color{yes/no}	It controls color output. yes enables colors, while no disables colors
-h	--help	Prints help information
-p	--pretty	This causes nmcli to produce more user friendly output, eg with headers, and values are aligned
-v	--version	Shows the nmcli version
-f	--fields{field1,...}	This option is used to specify what fields should be printed. Valid fields names differ for specific commands.
-g	--get-value{field1,.}	This option is used to print values from specific field. It is a shortcut for --mode tabular --terse --fields

General Commands

Command	Description
`nmcli general status`	Show overall NetworkManager status
`nmcli general hostname`	Display current hostname

Networking Commands

Command	Description
`nmcli networking on`	Enable all networking
`nmcli networking off`	Disable all networking
`nmcli networking connectivity`	Check network connectivity status

Radio Commands

Command	Description
`nmcli radio wifi on`	Enable Wi-Fi radio
`nmcli radio wifi off`	Disable Wi-Fi radio
`nmcli radio all`	Show status of all radio switches
`nmcli radio wifi`	Show Wi-Fi radio status

Connection Management Commands

Command	Description
`nmcli connection show`	List all saved connection profiles
`nmcli connection show --active`	List only active connections
`nmcli connection show "NAME"`	Show detailed info about specific connection
`nmcli connection up "NAME"`	Activate a connection
`nmcli connection down "NAME"`	Deactivate a connection
`nmcli connection modify "NAME" [OPTIONS]`	Modify connection settings
`nmcli connection delete "NAME"`	Delete a connection profile
`nmcli connection reload`	Reload all connection files from disk

Device Management Commands

646

Command	Description
`nmcli device status`	Show status of all devices
`nmcli device show "DEVICE"`	Show detailed info for specific device
`nmcli device disconnect "DEVICE"`	Disconnect from a device
`nmcli device wifi list`	List all available Wi-Fi networks
`nmcli device wifi connect "SSID" password "PWD"`	Connect to password-protected Wi-Fi

The `badblocks` command

The `badblocks` command is used to search for bad blocks on a storage device. It can scan hard drives, SSDs, USB drives, and other storage media to identify sectors that cannot reliably store data. This is essential for maintaining data integrity and system reliability.

Syntax

648

```
badblocks [options] device [last-block] [first-block]
```

Key Features

649

- **Non-destructive Testing**: Read-only tests by default
- **Destructive Testing**: Write tests for thorough checking
- **Pattern Testing**: Uses specific patterns to detect errors
- **Progress Reporting**: Shows scan progress and results
- **Output Options**: Various formats for different use cases

Basic Usage

650

Simple Read Test

```
# Basic read test (non-destructive)
sudo badblocks /dev/sdb

# Test specific partition
sudo badblocks /dev/sdb1

# Verbose output
sudo badblocks -v /dev/sdb
```

Show Progress

```
# Show progress during scan
sudo badblocks -s /dev/sdb

# Show progress with verbose output
sudo badblocks -sv /dev/sdb
```

Testing Modes

651

1. Read-Only Test (Default)

```
# Non-destructive read test
sudo badblocks /dev/sdb

# Read test with verbose output
sudo badblocks -v /dev/sdb

# Read test showing progress
sudo badblocks -sv /dev/sdb
```

2. Non-Destructive Read-Write Test

```
# Non-destructive read-write test
sudo badblocks -n /dev/sdb

# Backup original data, test, then restore
sudo badblocks -nv /dev/sdb
```

3. Destructive Write Test

```
# WARNING: This will destroy all data!
sudo badblocks -w /dev/sdb

# Destructive test with verbose output
sudo badblocks -wv /dev/sdb

# Write test with progress
sudo badblocks -wsv /dev/sdb
```

Common Options

652

Basic Options

```
# -v: Verbose output
sudo badblocks -v /dev/sdb

# -s: Show progress
sudo badblocks -s /dev/sdb

# -o: Output bad blocks to file
sudo badblocks -o badblocks.txt /dev/sdb

# -b: Specify block size (default 1024)
sudo badblocks -b 4096 /dev/sdb
```

Advanced Options

```
# -c: Number of blocks to test at once
sudo badblocks -c 65536 /dev/sdb

# -p: Number of passes (for write tests)
sudo badblocks -w -p 2 /dev/sdb

# -t: Test pattern for write tests
sudo badblocks -w -t 0xaa /dev/sdb

# -f: Force operation even if mounted
sudo badblocks -f /dev/sdb
```

Practical Examples

1. Basic Drive Health Check

```
# Unmount the device first
sudo umount /dev/sdb1

# Run basic read test
sudo badblocks -sv /dev/sdb

# Save results to file
sudo badblocks -sv -o badblocks_sdb.txt /dev/sdb
```

2. Thorough Drive Testing

```
# Full destructive test (destroys data!)
# Make sure to backup first!
sudo badblocks -wsv /dev/sdb

# Multiple pass destructive test
sudo badblocks -wsv -p 3 /dev/sdb
```

3. Testing Specific Range

```
# Test blocks 1000 to 2000
sudo badblocks -sv /dev/sdb 2000 1000

# Test first 1GB (assuming 4K blocks)
sudo badblocks -sv -b 4096 /dev/sdb 262144 0
```

4. Integration with fsck   654

```
# Create bad blocks file
sudo badblocks -sv -o /tmp/badblocks /dev/sdb1

# Use with fsck to mark bad blocks
sudo fsck.ext4 -l /tmp/badblocks /dev/sdb1

# For new filesystem
sudo mke2fs -l /tmp/badblocks /dev/sdb1
```

Different Block Sizes

Choosing Block Size

```
# 1KB blocks (default)
sudo badblocks -b 1024 /dev/sdb

# 4KB blocks (common for modern drives)
sudo badblocks -b 4096 /dev/sdb

# 512 byte blocks (traditional sector size)
sudo badblocks -b 512 /dev/sdb

# Match filesystem block size
sudo tune2fs -l /dev/sdb1 | grep "Block size"
sudo badblocks -b 4096 /dev/sdb1
```

Test Patterns

Write Test Patterns

```
# Alternating pattern (0xaa = 10101010)
sudo badblocks -w -t 0xaa /dev/sdb

# All ones pattern
sudo badblocks -w -t 0xff /dev/sdb

# All zeros pattern
sudo badblocks -w -t 0x00 /dev/sdb

# Random pattern
sudo badblocks -w -t random /dev/sdb
```

Multiple Patterns

```
# Test with multiple patterns
sudo badblocks -w -t 0xaa -t 0x55 -t 0xff -t 0x00 /dev/sdb

# Four-pass test with different patterns
sudo badblocks -wsv -p 4 -t 0xaa -t 0x55 -t 0xff -t 0x00
/dev/sdb
```

Output and Reporting

Standard Output

```
# Basic output (just bad block numbers)
sudo badblocks /dev/sdb

# Verbose output with details
sudo badblocks -v /dev/sdb

# Progress indicator
sudo badblocks -s /dev/sdb
```

Save Results to File

```
# Save bad blocks list
sudo badblocks -o badblocks.txt /dev/sdb

# Append to existing file
sudo badblocks -o badblocks.txt /dev/sdb >> all_badblocks.txt

# Save with verbose output to different files
sudo badblocks -v -o badblocks.txt /dev/sdb 2> scan_log.txt
```

Working with Different Storage Types

1. Traditional Hard Drives

```
# Standard test for HDDs
sudo badblocks -sv /dev/sda

# Thorough test with multiple passes
sudo badblocks -wsv -p 4 /dev/sda
```

2. Solid State Drives

```
# Read-only test (preferred for SSDs)
sudo badblocks -sv /dev/sdb

# Non-destructive test
sudo badblocks -nsv /dev/sdb

# Avoid excessive write tests on SSDs
```

3. USB Drives

```
# Test USB drive
sudo badblocks -sv /dev/sdc

# Fast test for quick verification
sudo badblocks -sv -c 65536 /dev/sdc
```

4. SD Cards

```
# Test SD card
sudo badblocks -sv /dev/mmcblk0

# Write test for fake capacity detection
sudo badblocks -wsv /dev/mmcblk0
```

Integration with File Systems

1. ext2/ext3/ext4

```
# Create filesystem with bad block check
sudo mke2fs -c /dev/sdb1

# Check existing filesystem
sudo fsck.ext4 -c /dev/sdb1

# Thorough check
sudo fsck.ext4 -cc /dev/sdb1
```

2. Using with e2fsck

```
# Create bad blocks list
sudo badblocks -sv -o /tmp/badblocks /dev/sdb1

# Apply to filesystem
sudo e2fsck -l /tmp/badblocks /dev/sdb1
```

Monitoring and Automation

1. Scheduled Checking

```bash
# Create script for regular checking
#!/bin/bash
DEVICE="/dev/sdb"
LOGFILE="/var/log/badblocks_$(date +%Y%m%d).log"

sudo badblocks -sv -o /tmp/badblocks "$DEVICE" > "$LOGFILE"
2>&1

if [ -s /tmp/badblocks ]; then
    echo "Bad blocks found on $DEVICE!" | mail -s "Bad Blocks
Alert" admin@domain.com
fi
```

2. SMART Integration

```bash
# Check SMART status first
sudo smartctl -a /dev/sdb

# Run badblocks if SMART shows issues
sudo smartctl -t short /dev/sdb
sleep 300
sudo smartctl -a /dev/sdb
sudo badblocks -sv /dev/sdb
```

Performance Considerations

1. Speed Optimization

```
# Increase block count for faster scanning
sudo badblocks -c 65536 /dev/sdb

# Use larger block size
sudo badblocks -b 4096 -c 16384 /dev/sdb

# Combine options for maximum speed
sudo badblocks -sv -b 4096 -c 65536 /dev/sdb
```

2. System Impact

```
# Run with lower priority
sudo nice -n 19 badblocks -sv /dev/sdb

# Limit I/O impact
sudo ionice -c 3 badblocks -sv /dev/sdb

# Combine nice and ionice
sudo nice -n 19 ionice -c 3 badblocks -sv /dev/sdb
```

Interpreting Results

1. No Bad Blocks

```
# Output: "Pass completed, 0 bad blocks found."
# This indicates a healthy drive
```

2. Bad Blocks Found

```
# Output shows block numbers of bad sectors
# Example:
# Pass completed, 5 bad blocks found. (0/0/5 errors)
# 1024
# 2048
# 4096
# 8192
# 16384
```

3. Understanding Block Numbers

```
# Convert block numbers to byte offsets
# Block 1024 with 4KB block size = 1024 * 4096 = 4,194,304
bytes
# This helps locate physical position on drive
```

Troubleshooting

1. Permission Denied

```
# Must run as root
sudo badblocks /dev/sdb

# Check device permissions
ls -l /dev/sdb
```

2. Device Busy

```
# Unmount all partitions first
sudo umount /dev/sdb1
sudo umount /dev/sdb2

# Check for active processes
lsof /dev/sdb*

# Use force option if necessary
sudo badblocks -f /dev/sdb
```

3. Slow Performance

```
# Increase block count
sudo badblocks -c 65536 /dev/sdb

# Use appropriate block size
sudo badblocks -b 4096 /dev/sdb

# Check system load
iostat 1
```

Safety Considerations

1. Data Backup

```
# Always backup before destructive tests
sudo dd if=/dev/sdb of=/backup/sdb_backup.img bs=1M

# Or use filesystem-level backup
sudo rsync -av /mount/point/ /backup/location/
```

2. Drive Health Assessment

```
# Check SMART data first
sudo smartctl -a /dev/sdb

# Look for reallocated sectors
sudo smartctl -A /dev/sdb | grep Reallocated
```

3. When to Replace Drive

```
# Replace if:
# - Many bad blocks found (>50-100)
# - Bad blocks increasing over time
# - SMART indicates drive failure
# - Critical system drive affected
```

Alternative Tools

1. SMART Tools

```
# Use smartctl for health monitoring
sudo smartctl -t long /dev/sdb
sudo smartctl -a /dev/sdb
```

2. Manufacturer Tools

```
# Many manufacturers provide specific tools
# - Western Digital: WD Data Lifeguard
# - Seagate: SeaTools
# - Samsung: Samsung Magician
```

Important Notes

- Read tests are safe and non-destructive
- Write tests destroy all data on the device
- Always unmount devices before testing
- Test results should be interpreted with other health indicators
- Regular testing helps prevent data loss
- Consider drive replacement if many bad blocks are found
- Modern drives have spare sectors for bad block management

The `badblocks` command is an essential tool for maintaining storage device health and preventing data loss.

For more details, check the manual: `man badblocks`

The `fsck` command

The `fsck` (file system check) command is used to check and repair Linux file systems. It can detect and fix various file system inconsistencies, corruption issues, and structural problems. It's an essential tool for maintaining file system integrity and recovering from system crashes.

Syntax

669

```
fsck [options] [filesystem...]
```

Key Features

670

- **Multiple File System Support**: Works with ext2, ext3, ext4, XFS, and more
- **Automatic Detection**: Can auto-detect file system type
- **Interactive Repair**: Prompts for confirmation before fixes
- **Batch Mode**: Can run automatically without user interaction
- **Read-Only Mode**: Check without making changes

Basic Usage

Simple File System Check

```
# Check specific partition
sudo fsck /dev/sdb1

# Check by mount point
sudo fsck /home

# Check with verbose output
sudo fsck -v /dev/sdb1
```

Check All File Systems

```
# Check all file systems in /etc/fstab
sudo fsck -A

# Check all except root
sudo fsck -AR

# Check all with progress
sudo fsck -AV
```

Common Options

Basic Options

```
# -y: Answer "yes" to all questions
sudo fsck -y /dev/sdb1

# -n: Answer "no" to all questions (read-only)
sudo fsck -n /dev/sdb1

# -f: Force check even if file system seems clean
sudo fsck -f /dev/sdb1

# -v: Verbose output
sudo fsck -v /dev/sdb1
```

Advanced Options

```
# -p: Automatically repair (preen mode)
sudo fsck -p /dev/sdb1

# -r: Interactive repair mode
sudo fsck -r /dev/sdb1

# -t: Specify file system type
sudo fsck -t ext4 /dev/sdb1

# -C: Show progress bar
sudo fsck -C /dev/sdb1
```

File System Specific Commands

1. ext2/ext3/ext4 (e2fsck)

```
# Check ext4 file system
sudo fsck.ext4 /dev/sdb1

# Force check
sudo e2fsck -f /dev/sdb1

# Fix bad blocks
sudo e2fsck -c /dev/sdb1

# Thorough check with bad block scan
sudo e2fsck -cc /dev/sdb1
```

2. XFS (xfs_repair)

```
# Check XFS file system (read-only)
sudo xfs_repair -n /dev/sdb1

# Repair XFS file system
sudo xfs_repair /dev/sdb1

# Verbose repair
sudo xfs_repair -v /dev/sdb1
```

3. FAT32 (fsck.fat)

```
# Check FAT32 file system
sudo fsck.fat /dev/sdb1

# Repair FAT32
sudo fsck.fat -r /dev/sdb1

# Verbose check
sudo fsck.fat -v /dev/sdb1
```

Practical Examples

1. Check Before Mounting

```
# Always check before mounting suspicious drives
sudo fsck -n /dev/sdb1

# If clean, mount normally
sudo mount /dev/sdb1 /mnt/usb

# If errors found, repair first
sudo fsck -y /dev/sdb1
sudo mount /dev/sdb1 /mnt/usb
```

2. System Recovery After Crash

```
# Boot from live CD/USB
# Check root file system
sudo fsck -f /dev/sda1

# Check other partitions
sudo fsck -f /dev/sda2
sudo fsck -f /dev/sda3

# Reboot if repairs were made
sudo reboot
```

3. Scheduled Maintenance

```
# Force check on all file systems
sudo fsck -Af

# Check with automatic repair
sudo fsck -Ap

# Check with progress indicators
sudo fsck -AVC
```

Working with Different Scenarios

1. Read-Only Check

```
# Check without making changes
sudo fsck -n /dev/sdb1

# Verbose read-only check
sudo fsck -nv /dev/sdb1

# Generate report only
sudo fsck -n /dev/sdb1 > fsck_report.txt 2>&1
```

2. Automatic Repair

```
# Repair automatically (dangerous!)
sudo fsck -y /dev/sdb1

# Safer automatic repair
sudo fsck -p /dev/sdb1

# Batch mode for multiple file systems
sudo fsck -Ap
```

3. Interactive Repair

```
# Interactive mode (default)
sudo fsck /dev/sdb1

# Ask before each fix
sudo fsck -r /dev/sdb1

# Show detailed information
sudo fsck -rv /dev/sdb1
```

Boot-Time File System Checks

1. Automatic Checks

```
# Configure automatic checks in /etc/fstab
# Sixth field controls fsck behavior:
# 0 = no check
# 1 = check first (root filesystem)
# 2 = check after root filesystem

# Example /etc/fstab entry:
# /dev/sda1 / ext4 defaults 1 1
# /dev/sda2 /home ext4 defaults 1 2
```

2. Force Check on Next Boot

```
# Create forcefsck file (traditional method)
sudo touch /forcefsck

# Or use tune2fs for ext filesystems
sudo tune2fs -C 1 -c 1 /dev/sda1

# Set maximum mount count
sudo tune2fs -c 30 /dev/sda1
```

3. Check Intervals

```
# Set check interval (ext filesystems)
sudo tune2fs -i 30d /dev/sda1  # Check every 30 days
sudo tune2fs -i 0 /dev/sda1    # Disable time-based checks

# Set mount count interval
sudo tune2fs -c 25 /dev/sda1   # Check every 25 mounts
sudo tune2fs -c 0 /dev/sda1    # Disable mount-based checks
```

Troubleshooting Common Issues

1. Unmountable File System

```
# Try read-only check first
sudo fsck -n /dev/sdb1

# If errors found, try repair
sudo fsck -y /dev/sdb1

# For severe corruption
sudo fsck -f /dev/sdb1
```

2. Bad Superblock

```
# List backup superblocks
sudo mke2fs -n /dev/sdb1

# Use backup superblock
sudo e2fsck -b 32768 /dev/sdb1

# Try different backup
sudo e2fsck -b 98304 /dev/sdb1
```

3. Lost+Found Directory

```
# Recovered files appear in lost+found
ls -la /mnt/partition/lost+found/

# Files are numbered by inode
# Use file command to identify type
file /mnt/partition/lost+found/*

# Restore files based on content
```

Advanced Repair Options

1. Bad Block Handling

```
# Scan for bad blocks during check
sudo e2fsck -c /dev/sdb1

# Thorough bad block scan
sudo e2fsck -cc /dev/sdb1

# Use existing bad block list
sudo badblocks -sv /dev/sdb1 > badblocks.list
sudo e2fsck -l badblocks.list /dev/sdb1
```

2. Journal Recovery

```
# ext3/ext4 journal recovery
sudo e2fsck -y /dev/sdb1

# Force journal recovery
sudo tune2fs -O ^has_journal /dev/sdb1
sudo e2fsck -f /dev/sdb1
sudo tune2fs -j /dev/sdb1
```

3. Inode Problems

```
# Check inode usage
sudo e2fsck -D /dev/sdb1

# Rebuild directory index
sudo e2fsck -D -f /dev/sdb1

# Fix inode count problems
sudo e2fsck -f /dev/sdb1
```

Monitoring and Logging

1. Check Results

```
# View fsck results
dmesg | grep -i fsck

# Check system logs
journalctl | grep fsck
tail -f /var/log/messages | grep fsck
```

2. File System Status

```
# Check filesystem status
sudo tune2fs -l /dev/sda1 | grep -i "filesystem state"

# Check last fsck time
sudo tune2fs -l /dev/sda1 | grep -i "last checked"

# Check mount count
sudo tune2fs -l /dev/sda1 | grep -i "mount count"
```

3. Automated Monitoring

```
# Script to check filesystem health
#!/bin/bash
for fs in $(awk '$3 ~ /^ext[234]$/ && $2 != "/" {print $1}'
/etc/fstab); do
    echo "Checking $fs..."
    sudo fsck -n "$fs" || echo "Errors found on $fs"
done
```

Prevention and Best Practices

1. Regular Maintenance

```
# Schedule regular checks
# Add to crontab for non-critical systems
# 0 3 * * 0 /sbin/fsck -Ap > /var/log/fsck.log 2>&1
```

2. Proper Shutdown

```
# Always shutdown properly
sudo shutdown -h now

# Use sync before emergency shutdown
sync
sudo shutdown -h now
```

3. UPS Protection

```
# Install UPS monitoring
sudo apt install apcupsd  # For APC UPS
sudo apt install nut       # Network UPS Tools

# Configure automatic shutdown on power loss
```

Recovery Scenarios

1. Boot Failure

```
# Boot from live USB/CD
# Mount root filesystem read-only
sudo mount -o ro /dev/sda1 /mnt

# Copy important data
sudo cp -r /mnt/home/user/important /backup/

# Unmount and check
sudo umount /mnt
sudo fsck -f /dev/sda1
```

2. Data Recovery

```
# Use ddrescue for severely damaged drives
sudo ddrescue /dev/sdb /backup/disk.img /backup/disk.log

# Then fsck the image
sudo fsck -f /backup/disk.img

# Mount and recover data
sudo mount -o loop /backup/disk.img /mnt/recovery
```

3. Multiple Errors

```
# Progressive repair approach
sudo fsck -n /dev/sdb1          # Check first
sudo fsck -p /dev/sdb1          # Auto-repair safe issues
sudo fsck -y /dev/sdb1          # Force repair remaining
issues
sudo fsck -f /dev/sdb1          # Final thorough check
```

Performance Considerations

1. Speed Optimization

```
# Use progress indicator
sudo fsck -C /dev/sdb1

# Parallel checking (careful with dependencies)
sudo fsck -A -P

# Skip time-consuming checks when appropriate
sudo fsck -p /dev/sdb1
```

2. System Impact

```
# Run during low-activity periods
# Schedule during maintenance windows
# Use ionice for lower priority
sudo ionice -c 3 fsck /dev/sdb1
```

Important Safety Notes

- **Always unmount** file systems before checking
- **Backup important data** before repairs
- **Never interrupt** fsck during operation
- **Use read-only mode** first to assess damage
- **Understand risks** of automatic repair modes
- **Boot from live media** for root filesystem checks
- **Have recovery plan** ready before starting repairs

Exit Codes

691

```
# fsck exit codes:
# 0: No errors
# 1: Filesystem errors corrected
# 2: System should be rebooted
# 4: Filesystem errors left uncorrected
# 8: Operational error
# 16: Usage or syntax error
# 32: Checking canceled by user request
# 128: Shared-library error
```

The `fsck` command is crucial for maintaining file system integrity and recovering from corruption issues.

For more details, check the manual: `man fsck`

The `mkfs` command

The `mkfs` (make file system) command is used to create file systems on storage devices. It formats partitions or entire disks with specific file system types like ext4, XFS, FAT32, and others. This is essential for preparing storage devices for use with Linux systems.

Syntax

693

```
mkfs [options] [-t type] device
mkfs.type [options] device
```

Key Features

694

- **Multiple File System Support**: ext2/3/4, XFS, FAT32, NTFS, and more
- **Custom Parameters**: Block size, inode ratio, labels, and features
- **Quick vs Full Format**: Fast formatting or thorough initialization
- **Advanced Options**: Encryption, compression, and performance tuning

Basic Usage

695

Create File Systems

```
# Auto-detect and create default filesystem
sudo mkfs /dev/sdb1

# Specify filesystem type
sudo mkfs -t ext4 /dev/sdb1

# Direct filesystem creation
sudo mkfs.ext4 /dev/sdb1
sudo mkfs.xfs /dev/sdb2
sudo mkfs.fat /dev/sdb3
```

Common File System Types

```
# ext4 (recommended for Linux)
sudo mkfs.ext4 /dev/sdb1

# XFS (good for large files)
sudo mkfs.xfs /dev/sdb1

# FAT32 (cross-platform compatibility)
sudo mkfs.fat -F32 /dev/sdb1

# NTFS (Windows compatibility)
sudo mkfs.ntfs /dev/sdb1
```

File System Specific Options

1. ext4 File System

```
# Basic ext4 creation
sudo mkfs.ext4 /dev/sdb1

# With label
sudo mkfs.ext4 -L MyData /dev/sdb1

# Custom block size
sudo mkfs.ext4 -b 4096 /dev/sdb1

# Custom inode ratio
sudo mkfs.ext4 -i 4096 /dev/sdb1

# With journal
sudo mkfs.ext4 -J size=128 /dev/sdb1
```

Advanced ext4 Options

```
# Disable journaling (ext2-like)
sudo mkfs.ext4 -O ^has_journal /dev/sdb1

# Enable encryption support
sudo mkfs.ext4 -O encrypt /dev/sdb1

# Set reserved blocks percentage
sudo mkfs.ext4 -m 1 /dev/sdb1

# Custom UUID
sudo mkfs.ext4 -U 12345678-1234-1234-1234-123456789012
/dev/sdb1
```

2. XFS File System

```
# Basic XFS creation
sudo mkfs.xfs /dev/sdb1

# Force creation (overwrite existing)
sudo mkfs.xfs -f /dev/sdb1

# With label
sudo mkfs.xfs -L MyXFS /dev/sdb1

# Custom block size
sudo mkfs.xfs -b size=4096 /dev/sdb1

# Custom sector size
sudo mkfs.xfs -s size=4096 /dev/sdb1
```

Advanced XFS Options

```
# Separate log device
sudo mkfs.xfs -l logdev=/dev/sdc1 /dev/sdb1

# Real-time device
sudo mkfs.xfs -r rtdev=/dev/sdd1 /dev/sdb1

# Custom inode size
sudo mkfs.xfs -i size=512 /dev/sdb1

# Allocation group size
sudo mkfs.xfs -d agcount=8 /dev/sdb1
```

3. FAT32 File System

```
# Basic FAT32 creation
sudo mkfs.fat -F32 /dev/sdb1

# With label
sudo mkfs.fat -F32 -n USBDRIVE /dev/sdb1

# Custom cluster size
sudo mkfs.fat -F32 -s 8 /dev/sdb1

# Custom volume ID
sudo mkfs.fat -F32 -i 12345678 /dev/sdb1
```

4. NTFS File System

```
# Basic NTFS creation
sudo mkfs.ntfs /dev/sdb1

# Quick format
sudo mkfs.ntfs -Q /dev/sdb1

# With label
sudo mkfs.ntfs -L WindowsData /dev/sdb1

# Custom cluster size
sudo mkfs.ntfs -c 4096 /dev/sdb1
```

Practical Examples

1. Preparing a USB Drive

```
# Check device name
lsblk

# Create partition table (if needed)
sudo fdisk /dev/sdc
# or
sudo cfdisk /dev/sdc

# Format with FAT32 for compatibility
sudo mkfs.fat -F32 -n MYUSB /dev/sdc1

# Mount and test
sudo mkdir /mnt/usb
sudo mount /dev/sdc1 /mnt/usb
```

2. Setting Up a Data Drive

```
# Create ext4 with optimal settings
sudo mkfs.ext4 -L DataDrive -b 4096 -m 1 /dev/sdb1

# Create mount point
sudo mkdir /mnt/data

# Add to fstab for automatic mounting
echo "LABEL=DataDrive /mnt/data ext4 defaults 0 2" | sudo tee -a /etc/fstab

# Mount
sudo mount /mnt/data
```

3. High-Performance Storage

```
# XFS for large files and high performance
sudo mkfs.xfs -f -L HighPerf -b size=4096 -d agcount=8
/dev/sdb1

# Or ext4 with performance optimizations
sudo mkfs.ext4 -L HighPerf -b 4096 -E stride=32,stripe-
width=64 /dev/sdb1
```

4. SSD Optimization

```
# ext4 for SSD with TRIM support
sudo mkfs.ext4 -L SSD -b 4096 -E discard /dev/sdb1

# XFS for SSD
sudo mkfs.xfs -f -L SSD -K /dev/sdb1

# Add discard option in fstab
# /dev/sdb1 /mnt/ssd ext4 defaults,discard 0 2
```

Advanced Configuration

1. Large File Systems

```
# ext4 for very large filesystems
sudo mkfs.ext4 -T largefile4 /dev/sdb1

# XFS with optimizations for large files
sudo mkfs.xfs -f -i size=512 -d agcount=32 /dev/sdb1

# Increase inode ratio for many small files
sudo mkfs.ext4 -T small /dev/sdb1
```

2. RAID Configurations

```
# ext4 for RAID arrays
sudo mkfs.ext4 -b 4096 -E stride=16,stripe-width=32 /dev/md0

# XFS for RAID
sudo mkfs.xfs -f -d su=64k,sw=2 /dev/md0

# Where:
# stride = (chunk-size / block-size)
# stripe-width = (number-of-data-disks * stride)
```

3. Encryption Support

```
# ext4 with encryption
sudo mkfs.ext4 -O encrypt /dev/sdb1

# Setup LUKS encryption first
sudo cryptsetup luksFormat /dev/sdb1
sudo cryptsetup open /dev/sdb1 encrypted_disk
sudo mkfs.ext4 /dev/mapper/encrypted_disk
```

Backup and Safety

1. Check Before Formatting

```
# Verify device
lsblk /dev/sdb
fdisk -l /dev/sdb

# Check for mounted filesystems
mount | grep /dev/sdb
lsof /dev/sdb*

# Backup important data
sudo dd if=/dev/sdb of=/backup/sdb_backup.img bs=1M
```

2. Partition Table Backup

```
# Backup partition table
sudo sfdisk -d /dev/sdb > sdb_partition_table.txt

# Restore if needed
sudo sfdisk /dev/sdb < sdb_partition_table.txt
```

3. Test Before Use

```
# Create filesystem
sudo mkfs.ext4 /dev/sdb1

# Mount and test
sudo mkdir /mnt/test
sudo mount /dev/sdb1 /mnt/test

# Basic functionality test
echo "test" | sudo tee /mnt/test/testfile
cat /mnt/test/testfile
sudo rm /mnt/test/testfile

# Unmount
sudo umount /mnt/test
```

Troubleshooting

1. Device Busy Errors

```
# Check what's using the device
lsof /dev/sdb1
fuser -v /dev/sdb1

# Unmount if mounted
sudo umount /dev/sdb1

# Kill processes if necessary
sudo fuser -k /dev/sdb1
```

2. Insufficient Space

```
# Check available space
fdisk -l /dev/sdb

# Verify partition size
cat /proc/partitions

# Check for existing filesystems
file -s /dev/sdb1
```

3. Permission Issues

```
# Must run as root
sudo mkfs.ext4 /dev/sdb1

# Check device permissions
ls -l /dev/sdb1

# Add user to disk group if needed
sudo usermod -a -G disk username
```

Performance Optimization

1. Block Size Selection

```
# For small files (default: 4096)
sudo mkfs.ext4 -b 1024 /dev/sdb1

# For large files
sudo mkfs.ext4 -b 4096 /dev/sdb1

# For very large files (ext4 only)
sudo mkfs.ext4 -b 65536 /dev/sdb1
```

2. Inode Configuration

```
# More inodes for many small files
sudo mkfs.ext4 -i 1024 /dev/sdb1

# Fewer inodes for large files
sudo mkfs.ext4 -i 16384 /dev/sdb1

# Fixed number of inodes
sudo mkfs.ext4 -N 1000000 /dev/sdb1
```

3. Journal Optimization

```
# Large journal for write-heavy workloads
sudo mkfs.ext4 -J size=128 /dev/sdb1

# External journal device
sudo mkfs.ext4 -J device=/dev/sdc1 /dev/sdb1

# Disable journal for read-only media
sudo mkfs.ext4 -O ^has_journal /dev/sdb1
```

Specialized Use Cases

1. Bootable Media

```
# FAT32 for UEFI boot
sudo mkfs.fat -F32 -n BOOT /dev/sdb1

# ext4 for Linux boot
sudo mkfs.ext4 -L BOOT /dev/sdb2

# Install bootloader after filesystem creation
```

2. Network Storage

```
# XFS for NFS exports
sudo mkfs.xfs -f -L NFSSHARE /dev/sdb1

# ext4 for Samba shares
sudo mkfs.ext4 -L SAMBA /dev/sdb1
```

3. Container Storage

```
# ext4 with specific features for containers
sudo mkfs.ext4 -O project -L CONTAINERS /dev/sdb1

# XFS with project quotas
sudo mkfs.xfs -f -i size=512 /dev/sdb1
```

Monitoring and Verification

1. Filesystem Information

```
# ext4 information
sudo tune2fs -l /dev/sdb1

# XFS information
sudo xfs_info /dev/sdb1

# General filesystem info
df -T /mnt/mountpoint
```

2. Health Checks

```
# Check filesystem after creation
sudo fsck -n /dev/sdb1

# Mount and verify
sudo mount /dev/sdb1 /mnt/test
sudo df -h /mnt/test
sudo ls -la /mnt/test
```

Best Practices

1. Planning

```
# Determine optimal filesystem type based on use case:
# - ext4: General purpose, good for most Linux use cases
# - XFS: Large files, high performance, NAS/database storage
# - FAT32: Cross-platform compatibility, small devices
# - NTFS: Windows compatibility
```

2. Labeling

```
# Always use descriptive labels
sudo mkfs.ext4 -L "SystemData" /dev/sdb1
sudo mkfs.xfs -L "MediaStorage" /dev/sdb2
sudo mkfs.fat -F32 -n "BACKUP" /dev/sdb3
```

3. Documentation

```
# Document filesystem configuration
echo "Created: $(date)" > /root/filesystem_log.txt
echo "Device: /dev/sdb1" >> /root/filesystem_log.txt
echo "Type: ext4" >> /root/filesystem_log.txt
echo "Label: DataDrive" >> /root/filesystem_log.txt
```

Important Notes

712

- **Always backup data** before formatting
- **Verify device name** carefully to avoid data loss
- **Unmount filesystem** before formatting
- **Choose appropriate filesystem** for your use case
- **Consider performance requirements** when selecting options
- **Test filesystem** after creation
- **Document configuration** for future reference

The `mkfs` command is fundamental for preparing storage devices and should be used with careful consideration of requirements and safety procedures.

For more details, check the manual: `man mkfs`

The `cpio` command

The `cpio` (copy in, copy out) command is a versatile archiving utility that can create and extract archives, copy files, and handle special file types. It's particularly useful for system backups, creating initramfs images, and working with tape archives.

Syntax

714

```
cpio [options] < name-list
cpio [options] -i [patterns] < archive
cpio [options] -o > archive
cpio [options] -p destination-directory
```

Operating Modes

1. Copy-out mode (-o)

```
# Create archive from file list
find . -name "*.txt" | cpio -o > archive.cpio

# Create compressed archive
find . -type f | cpio -o | gzip > archive.cpio.gz

# Verbose output
find . -name "*.conf" | cpio -ov > config_backup.cpio
```

2. Copy-in mode (-i)

```
# Extract archive
cpio -i < archive.cpio

# Extract to specific directory
cd /restore && cpio -i < /backup/archive.cpio

# Extract specific files
cpio -i "*.txt" < archive.cpio
```

3. Pass-through mode (-p)

```
# Copy files to another directory
find /source -type f | cpio -p /destination

# Preserve attributes while copying
find /etc -name "*.conf" | cpio -pdm /backup/configs
```

Common Options

Basic Options

```
# -o: Copy-out (create archive)
find . | cpio -o > archive.cpio

# -i: Copy-in (extract archive)
cpio -i < archive.cpio

# -p: Pass-through (copy files)
find . | cpio -p /destination

# -v: Verbose output
find . | cpio -ov > archive.cpio

# -t: List archive contents
cpio -tv < archive.cpio
```

Advanced Options

```
# -d: Create directories as needed
cpio -id < archive.cpio

# -m: Preserve modification times
cpio -im < archive.cpio

# -u: Unconditional extraction (overwrite)
cpio -iu < archive.cpio

# -H: Specify archive format
cpio -oH newc > archive.cpio

# -B: Use 5120-byte blocks
cpio -oB > archive.cpio
```

Archive Formats

Available Formats

```
# Binary format (default, obsolete)
cpio -o > archive.cpio

# New ASCII format (recommended)
cpio -oH newc > archive.cpio

# Old ASCII format
cpio -oH odc > archive.cpio

# CRC format
cpio -oH crc > archive.cpio

# TAR format
cpio -oH tar > archive.tar

# USTAR format
cpio -oH ustar > archive.tar
```

Practical Examples

1. System Backup

```
# Backup entire system (excluding certain directories)
find / -path /proc -prune -o -path /sys -prune -o -path /dev -
prune -o -print | \
sudo cpio -oH newc | gzip > system_backup.cpio.gz

# Backup specific directories
find /etc /home /var -type f | cpio -oH newc >
important_files.cpio

# Backup with verbose output
find /home/user -type f | cpio -ovH newc > user_backup.cpio
```

2. Creating initramfs

```
# Create initramfs image (common in Linux boot process)
cd /tmp/initramfs
find . | cpio -oH newc | gzip > /boot/initramfs.img

# Extract existing initramfs
cd /tmp/extract
zcat /boot/initramfs.img | cpio -id
```

3. Selective File Operations

```
# Archive only configuration files
find /etc -name "*.conf" -o -name "*.cfg" | cpio -oH newc >
configs.cpio

# Archive files modified in last 7 days
find /home -type f -mtime -7 | cpio -oH newc >
recent_files.cpio

# Archive files larger than 1MB
find /var/log -type f -size +1M | cpio -oH newc >
large_logs.cpio
```

4. Working with Compressed Archives

```
# Create compressed archive
find . -type f | cpio -oH newc | gzip > archive.cpio.gz
find . -type f | cpio -oH newc | bzip2 > archive.cpio.bz2
find . -type f | cpio -oH newc | xz > archive.cpio.xz

# Extract compressed archives
zcat archive.cpio.gz | cpio -id
bzcat archive.cpio.bz2 | cpio -id
xzcat archive.cpio.xz | cpio -id
```

Special File Types

1. Device Files and Special Files

```
# Archive including device files
find /dev -type c -o -type b | sudo cpio -oH newc >
device_files.cpio

# Archive symbolic links
find . -type l | cpio -oH newc > symlinks.cpio

# Archive with all file types preserved
find . | sudo cpio -oH newc > complete_backup.cpio
```

2. Preserving Attributes

```
# Preserve ownership and permissions
find . | sudo cpio -oH newc > backup.cpio
sudo cpio -idm < backup.cpio

# Preserve modification times
cpio -im < archive.cpio

# Preserve all attributes in pass-through mode
find /source | sudo cpio -pdm /destination
```

File Filtering and Selection

1. Pattern Matching

```
# Extract specific file patterns
cpio -i "*.txt" "*.conf" < archive.cpio

# Extract files in specific directory
cpio -i "etc/*" < archive.cpio

# Extract with shell globbing
cpio -i "*backup*" < archive.cpio
```

2. Exclude Patterns

```
# Create archive excluding certain files
find . -type f ! -name "*.tmp" ! -name "*.log" | cpio -oH newc
> clean_archive.cpio

# Use grep to filter
find . -type f | grep -v -E '\.(tmp|log|cache)$' | cpio -oH
newc > filtered.cpio
```

Network Operations

1. Remote Backup

```
# Send archive over network
find /home | cpio -oH newc | ssh user@remote "cat >
backup.cpio"

# Compressed network backup
find /data | cpio -oH newc | gzip | ssh user@remote "cat >
data_backup.cpio.gz"

# Receive archive from network
ssh user@remote "cat backup.cpio" | cpio -id
```

2. Tape Operations

```
# Write to tape device
find /home | cpio -oH newc > /dev/st0

# Read from tape
cpio -itv < /dev/st0   # List contents
cpio -id < /dev/st0    # Extract

# Multi-volume archives
find /large_dataset | cpio -oH newc --split-at=700M > /dev/st0
```

Archive Management

1. Listing Archive Contents

```
# List all files in archive
cpio -tv < archive.cpio

# List with detailed information
cpio -itv < archive.cpio

# Count files in archive
cpio -it < archive.cpio | wc -l

# Search for specific files
cpio -it < archive.cpio | grep "filename"
```

2. Verifying Archives

```
# Test archive integrity
cpio -it < archive.cpio > /dev/null

# Compare with filesystem
find . | cpio -oH newc | cpio -it | sort > archive_list.txt
find . | sort > filesystem_list.txt
diff archive_list.txt filesystem_list.txt
```

3. Updating Archives

```
# Append to existing archive (not directly supported)
# Workaround: extract, add files, recreate
mkdir /tmp/archive_work
cd /tmp/archive_work
cpio -id < /path/to/archive.cpio
# Add new files
find . | cpio -oH newc > /path/to/new_archive.cpio
```

Performance Optimization

1. Block Size Tuning

```
# Use larger block size for better performance
cpio -oB > archive.cpio            # 5120 bytes
cpio -o --block-size=32768 > archive.cpio  # 32KB blocks

# For tape drives
cpio -o --block-size=65536 > /dev/st0  # 64KB blocks
```

2. Compression Strategies

```
# Different compression methods
find . | cpio -oH newc | gzip -1 > fast_compress.cpio.gz    #
Fast
find . | cpio -oH newc | gzip -9 > best_compress.cpio.gz    #
Best ratio
find . | cpio -oH newc | lz4 > lz4_compress.cpio.lz4        #
Very fast
find . | cpio -oH newc | xz -9 > xz_compress.cpio.xz        #
Best ratio
```

3. Parallel Processing

```
# Use parallel compression
find . | cpio -oH newc | pigz > parallel_compressed.cpio.gz

# Parallel decompression
pigz -dc archive.cpio.gz | cpio -id
```

Integration with Other Tools

1. With find

```
# Complex find expressions
find /var/log -name "*.log" -size +10M -mtime +30 | cpio -oH
newc > old_large_logs.cpio

# Execute commands during find
find . -name "*.txt" -exec grep -l "important" {} \; | cpio -
oH newc > important_texts.cpio
```

2. With rsync

```
# Create incremental backups
rsync -av --link-dest=/backup/previous /source/
/backup/current/
find /backup/current -type f | cpio -oH newc >
incremental.cpio
```

3. With tar compatibility

```
# Convert tar to cpio
tar -cf - files | cpio -oH tar > archive.tar

# Convert cpio to tar
cpio -it < archive.cpio | tar -cf archive.tar -T -
```

Troubleshooting

1. Common Errors

```
# "Premature end of file" error
# Check if archive is complete or corrupted
file archive.cpio

# Permission denied errors
# Use sudo for system files
sudo cpio -id < archive.cpio

# "Cannot create directory" errors
# Use -d option
cpio -id < archive.cpio
```

2. Debugging

```
# Verbose mode for debugging
cpio -idv < archive.cpio

# Check archive format
file archive.cpio
hexdump -C archive.cpio | head

# Test archive before extraction
cpio -it < archive.cpio > /dev/null
echo $?  # Should return 0 for success
```

3. Recovery

```
# Partial archive recovery
dd if=damaged_archive.cpio of=partial.cpio bs=512 count=1000
cpio -id < partial.cpio

# Skip damaged portions
cpio -id --only-verify-crc < archive.cpio
```

Scripting and Automation

1. Backup Scripts

```bash
#!/bin/bash
# Automated backup script
BACKUP_DIR="/backup/$(date +%Y%m%d)"
SOURCE="/home /etc /var/log"

mkdir -p "$BACKUP_DIR"
for dir in $SOURCE; do
    find "$dir" -type f | cpio -oH newc | gzip >
"$BACKUP_DIR/$(basename $dir).cpio.gz"
done
```

2. Restoration Scripts

```bash
#!/bin/bash
# Automated restoration script
ARCHIVE="$1"
DEST="${2:-/restore}"

if [ ! -f "$ARCHIVE" ]; then
    echo "Archive not found: $ARCHIVE"
    exit 1
fi

mkdir -p "$DEST"
cd "$DEST"

if [[ "$ARCHIVE" =~ \.gz$ ]]; then
    zcat "$ARCHIVE" | cpio -idm
else
    cpio -idm < "$ARCHIVE"
fi
```

Best Practices

730

1. Archive Naming

```
# Use descriptive names with dates
backup_$(hostname)_$(date +%Y%m%d_%H%M%S).cpio.gz

# Include source information
home_backup_$(date +%Y%m%d).cpio
etc_configs_$(date +%Y%m%d).cpio
```

2. Verification

```
# Always verify critical archives
find /important | cpio -oH newc > critical.cpio
cpio -it < critical.cpio | wc -l
find /important | wc -l
```

3. Documentation

```
# Document archive contents
cpio -itv < archive.cpio > archive_contents.txt
echo "Created: $(date)" >> archive_info.txt
echo "Source: /path/to/source" >> archive_info.txt
```

Important Notes

731

- **Choose appropriate format**: Use `newc` format for modern systems
- **Preserve permissions**: Use `-m` and run as appropriate user
- **Test archives**: Always verify archive integrity
- **Use compression**: Combine with gzip/bzip2/xz for space efficiency
- **Handle special files**: Be careful with device files and symlinks
- **Network security**: Use secure channels for remote operations
- **Backup strategy**: Regular backups with verification

The `cpio` command is powerful for creating flexible archives and system backups, especially when combined with find and compression tools.

For more details, check the manual: `man cpio`

The `lsb_release` command

The `lsb_release` command displays information about the Linux Standard Base (LSB) and distribution-specific information. It provides details about the Linux distribution version, codename, and other identifying information.

Syntax

733

```
lsb_release [options]
```

Key Features

734

- **Distribution Information**: Name, version, codename
- **LSB Compliance**: Shows LSB version support
- **Standard Format**: Consistent output across distributions
- **Scripting Friendly**: Easy to parse output

Basic Usage

Show All Information

```
# Display all available information
lsb_release -a

# Example output:
# Distributor ID: Ubuntu
# Description:    Ubuntu 22.04.1 LTS
# Release:        22.04
# Codename:       jammy
```

Individual Information

```
# Distribution ID only
lsb_release -i

# Release version only
lsb_release -r

# Codename only
lsb_release -c

# Description only
lsb_release -d
```

Common Options

Basic Options

```
# -a: Show all information
lsb_release -a

# -i: Distributor ID
lsb_release -i

# -d: Description
lsb_release -d

# -r: Release number
lsb_release -r

# -c: Codename
lsb_release -c
```

Output Format Options

```
# -s: Short format (no field names)
lsb_release -a -s

# Example output:
# Ubuntu
# Ubuntu 22.04.1 LTS
# 22.04
# jammy

# Individual fields in short format
lsb_release -i -s  # Output: Ubuntu
lsb_release -r -s  # Output: 22.04
lsb_release -c -s  # Output: jammy
```

Practical Examples

737

1. System Identification

```
# Get distribution name
DISTRO=$(lsb_release -i -s)
echo "Running on: $DISTRO"

# Get version
VERSION=$(lsb_release -r -s)
echo "Version: $VERSION"

# Get codename
CODENAME=$(lsb_release -c -s)
echo "Codename: $CODENAME"
```

2. Conditional Scripting

```bash
#!/bin/bash
# Script that behaves differently based on distribution

DISTRO=$(lsb_release -i -s)
VERSION=$(lsb_release -r -s)

case $DISTRO in
    "Ubuntu")
        echo "Ubuntu detected, version $VERSION"
        if [[ "$VERSION" == "22.04" ]]; then
            echo "Running on Ubuntu 22.04 LTS"
        fi
        ;;
    "Debian")
        echo "Debian detected, version $VERSION"
        ;;
    "CentOS"|"RedHatEnterprise")
        echo "Red Hat based system detected"
        ;;
    *)
        echo "Unknown distribution: $DISTRO"
        ;;
esac
```

3. Package Management Scripts

```bash
#!/bin/bash
# Install packages based on distribution

install_package() {
    local package=$1
    local distro=$(lsb_release -i -s)

    case $distro in
        "Ubuntu"|"Debian")
            sudo apt-get install -y "$package"
            ;;
        "CentOS"|"RedHatEnterprise")
            sudo yum install -y "$package"
            ;;
        "Fedora")
            sudo dnf install -y "$package"
            ;;
        *)
            echo "Unsupported distribution: $distro"
            return 1
            ;;
    esac
}

install_package "curl"
```

Distribution-Specific Examples

1. Ubuntu/Debian Systems

```
# Check Ubuntu version
lsb_release -a
# Distributor ID: Ubuntu
# Description:    Ubuntu 22.04.1 LTS
# Release:        22.04
# Codename:       jammy

# Check if it's LTS version
if lsb_release -d -s | grep -q "LTS"; then
    echo "This is an LTS release"
fi
```

2. CentOS/RHEL Systems

```
# CentOS example
lsb_release -a
# Distributor ID: CentOS
# Description:    CentOS Linux release 8.4.2105
# Release:        8.4.2105
# Codename:       n/a

# Check major version
MAJOR_VERSION=$(lsb_release -r -s | cut -d. -f1)
echo "Major version: $MAJOR_VERSION"
```

3. Fedora Systems

```
# Fedora example
lsb_release -a
# Distributor ID: Fedora
# Description:    Fedora release 36 (Thirty Six)
# Release:        36
# Codename:       ThirtySix
```

Alternative Information Sources

1. When lsb_release is not available

```bash
# Check if lsb_release exists
if command -v lsb_release >/dev/null 2>&1; then
    lsb_release -a
else
    echo "lsb_release not available, using alternatives:"

    # Try /etc/os-release (systemd standard)
    if [ -f /etc/os-release ]; then
        cat /etc/os-release
    fi

    # Try distribution-specific files
    if [ -f /etc/redhat-release ]; then
        cat /etc/redhat-release
    elif [ -f /etc/debian_version ]; then
        echo "Debian $(cat /etc/debian_version)"
    elif [ -f /etc/issue ]; then
        cat /etc/issue
    fi
fi
```

2. Using /etc/os-release

```
# Modern alternative to lsb_release
cat /etc/os-release

# Example output:
# PRETTY_NAME="Ubuntu 22.04.1 LTS"
# NAME="Ubuntu"
# VERSION_ID="22.04"
# VERSION="22.04.1 LTS (Jammy Jellyfish)"
# ID=ubuntu
# ID_LIKE=debian
# HOME_URL="https://www.ubuntu.com/"
# SUPPORT_URL="https://help.ubuntu.com/"

# Parse specific values
source /etc/os-release
echo "Distribution: $NAME"
echo "Version: $VERSION_ID"
echo "Pretty Name: $PRETTY_NAME"
```

Scripting and Automation

1. System Information Script

```bash
#!/bin/bash
# Comprehensive system information

echo "=== System Information ==="
echo "Date: $(date)"
echo "Hostname: $(hostname)"
echo "Uptime: $(uptime)"
echo

echo "=== Distribution Information ==="
if command -v lsb_release >/dev/null 2>&1; then
    lsb_release -a
else
    echo "lsb_release not available"
    if [ -f /etc/os-release ]; then
        echo "Using /etc/os-release:"
        cat /etc/os-release
    fi
fi
echo

echo "=== Kernel Information ==="
uname -a
echo

echo "=== Hardware Information ==="
lscpu | head -10
echo
free -h
echo
df -h
```

2. Environment Setup Script

```bash
#!/bin/bash
# Setup development environment based on distribution

DISTRO=$(lsb_release -i -s 2>/dev/null || echo "Unknown")
VERSION=$(lsb_release -r -s 2>/dev/null || echo "Unknown")

echo "Setting up environment for $DISTRO $VERSION"

case $DISTRO in
    "Ubuntu")
        echo "Configuring for Ubuntu..."
        sudo apt update
        sudo apt install -y build-essential git curl

        if [[ "$VERSION" == "22.04" ]]; then
            echo "Ubuntu 22.04 specific setup..."
            # Add 22.04 specific configurations
        fi
        ;;
    "Debian")
        echo "Configuring for Debian..."
        sudo apt update
        sudo apt install -y build-essential git curl
        ;;
    "CentOS"|"RedHatEnterprise")
        echo "Configuring for Red Hat based system..."
        sudo yum groupinstall -y "Development Tools"
        sudo yum install -y git curl
        ;;
    *)
        echo "Unknown or unsupported distribution: $DISTRO"
        echo "Manual configuration required"
        ;;
esac
```

3. Version Comparison

```bash
#!/bin/bash
# Compare distribution versions

check_minimum_version() {
    local current_version=$(lsb_release -r -s)
    local minimum_version=$1
    local distro=$(lsb_release -i -s)

    echo "Current $distro version: $current_version"
    echo "Minimum required version: $minimum_version"

    if [ "$(printf '%s\n' "$minimum_version"
"$current_version" | sort -V | head -n1)" = "$minimum_version"
]; then
        echo "Version requirement satisfied"
        return 0
    else
        echo "Version requirement NOT satisfied"
        return 1
    fi
}

# Check if Ubuntu 20.04 or later
if [[ "$(lsb_release -i -s)" == "Ubuntu" ]]; then
    check_minimum_version "20.04"
fi
```

Integration with Configuration Management

1. Ansible Facts

```
# lsb_release information is often used in Ansible
# Example Ansible task:

# - name: Install package on Ubuntu
#   apt:
#     name: nginx
#     state: present
#   when: ansible_lsb.id == "Ubuntu"

# - name: Install package on CentOS
#   yum:
#     name: nginx
#     state: present
#   when: ansible_lsb.id == "CentOS"
```

2. Docker Integration

```bash
#!/bin/bash
# Create Dockerfile based on host distribution

HOST_DISTRO=$(lsb_release -i -s)
HOST_VERSION=$(lsb_release -r -s)

cat > Dockerfile << EOF
# Generated Dockerfile based on host: $HOST_DISTRO
$HOST_VERSION
FROM ${HOST_DISTRO,,}:$HOST_VERSION

RUN apt-get update && apt-get install -y \\
    curl \\
    wget \\
    git

# Add application-specific configurations
EOF

echo "Dockerfile generated for $HOST_DISTRO $HOST_VERSION"
```

Troubleshooting

749

1. Command Not Found

```
# Install lsb_release if missing

# Ubuntu/Debian
sudo apt-get install lsb-release

# CentOS/RHEL 7
sudo yum install redhat-lsb-core

# CentOS/RHEL 8/Fedora
sudo dnf install redhat-lsb-core

# Alternative: use built-in files
cat /etc/os-release
```

2. Inconsistent Output

```
# Some distributions may not fully populate LSB information
# Use fallback methods

get_distro_info() {
    if command -v lsb_release >/dev/null 2>&1; then
        echo "Distribution: $(lsb_release -i -s)"
        echo "Version: $(lsb_release -r -s)"
        echo "Codename: $(lsb_release -c -s)"
    else
        echo "Using alternative detection methods..."
        if [ -f /etc/os-release ]; then
            source /etc/os-release
            echo "Distribution: $NAME"
            echo "Version: $VERSION_ID"
        fi
    fi
}
```

3. Parsing Issues

```
# Handle cases where information might be incomplete
parse_distro_safe() {
    local distro=$(lsb_release -i -s 2>/dev/null)
    local version=$(lsb_release -r -s 2>/dev/null)

    if [ -z "$distro" ]; then
        distro="Unknown"
    fi

    if [ -z "$version" ]; then
        version="Unknown"
    fi

    echo "Distribution: $distro"
    echo "Version: $version"
}
```

Modern Alternatives

1. hostnamectl (systemd)

```
# Modern systemd-based alternative
hostnamectl

# Example output:
#    Static hostname: ubuntu-server
#         Icon name: computer-vm
#           Chassis: vm
#        Machine ID: 12345...
#           Boot ID: 67890...
#     Virtualization: vmware
#   Operating System: Ubuntu 22.04.1 LTS
#            Kernel: Linux 5.15.0-58-generic
#       Architecture: x86-64
```

2. /etc/os-release

```
# Standard file across modern distributions
cat /etc/os-release

# Parse specific fields
grep '^NAME=' /etc/os-release | cut -d= -f2 | tr -d '"'
grep '^VERSION_ID=' /etc/os-release | cut -d= -f2 | tr -d '"'
```

Best Practices

752

1. Error Handling

```
# Always check if command exists before using
if ! command -v lsb_release >/dev/null 2>&1; then
    echo "lsb_release not available, using fallback method"
    # Use alternative method
fi
```

2. Caching Results

```
# Cache results in scripts that call lsb_release multiple
times
DISTRO_INFO=$(lsb_release -a 2>/dev/null)
DISTRO_ID=$(echo "$DISTRO_INFO" | awk '/Distributor ID:/
{print $3}')
DISTRO_VERSION=$(echo "$DISTRO_INFO" | awk '/Release:/ {print
$2}')
```

3. Cross-Platform Compatibility

```bash
# Write scripts that work across different distributions
detect_os() {
    if [ -f /etc/os-release ]; then
        source /etc/os-release
        echo "$NAME $VERSION_ID"
    elif command -v lsb_release >/dev/null 2>&1; then
        lsb_release -d -s
    elif [ -f /etc/redhat-release ]; then
        cat /etc/redhat-release
    else
        echo "Unknown"
    fi
}
```

Important Notes

754

- **Installation Required**: lsb_release may not be installed by default
- **LSB Standard**: Follows Linux Standard Base specifications
- **Distribution Specific**: Output format may vary between distributions
- **Scripting Use**: Excellent for distribution-aware scripts
- **Modern Alternative**: Consider using /etc/os-release for newer systems
- **Fallback Methods**: Always have alternative detection methods

The `lsb_release` command is essential for creating distribution-aware scripts and system identification.

For more details, check the manual: `man lsb_release`

The `dmidecode` command

The `dmidecode` command is used to retrieve hardware information from the Desktop Management Interface (DMI) table, also known as SMBIOS (System Management BIOS). It provides detailed information about system hardware components including motherboard, CPU, RAM, BIOS, and other system components.

Syntax

756

```
dmidecode [options] [type]
```

Key Features

757

- **Hardware Information**: CPU, memory, motherboard, BIOS details
- **SMBIOS Access**: Direct access to system management information
- **Structured Output**: Well-formatted hardware inventory
- **System Identification**: Serial numbers, model information
- **No Root Required**: Basic information accessible to all users

Basic Usage

758

Show All Information

```
# Display all available hardware information
sudo dmidecode

# Show specific information types
sudo dmidecode -t system
sudo dmidecode -t processor
sudo dmidecode -t memory
```

Common Information Types

```
# BIOS information
sudo dmidecode -t bios

# System information
sudo dmidecode -t system

# Base board (motherboard) information
sudo dmidecode -t baseboard

# Chassis information
sudo dmidecode -t chassis

# Processor information
sudo dmidecode -t processor

# Memory information
sudo dmidecode -t memory
```

Information Types

759

Numeric Type Codes

```
# Type 0: BIOS Information
sudo dmidecode -t 0

# Type 1: System Information
sudo dmidecode -t 1

# Type 2: Base Board Information
sudo dmidecode -t 2

# Type 3: Chassis Information
sudo dmidecode -t 3

# Type 4: Processor Information
sudo dmidecode -t 4

# Type 16: Physical Memory Array
sudo dmidecode -t 16

# Type 17: Memory Device
sudo dmidecode -t 17
```

String Type Names

```
# BIOS information
sudo dmidecode -t bios

# System information (manufacturer, model, serial)
sudo dmidecode -t system

# Motherboard information
sudo dmidecode -t baseboard

# Case/chassis information
sudo dmidecode -t chassis

# CPU information
sudo dmidecode -t processor

# Memory information
sudo dmidecode -t memory

# Slot information
sudo dmidecode -t slot

# Cache information
sudo dmidecode -t cache
```

Practical Examples

1. System Identification

```
# Get system manufacturer and model
sudo dmidecode -s system-manufacturer
sudo dmidecode -s system-product-name
sudo dmidecode -s system-serial-number

# Example output:
# Dell Inc.
# OptiPlex 7090
# 1234567

# Complete system information
sudo dmidecode -t system
```

2. Hardware Inventory

```
# CPU information
echo "=== CPU Information ==="
sudo dmidecode -t processor | grep -E
"(Family|Model|Speed|Cores|Threads)"

# Memory information
echo "=== Memory Information ==="
sudo dmidecode -t memory | grep -E
"(Size|Speed|Manufacturer|Type)"

# Motherboard information
echo "=== Motherboard Information ==="
sudo dmidecode -t baseboard | grep -E
"(Manufacturer|Product|Version)"
```

3. Memory Details 762

```
# Total memory slots and usage
echo "Memory Slots:"
sudo dmidecode -t memory | grep -E "(Locator|Size|Speed)" |
grep -v "Bank Locator"

# Maximum memory capacity
echo "Maximum Memory Capacity:"
sudo dmidecode -t 16 | grep "Maximum Capacity"

# Memory module details
echo "Installed Memory Modules:"
sudo dmidecode -t 17 | grep -E "(Size|Speed|Manufacturer|Part
Number)" | grep -v "No Module"
```

Specific Hardware Information

1. BIOS Information

```
# BIOS details
sudo dmidecode -t bios

# Specific BIOS information
sudo dmidecode -s bios-vendor
sudo dmidecode -s bios-version
sudo dmidecode -s bios-release-date

# Example output:
# American Megatrends Inc.
# 2.15.1237
# 03/15/2023
```

2. CPU Information

```
# Processor details
sudo dmidecode -t processor

# Specific CPU information
echo "CPU Family: $(sudo dmidecode -s processor-family)"
echo "CPU Manufacturer: $(sudo dmidecode -s processor-
manufacturer)"
echo "CPU Version: $(sudo dmidecode -s processor-version)"

# CPU specifications
sudo dmidecode -t processor | grep -E "(Version|Family|Max
Speed|Current Speed|Core Count|Thread Count)"
```

3. Memory Information

```
# Memory array information
sudo dmidecode -t 16

# Individual memory modules
sudo dmidecode -t 17

# Memory summary
echo "Total Memory Slots:"
sudo dmidecode -t 17 | grep "Locator:" | wc -l

echo "Occupied Memory Slots:"
sudo dmidecode -t 17 | grep "Size:" | grep -v "No Module" | wc
-l

echo "Total Installed Memory:"
sudo dmidecode -t 17 | grep "Size:" | grep -v "No Module" |
awk '{sum+=$2} END {print sum " MB"}'
```

4. Motherboard Information

```
# Motherboard details
sudo dmidecode -t baseboard

# Specific motherboard information
echo "Motherboard Manufacturer: $(sudo dmidecode -s baseboard-
manufacturer)"
echo "Motherboard Model: $(sudo dmidecode -s baseboard-
product-name)"
echo "Motherboard Version: $(sudo dmidecode -s baseboard-
version)"
echo "Motherboard Serial: $(sudo dmidecode -s baseboard-
serial-number)"
```

Advanced Usage

1. Parsing and Filtering

```
# Extract specific information using grep and awk
get_memory_info() {
    sudo dmidecode -t 17 | awk '
    /Memory Device/,/^$/ {
        if (/Size:/) size=$2" "$3
        if (/Speed:/) speed=$2" "$3
        if (/Manufacturer:/) manufacturer=$2
        if (/Part Number:/) part=$3
        if (/Locator:/ && !/Bank/) {
            locator=$2
            if (size != "No Module") {
                print locator": "size" @ "speed"
("manufacturer" "part")"
            }
        }
    }'
}

get_memory_info
```

2. System Asset Information

```
# Create system asset report
create_asset_report() {
    echo "=== System Asset Report ==="
    echo "Generated: $(date)"
    echo

    echo "System Information:"
    echo "  Manufacturer: $(sudo dmidecode -s system-
manufacturer)"
    echo "  Model: $(sudo dmidecode -s system-product-name)"
    echo "  Serial Number: $(sudo dmidecode -s system-serial-
number)"
    echo "  UUID: $(sudo dmidecode -s system-uuid)"
    echo

    echo "BIOS Information:"
    echo "  Vendor: $(sudo dmidecode -s bios-vendor)"
    echo "  Version: $(sudo dmidecode -s bios-version)"
    echo "  Date: $(sudo dmidecode -s bios-release-date)"
    echo

    echo "Motherboard Information:"
    echo "  Manufacturer: $(sudo dmidecode -s baseboard-
manufacturer)"
    echo "  Model: $(sudo dmidecode -s baseboard-product-
name)"
    echo "  Serial: $(sudo dmidecode -s baseboard-serial-
number)"
    echo

    echo "Chassis Information:"
    echo "  Type: $(sudo dmidecode -s chassis-type)"
    echo "  Manufacturer: $(sudo dmidecode -s chassis-
manufacturer)"
    echo "  Serial: $(sudo dmidecode -s chassis-serial-
number)"
}

create_asset_report > system_asset_report.txt
```

3. Hardware Validation

```
# Validate hardware configuration
validate_hardware() {
    echo "=== Hardware Validation ==="

    # Check if virtualization is supported
    if sudo dmidecode -t processor | grep -q "VMX\|SVM"; then
        echo "✓ Virtualization supported"
    else
        echo "✗ Virtualization not supported"
    fi

    # Check memory configuration
    local total_slots=$(sudo dmidecode -t 17 | grep "Locator:"
| wc -l)
    local used_slots=$(sudo dmidecode -t 17 | grep "Size:" |
grep -v "No Module" | wc -l)
    echo "Memory: $used_slots/$total_slots slots used"

    # Check for ECC memory
    if sudo dmidecode -t 17 | grep -q "Error Correction
Type.*ECC"; then
        echo "✓ ECC memory detected"
    else
        echo "□ No ECC memory detected"
    fi

    # Check system age (approximate)
    local bios_date=$(sudo dmidecode -s bios-release-date)
    local year=$(echo $bios_date | awk -F'/' '{print $3}')
    local current_year=$(date +%Y)
    local age=$((current_year - year))
    echo "Approximate system age: $age years"
}

validate_hardware
```

String Options

Available String Options

```
# System strings
sudo dmidecode -s system-manufacturer
sudo dmidecode -s system-product-name
sudo dmidecode -s system-version
sudo dmidecode -s system-serial-number
sudo dmidecode -s system-uuid

# BIOS strings
sudo dmidecode -s bios-vendor
sudo dmidecode -s bios-version
sudo dmidecode -s bios-release-date

# Baseboard strings
sudo dmidecode -s baseboard-manufacturer
sudo dmidecode -s baseboard-product-name
sudo dmidecode -s baseboard-version
sudo dmidecode -s baseboard-serial-number

# Chassis strings
sudo dmidecode -s chassis-manufacturer
sudo dmidecode -s chassis-type
sudo dmidecode -s chassis-version
sudo dmidecode -s chassis-serial-number

# Processor strings
sudo dmidecode -s processor-family
sudo dmidecode -s processor-manufacturer
sudo dmidecode -s processor-version
sudo dmidecode -s processor-frequency
```

Output Options

Formatting Options

```
# Quiet output (suppress headers)
sudo dmidecode -q -t system

# No piping warning
sudo dmidecode --no-sysfs -t memory

# Dump raw DMI data
sudo dmidecode --dump-bin dmi.bin

# Read from binary dump
dmidecode --from-dump dmi.bin
```

Scripting and Automation

1. Hardware Monitoring Script

```bash
#!/bin/bash
# Monitor hardware changes

HARDWARE_LOG="/var/log/hardware_changes.log"
CURRENT_STATE="/tmp/current_hardware.txt"
PREVIOUS_STATE="/var/lib/hardware_baseline.txt"

# Generate current hardware state
{
    echo "=== Hardware State $(date) ==="
    sudo dmidecode -s system-serial-number
    sudo dmidecode -t memory | grep -E "(Size|Speed)" | grep -
v "No Module"
    sudo dmidecode -t processor | grep -E
"(Version|Speed|Cores)"
} > "$CURRENT_STATE"

# Compare with previous state
if [ -f "$PREVIOUS_STATE" ]; then
    if ! diff -q "$PREVIOUS_STATE" "$CURRENT_STATE"
>/dev/null; then
        echo "$(date): Hardware configuration changed" >>
"$HARDWARE_LOG"
        diff "$PREVIOUS_STATE" "$CURRENT_STATE" >>
"$HARDWARE_LOG"
    fi
fi

# Update baseline
cp "$CURRENT_STATE" "$PREVIOUS_STATE"
```

2. Inventory Collection

```bash
#!/bin/bash
# Collect hardware inventory for multiple systems

collect_inventory() {
    local hostname=$(hostname)
    local output_file="inventory_${hostname}_$(date
+%Y%m%d).txt"

    {
        echo "Hardware Inventory for $hostname"
        echo "Generated: $(date)"
        echo "================================="
        echo

        echo "System Information:"
        sudo dmidecode -t system | grep -E
"(Manufacturer|Product|Serial|UUID)"
        echo

        echo "BIOS Information:"
        sudo dmidecode -t bios | grep -E
"(Vendor|Version|Release Date)"
        echo

        echo "Memory Configuration:"
        sudo dmidecode -t memory | grep -E "(Maximum
Capacity|Number Of Devices)"
        sudo dmidecode -t 17 | grep -E
"(Locator|Size|Speed|Manufacturer)" | grep -v "Bank Locator"
        echo

        echo "Processor Information:"
        sudo dmidecode -t processor | grep -E
"(Family|Version|Speed|Core Count|Thread Count)"
        echo

        echo "Motherboard Information:"
        sudo dmidecode -t baseboard | grep -E
"(Manufacturer|Product|Version|Serial)"

    } > "$output_file"
```

```
    echo "Inventory saved to: $output_file"
}

collect_inventory
```

3. License Management

```bash
#!/bin/bash
# Extract information for software licensing

get_license_info() {
    echo "System Identification for Licensing:"
    echo "UUID: $(sudo dmidecode -s system-uuid)"
    echo "Serial: $(sudo dmidecode -s system-serial-number)"
    echo "Manufacturer: $(sudo dmidecode -s system-manufacturer)"
    echo "Model: $(sudo dmidecode -s system-product-name)"

    # Generate unique system fingerprint
    local fingerprint=$(sudo dmidecode -s system-uuid)$(sudo dmidecode -s baseboard-serial-number)
    echo "System Fingerprint: $(echo -n "$fingerprint" | sha256sum | cut -d' ' -f1)"
}

get_license_info
```

Troubleshooting

1. Permission Issues

```
# dmidecode requires root privileges for full access
# Some information may be available without root

# Check what's available without root
dmidecode 2>/dev/null || echo "Root access required for
complete information"

# Use sudo for full access
sudo dmidecode -t system
```

2. Virtual Machine Considerations

```
# In VMs, some information may be virtualized or missing
check_virtualization() {
    local system_product=$(sudo dmidecode -s system-product-
name)

    case "$system_product" in
        *"VMware"*|*"Virtual Machine"*|*"VirtualBox"*)
            echo "Running in virtual machine: $system_product"
            echo "Some hardware information may be
virtualized"
            ;;
        *)
            echo "Physical machine detected"
            ;;
    esac
}

check_virtualization
```

3. Missing Information

```bash
# Some fields may show "Not Specified" or be empty
# Handle missing data gracefully

get_safe_value() {
    local value=$(sudo dmidecode -s "$1" 2>/dev/null)
    if [ -z "$value" ] || [ "$value" = "Not Specified" ]; then
        echo "Unknown"
    else
        echo "$value"
    fi
}

echo "Manufacturer: $(get_safe_value system-manufacturer)"
echo "Model: $(get_safe_value system-product-name)"
```

Integration with Other Tools

1. Combining with lscpu

```
# Comprehensive CPU information
echo "=== CPU Information ==="
echo "DMI Information:"
sudo dmidecode -t processor | grep -E
"(Family|Version|Speed|Cores)"
echo
echo "Kernel Information:"
lscpu
```

2. Memory Cross-Reference

```
# Compare dmidecode with /proc/meminfo
echo "DMI Memory Information:"
sudo dmidecode -t 16 | grep "Maximum Capacity"
sudo dmidecode -t 17 | grep "Size:" | grep -v "No Module"
echo
echo "Kernel Memory Information:"
cat /proc/meminfo | grep -E "(MemTotal|MemAvailable)"
```

3. System Monitoring Integration

```
# Add to monitoring systems
create_monitoring_metrics() {
    local uuid=$(sudo dmidecode -s system-uuid)
    local serial=$(sudo dmidecode -s system-serial-number)
    local manufacturer=$(sudo dmidecode -s system-
manufacturer)

    # Output in monitoring format (e.g., Prometheus)
    echo
"system_info{uuid=\"$uuid\",serial=\"$serial\",manufacturer=\"
$manufacturer\"} 1"
}
```

Best Practices

1. Caching Results

```
# Cache dmidecode output for scripts that use it multiple
times
DMI_CACHE="/tmp/dmidecode_cache.txt"

get_cached_dmidecode() {
    if [ ! -f "$DMI_CACHE" ] || [ "$(find "$DMI_CACHE" -mmin
+60)" ]; then
        sudo dmidecode > "$DMI_CACHE"
    fi
    cat "$DMI_CACHE"
}
```

2. Error Handling

```
# Always check if dmidecode is available and handle errors
run_dmidecode() {
    if ! command -v dmidecode >/dev/null 2>&1; then
        echo "dmidecode not available"
        return 1
    fi

    if ! sudo dmidecode "$@" 2>/dev/null; then
        echo "Failed to read DMI information"
        return 1
    fi
}
```

3. Security Considerations

```
# Be careful with sensitive information in logs
sanitize_output() {
    sudo dmidecode "$@" | sed 's/Serial Number:.*/Serial
Number: [REDACTED]/'
}
```

Important Notes

- **Root Access**: Full functionality requires root privileges
- **Hardware Dependent**: Output depends on BIOS/UEFI implementation
- **Virtual Machines**: Information may be virtualized or limited
- **Manufacturer Specific**: Some fields may be manufacturer-specific
- **Version Differences**: Output format may vary between dmidecode versions
- **Security**: Be careful with sensitive hardware information in logs

The `dmidecode` command is essential for hardware inventory, system identification, and troubleshooting hardware-related issues.

For more details, check the manual: `man dmidecode`

The `apropos` command

The `apropos` command searches the manual page names and descriptions for keywords. It's essentially equivalent to `man -k` and helps users find relevant commands and documentation when they know what they want to do but don't know the exact command name.

Syntax

781

```
apropos [options] keyword...
```

Key Features

782

- **Keyword Search**: Searches manual page names and descriptions
- **Regular Expression Support**: Allows pattern matching
- **Multiple Keywords**: Search for multiple terms
- **Section Filtering**: Limit search to specific manual sections
- **Exact Matching**: Find exact word matches

Basic Usage

Simple Keyword Search

```
# Search for commands related to "copy"
apropos copy

# Search for commands related to "network"
apropos network

# Search for file-related commands
apropos file

# Search for text processing commands
apropos text
```

Multiple Keywords

```
# Search for commands related to both "file" and "system"
apropos file system

# Search for networking and configuration
apropos network config

# Search for user and management
apropos user management
```

Common Options

Basic Options

```
# -a: AND search (all keywords must match)
apropos -a file system

# -e: Exact match
apropos -e copy

# -r: Use regular expressions
apropos -r "^net"

# -w: Match whole words only
apropos -w net
```

Advanced Options

```
# -s: Search specific manual sections
apropos -s 1 copy          # User commands only
apropos -s 8 network       # System administration commands

# -l: Long output format
apropos -l file

# -M: Specify manual path
apropos -M /usr/share/man file
```

Practical Examples

1. Finding Commands by Function

```
# Find compression commands
apropos compress

# Find archive commands
apropos archive

# Find text editors
apropos editor

# Find file system commands
apropos filesystem

# Find process management commands
apropos process
```

2. Network-Related Commands

```
# General network commands
apropos network

# Network configuration
apropos -a network config

# Network troubleshooting
apropos -a network trouble

# Firewall commands
apropos firewall

# DNS-related commands
apropos dns
```

3. System Administration

```
# User management
apropos user

# Service management
apropos service

# System monitoring
apropos -a system monitor

# Log management
apropos log

# Package management
apropos package
```

4. Development and Programming

```
# Compiler commands
apropos compiler

# Debugging tools
apropos debug

# Version control
apropos version

# Build tools
apropos build

# Library commands
apropos library
```

Using Regular Expressions

1. Pattern Matching

```
# Find commands starting with "net"
apropos -r "^net"

# Find commands ending with "fs"
apropos -r "fs$"

# Find commands containing "config"
apropos -r ".*config.*"

# Find commands with specific patterns
apropos -r "[0-9]+"
```

2. Complex Patterns

```
# Multiple patterns
apropos -r "(copy|move|transfer)"

# Case-insensitive search
apropos -r -i "FILE"

# Word boundaries
apropos -r "\bnet\b"
```

Section-Specific Searches

Manual Sections

```
# Section 1: User commands
apropos -s 1 copy

# Section 2: System calls
apropos -s 2 file

# Section 3: Library functions
apropos -s 3 string

# Section 5: File formats
apropos -s 5 config

# Section 8: System administration
apropos -s 8 mount
```

Understanding Sections

```
# List all sections
man man | grep -A 10 "MANUAL SECTIONS"

# Common sections:
# 1 - User commands
# 2 - System calls
# 3 - Library functions
# 4 - Device files
# 5 - File formats and conventions
# 6 - Games
# 7 - Miscellaneous
# 8 - System administration commands
```

Advanced Usage

1. Combining with Other Commands

```
# Search and count results
apropos network | wc -l

# Search and sort
apropos file | sort

# Search and filter
apropos copy | grep -v "manual"

# Search and format
apropos -l network | column -t
```

2. Scripting Applications

```bash
#!/bin/bash
# Find commands for specific tasks

find_commands() {
    local task="$1"
    echo "Commands related to '$task':"
    apropos "$task" | head -10
    echo
}

# Usage examples
find_commands "backup"
find_commands "monitor"
find_commands "security"
```

3. Learning Tool

790

```bash
# Daily command discovery
daily_discovery() {
    local keywords=("network" "file" "text" "system" "process")
    local keyword=${keywords[$RANDOM % ${#keywords[@]}]}

    echo "Today's command discovery - Topic: $keyword"
    apropos "$keyword" | shuf | head -5
}

daily_discovery
```

Troubleshooting Common Issues

1. No Results Found

```
# Update manual database if no results
sudo mandb

# Check if manual pages are installed
ls /usr/share/man/man1/ | head

# Try different keywords
apropos copy
apropos duplicate
apropos clone
```

2. Database Issues

```
# Rebuild manual database
sudo mandb -c

# Force database rebuild
sudo mandb -f

# Check database status
mandb --version
```

3. Permission Issues

```
# Check manual path permissions
ls -la /usr/share/man/

# Check database location
find /var -name "*man*" -type d 2>/dev/null

# Run with specific path
apropos -M /usr/local/man keyword
```

Useful Search Patterns

1. Common Tasks

```
# File operations
apropos -a file copy
apropos -a file move
apropos -a file remove
apropos -a file search

# System information
apropos -a system info
apropos -a hardware info
apropos -a disk usage

# Network operations
apropos -a network interface
apropos -a network test
apropos -a network config
```

2. By Software Category

```
# Text processing
apropos -r "(awk|sed|grep|cut|sort)"

# Compression tools
apropos -r "(gzip|tar|zip|compress)"

# System monitoring
apropos -r "(top|ps|iostat|netstat)"

# File systems
apropos -r "(mount|umount|fsck|mkfs)"
```

3. Administration Tasks

```
# User management
apropos -a user add
apropos -a user delete
apropos -a user modify

# Service management
apropos -a service start
apropos -a service stop
apropos -a service status

# Package management
apropos -a package install
apropos -a package remove
apropos -a package update
```

Integration with Learning

1. Command Discovery Script

```bash
#!/bin/bash
# Interactive command discovery

discover_commands() {
    echo "What do you want to do? (e.g., 'copy files',
'monitor system')"
    read -r task

    echo "Searching for commands related to: $task"
    echo "===================================="

    apropos "$task" | while read -r line; do
        cmd=$(echo "$line" | awk '{print $1}')
        desc=$(echo "$line" | cut -d' ' -f2-)

        echo "Command: $cmd"
        echo "Description: $desc"
        echo "Try: man $cmd"
        echo "---"
    done
}

discover_commands
```

2. Command Recommendation

```bash
#!/bin/bash
# Recommend commands based on keywords

recommend_command() {
    local keyword="$1"
    echo "For '$keyword', you might want to try:"

    apropos "$keyword" | head -5 | while read -r line; do
        cmd=$(echo "$line" | awk '{print $1}')
        echo "  • $cmd - $(man -f $cmd 2>/dev/null | cut -d' '
-f2- || echo 'No description')"
    done
}

# Examples
recommend_command "backup"
recommend_command "monitor"
recommend_command "compress"
```

Comparison with Similar Commands

1. apropos vs man -k

```
# These are equivalent
apropos network
man -k network

# Both search manual page descriptions
```

2. apropos vs whatis

```
# apropos: searches descriptions (broader)
apropos copy

# whatis: exact command name match (narrower)
whatis cp
```

3. apropos vs which/whereis

```
# apropos: finds commands by description
apropos file

# which: finds command location
which cp

# whereis: finds command, source, manual locations
whereis cp
```

Configuration and Customization

1. Manual Path Configuration

```
# Check current manual paths
manpath

# Add custom manual path
export MANPATH="/usr/local/man:$MANPATH"

# Make permanent in shell profile
echo 'export MANPATH="/usr/local/man:$MANPATH"' >> ~/.bashrc
```

2. Database Configuration

```
# Manual database location
echo $MANDB

# Update configuration
sudo nano /etc/manpath.config

# Force database update
sudo mandb -f
```

Advanced Scripting Examples

1. Command Explorer

```bash
#!/bin/bash
# Interactive command explorer

while true; do
    echo -n "Enter search term (or 'quit' to exit): "
    read -r term

    if [ "$term" = "quit" ]; then
        break
    fi

    results=$(apropos "$term" 2>/dev/null)

    if [ -z "$results" ]; then
        echo "No commands found for '$term'"
        echo "Try: sudo mandb  # to update manual database"
    else
        echo "Commands related to '$term':"
        echo "$results" | nl
        echo
        echo -n "Enter number to view manual (or press Enter
to continue): "
        read -r choice

        if [[ "$choice" =~ ^[0-9]+$ ]]; then
            cmd=$(echo "$results" | sed -n "${choice}p" | awk
'{print $1}')
            if [ -n "$cmd" ]; then
                man "$cmd"
            fi
        fi
    fi
    echo
done
```

2. Command Category Browser

```bash
#!/bin/bash
# Browse commands by category

categories=(
    "file:File operations"
    "network:Network commands"
    "system:System administration"
    "text:Text processing"
    "process:Process management"
    "security:Security tools"
    "backup:Backup and archive"
    "monitor:System monitoring"
)

echo "Command Categories:"
for i in "${!categories[@]}"; do
    desc=$(echo "${categories[i]}" | cut -d: -f2)
    echo "$((i+1)). $desc"
done

echo -n "Select category (1-${#categories[@]}): "
read -r choice

if [[ "$choice" =~ ^[0-9]+$ ]] && [ "$choice" -ge 1 ] && [ \
"$choice" -le "${#categories[@]}" ]; then
    keyword=$(echo "${categories[$((choice-1))]}" | cut -d: -\
f1)
    desc=$(echo "${categories[$((choice-1))]}" | cut -d: -f2)

    echo "Commands for $desc:"
    apropos "$keyword" | head -10
fi
```

Best Practices

1. Effective Searching

```
# Start with broad terms, then narrow down
apropos file
apropos -a file copy
apropos -a file copy system

# Use synonyms if no results
apropos copy || apropos duplicate || apropos clone
```

2. Regular Database Updates

```
# Add to crontab for regular updates
# 0 3 * * 0 /usr/bin/mandb -q

# Or create update script
#!/bin/bash
echo "Updating manual database..."
sudo mandb -q
echo "Manual database updated."
```

3. Learning Integration

```
# Create learning aliases
alias learn='apropos'
alias find-cmd='apropos'
alias what-cmd='apropos'

# Create help function
help-me() {
    echo "What do you want to do?"
    echo "Example: help-me copy files"
    apropos "$*"
}
```

Important Notes

- **Database Dependency**: Requires updated manual database (`mandb`)
- **Keyword Quality**: Results depend on quality of search terms
- **Manual Completeness**: Only finds documented commands
- **Regular Expressions**: Use `-r` for pattern matching
- **Section Awareness**: Use `-s` for section-specific searches
- **Case Sensitivity**: Generally case-insensitive by default

The `apropos` command is invaluable for discovering commands and learning about system capabilities when you know what you want to accomplish but not the specific command to use.

For more details, check the manual: `man apropos`

The `dmesg` command

The `dmesg` command displays messages from the kernel ring buffer. It shows boot messages, hardware detection, driver loading, and system events. This is essential for troubleshooting hardware issues, driver problems, and understanding system startup processes.

Syntax

```
dmesg [options]
```

Key Features

- **Kernel Messages**: Shows kernel ring buffer contents
- **Boot Information**: Hardware detection and driver loading
- **Real-time Monitoring**: Can follow new messages
- **Filtering Options**: Filter by facility, level, or time
- **Multiple Formats**: Human-readable and raw formats

Basic Usage

Show All Messages

```
# Display all kernel messages
dmesg

# Display with human-readable timestamps
dmesg -T

# Display last 20 lines
dmesg | tail -20

# Display first 20 lines (boot messages)
dmesg | head -20
```

Follow New Messages

```
# Follow new kernel messages (like tail -f)
dmesg -w

# Follow with timestamps
dmesg -T -w

# Follow last 10 lines and continue
dmesg | tail -10 && dmesg -w
```

Message Levels and Facilities

Message Levels

```
# Emergency messages (level 0)
dmesg -l emerg

# Alert messages (level 1)
dmesg -l alert

# Critical messages (level 2)
dmesg -l crit

# Error messages (level 3)
dmesg -l err

# Warning messages (level 4)
dmesg -l warn

# Notice messages (level 5)
dmesg -l notice

# Info messages (level 6)
dmesg -l info

# Debug messages (level 7)
dmesg -l debug
```

Multiple Levels

```
# Show errors and warnings
dmesg -l err,warn

# Show critical and above (crit, alert, emerg)
dmesg -l crit+

# Show warnings and below (warn, notice, info, debug)
dmesg -l warn-
```

Facilities

```
# Kernel messages
dmesg -f kern

# User-space messages
dmesg -f user

# Mail system messages
dmesg -f mail

# System daemons
dmesg -f daemon

# Authorization messages
dmesg -f auth

# Syslog messages
dmesg -f syslog
```

Common Options

Display Options

```
# -T: Human-readable timestamps
dmesg -T

# -t: Don't show timestamps
dmesg -t

# -x: Decode facility and level to human-readable
dmesg -x

# -H: Enable human-readable output
dmesg -H

# -k: Show kernel messages only
dmesg -k
```

Buffer Options

```
# -c: Clear ring buffer after reading
sudo dmesg -c

# -C: Clear ring buffer
sudo dmesg -C

# -s: Buffer size
dmesg -s 65536

# -n: Set console log level
sudo dmesg -n 1
```

Practical Examples

1. Hardware Detection

```
# Check USB device detection
dmesg | grep -i usb

# Check network interface detection
dmesg | grep -i eth

# Check disk detection
dmesg | grep -i -E "(sda|sdb|sdc|nvme)"

# Check memory detection
dmesg | grep -i memory

# Check CPU detection
dmesg | grep -i cpu
```

2. Driver Loading

```
# Check loaded drivers
dmesg | grep -i driver

# Check specific driver (e.g., nvidia)
dmesg | grep -i nvidia

# Check wireless driver
dmesg | grep -i wifi

# Check bluetooth driver
dmesg | grep -i bluetooth

# Check sound driver
dmesg | grep -i audio
```

3. Error Debugging

```
# Show only error messages
dmesg -l err

# Show errors and warnings
dmesg -l err,warn

# Search for specific errors
dmesg | grep -i error

# Search for failed operations
dmesg | grep -i fail

# Search for timeout issues
dmesg | grep -i timeout
```

4. Boot Process Analysis

```
# Show boot messages with timestamps
dmesg -T | head -50

# Find boot completion time
dmesg | grep -i "boot"

# Check service startup
dmesg | grep -i systemd

# Check filesystem mounting
dmesg | grep -i mount

# Check network initialization
dmesg | grep -i network
```

Time-Based Filtering

Recent Messages

```
# Messages from last 10 minutes
dmesg --since="10 minutes ago"

# Messages from last hour
dmesg --since="1 hour ago"

# Messages from today
dmesg --since="today"

# Messages from specific time
dmesg --since="2023-01-01 00:00:00"
```

Time Ranges

```
# Messages between specific times
dmesg --since="2023-01-01" --until="2023-01-02"

# Messages from last boot
dmesg --since="last boot"

# Messages from specific duration
dmesg --since="30 minutes ago" --until="10 minutes ago"
```

Advanced Filtering

1. Combining Filters

```
# Errors from last hour
dmesg -l err --since="1 hour ago"

# Kernel warnings with timestamps
dmesg -T -l warn -f kern

# USB-related errors
dmesg -l err | grep -i usb

# Network errors from today
dmesg --since="today" | grep -i -E "(network|eth|wifi)"
```

2. Output Formatting

```
# JSON format
dmesg --json

# Raw format (no timestamp processing)
dmesg -r

# Colored output (if supported)
dmesg --color=always

# No pager
dmesg --nopager
```

3. Custom Formatting

```
# Extract specific information
extract_usb_devices() {
    dmesg | grep -E "usb.*: New USB device" | \
    sed -n 's/.*New USB device found, idVendor=\([^,]*\),
idProduct=\([^ ]*\).*/Vendor: \1, Product: \2/p'
}

extract_usb_devices
```

Monitoring and Alerting

1. Real-time Monitoring

```bash
# Monitor for specific events
monitor_errors() {
    dmesg -w -l err | while read line; do
        echo "$(date): $line"
        # Send alert
        echo "$line" | mail -s "Kernel Error Alert"
admin@domain.com
    done
}

# Monitor USB connections
monitor_usb() {
    dmesg -w | grep --line-buffered -i usb | while read line;
do
        echo "USB Event: $line"
    done
}
```

2. Log Analysis Scripts

```bash
#!/bin/bash
# Analyze kernel messages for issues

analyze_dmesg() {
    echo "=== Kernel Message Analysis ==="
    echo "Generated: $(date)"
    echo

    echo "Error Messages:"
    dmesg -l err | tail -10
    echo

    echo "Warning Messages:"
    dmesg -l warn | tail -10
    echo

    echo "Recent Hardware Events:"
    dmesg --since="1 hour ago" | grep -i -E
"(usb|disk|network|memory)" | tail -10
    echo

    echo "Driver Loading Issues:"
    dmesg | grep -i -E "(failed|error|timeout)" | grep -i
driver | tail -5
}

analyze_dmesg > kernel_analysis.txt
```

3. System Health Check

```bash
#!/bin/bash
# Check system health using dmesg

check_system_health() {
    local issues=0

    echo "=== System Health Check ==="

    # Check for critical errors
    critical_errors=$(dmesg -l crit,alert,emerg --since="24
```

```
hours ago" | wc -l)
    if [ $critical_errors -gt 0 ]; then
        echo "⚠️  $critical_errors critical errors found in
last 24 hours"
        ((issues++))
    else
        echo "✓ No critical errors in last 24 hours"
    fi

    # Check for hardware errors
    hw_errors=$(dmesg --since="24 hours ago" | grep -i -c -E
"(hardware error|machine check|MCE)")
    if [ $hw_errors -gt 0 ]; then
        echo "⚠️  $hw_errors hardware errors detected"
        ((issues++))
    else
        echo "✓ No hardware errors detected"
    fi

    # Check for out of memory
    oom_events=$(dmesg --since="24 hours ago" | grep -i -c
"out of memory")
    if [ $oom_events -gt 0 ]; then
        echo "⚠️  $oom_events out of memory events"
        ((issues++))
    else
        echo "✓ No out of memory events"
    fi

    # Check for filesystem errors
    fs_errors=$(dmesg --since="24 hours ago" | grep -i -c -E
"(filesystem error|ext[234] error|xfs error)")
    if [ $fs_errors -gt 0 ]; then
        echo "⚠️  $fs_errors filesystem errors"
        ((issues++))
    else
        echo "✓ No filesystem errors"
    fi

    echo
    if [ $issues -eq 0 ]; then
        echo "✓ System appears healthy!"
    else
        echo "⚠️  Found $issues types of issues - check
details above"
```

```
    fi
}

check_system_health
```

Common Use Cases

1. USB Device Troubleshooting

```bash
# Monitor USB device connections
monitor_usb_debug() {
    echo "Monitoring USB events (Ctrl+C to stop)..."
    dmesg -w | grep --line-buffered -i usb | while read line;
do
        timestamp=$(date '+%Y-%m-%d %H:%M:%S')
        echo "[$timestamp] $line"
    done
}

# Check USB device history
usb_device_history() {
    echo "USB Device Connection History:"
    dmesg | grep -i "usb.*: New USB device" | tail -10
    echo
    echo "USB Device Disconnections:"
    dmesg | grep -i "usb.*disconnect" | tail -10
}
```

2. Network Interface Issues

```
# Check network interface events
check_network_issues() {
    echo "Network Interface Events:"
    dmesg | grep -i -E "(eth[0-9]|wlan[0-9]|enp|wlp)" | tail
-10
    echo

    echo "Network Driver Issues:"
    dmesg | grep -i -E "network.*error|ethernet.*error" | tail
-5
    echo

    echo "Link Status Changes:"
    dmesg | grep -i "link" | tail -10
}
```

3. Storage Device Monitoring

```
# Check disk health and errors
check_storage_health() {
    echo "Storage Device Detection:"
    dmesg | grep -i -E "(sda|sdb|nvme)" | grep -i "sectors" |
tail -5
    echo

    echo "Storage Errors:"
    dmesg | grep -i -E "(I/O error|Medium Error|critical
medium error)" | tail -5
    echo

    echo "Filesystem Events:"
    dmesg | grep -i -E "(mounted|unmounted|remounted)" | tail
-10
}
```

Integration with Other Tools

1. Combining with journalctl

```
# Compare kernel messages
echo "=== dmesg output ==="
dmesg --since="1 hour ago" | head -5

echo "=== journalctl kernel messages ==="
journalctl -k --since="1 hour ago" | head -5
```

2. Log File Integration

```
# Save dmesg to file with rotation
save_dmesg_log() {
    local logfile="/var/log/dmesg.log"
    local timestamp=$(date '+%Y-%m-%d %H:%M:%S')

    echo "[$timestamp] === dmesg snapshot ===" >> "$logfile"
    dmesg -T >> "$logfile"

    # Rotate if file gets too large
    if [ $(stat -f%z "$logfile" 2>/dev/null || stat -c%s
"$logfile") -gt 10485760 ]; then
        mv "$logfile" "$logfile.old"
        echo "Log rotated at $timestamp" > "$logfile"
    fi
}
```

3. Monitoring System Integration

```
# Create metrics for monitoring systems
create_dmesg_metrics() {
    local errors=$(dmesg -l err --since="1 hour ago" | wc -l)
    local warnings=$(dmesg -l warn --since="1 hour ago" | wc -l)
    local timestamp=$(date +%s)

    # Output in Prometheus format
    echo "kernel_errors_total $errors $timestamp"
    echo "kernel_warnings_total $warnings $timestamp"
}
```

Troubleshooting

1. Buffer Size Issues

```
# Check current buffer size
dmesg | wc -l

# Increase buffer size (temporary)
sudo dmesg -s 1048576

# Make permanent (add to kernel parameters)
# Add to GRUB: log_buf_len=1048576
```

2. Permission Issues

```
# Check if non-root can read dmesg
dmesg >/dev/null 2>&1 && echo "Can read dmesg" || echo "Cannot read dmesg"

# Check kernel parameter
cat /proc/sys/kernel/dmesg_restrict

# Allow non-root users (temporary)
sudo sysctl kernel.dmesg_restrict=0

# Make permanent
echo "kernel.dmesg_restrict = 0" | sudo tee -a /etc/sysctl.conf
```

3. Message Filtering Issues

```
# Check available facilities and levels
dmesg --help | grep -A 20 "supported facilities"

# Test filtering
dmesg -l err | head -5
dmesg -f kern | head -5

# Check if systemd is interfering
systemctl status systemd-journald
```

Performance Considerations

1. Large Buffer Handling

```bash
# Process large buffers efficiently
process_large_dmesg() {
    # Use streaming instead of loading all into memory
    dmesg | while IFS= read -r line; do
        # Process line by line
        echo "$line" | grep -q "error" && echo "Error: $line"
    done
}
```

2. Efficient Searching

```bash
# Use specific filters instead of post-processing
dmesg -l err                 # Better than dmesg | grep -i error
dmesg --since="1 hour ago" # Better than dmesg | filtering by time
```

Best Practices

1. Regular Monitoring

```bash
# Create cron job for regular checks
# 0 */4 * * * /usr/local/bin/check_dmesg_errors.sh

# Create check script
#!/bin/bash
errors=$(dmesg -l err --since="4 hours ago" | wc -l)
if [ $errors -gt 0 ]; then
    dmesg -l err --since="4 hours ago" | mail -s "Kernel
Errors Detected" admin@domain.com
fi
```

2. Log Retention

```bash
# Save important messages
save_important_messages() {
    local date_str=$(date '+%Y%m%d')
    dmesg -l err,crit,alert,emerg >
"/var/log/kernel_errors_$date_str.log"
}
```

3. Documentation

```
# Document system events
document_system_event() {
    local event="$1"
    local logfile="/var/log/system_events.log"

    echo "$(date): $event" >> "$logfile"
    echo "=== dmesg at time of event ===" >> "$logfile"
    dmesg -T | tail -20 >> "$logfile"
    echo "=================================" >> "$logfile"
}
```

Important Notes

829

- **Root Access**: Some distributions restrict dmesg to root users
- **Buffer Size**: Ring buffer has limited size; old messages are overwritten
- **Timestamps**: Use `-T` for human-readable timestamps
- **Levels**: Understand message levels for effective filtering
- **Real-time**: Use `-w` for monitoring new messages
- **Performance**: Large buffers can impact performance
- **Security**: Be cautious about exposing kernel messages

The `dmesg` command is essential for system troubleshooting, hardware debugging, and understanding kernel behavior.

For more details, check the manual: `man dmesg`

The !! command (History Expansion)

The !! command is a bash history expansion feature that repeats the last command. It's part of a broader set of history expansion capabilities that allow you to quickly re-execute, modify, or reference previous commands. This is extremely useful for correcting mistakes, adding sudo, or repeating complex commands.

Syntax

```
!!
!<event>
!<string>
!?<string>?
```

Key Features

832

- **Last Command Repetition**: Quickly re-run the previous command
- **Command Modification**: Modify and re-execute previous commands
- **Argument Extraction**: Extract specific arguments from previous commands
- **Pattern Matching**: Find commands by pattern
- **Time Saving**: Avoid retyping complex commands

Basic Usage

Simple History Expansion

```
# Run a command
ls -la /home/user

# Repeat the last command
!!

# Add sudo to the last command
sudo !!

# This is equivalent to:
sudo ls -la /home/user
```

Common Scenarios

```
# Forgot sudo
apt update
# Permission denied

# Fix with sudo
sudo !!
# Executes: sudo apt update

# Wrong directory
cd /var/logs
# No such file or directory

# Fix the path
cd /var/log
# Then repeat with correct path
!!
```

History Expansion Patterns

1. Event Designators

```
# !! - Last command
!!

# !n - Command number n
!123

# !-n - nth command back
!-2     # Two commands ago
!-5     # Five commands ago

# !string - Most recent command starting with 'string'
!ls    # Last command starting with 'ls'
!git   # Last command starting with 'git'

# !?string? - Most recent command containing 'string'
!?config?   # Last command containing 'config'
!?file?     # Last command containing 'file'
```

2. Word Designators

```
# Previous command: git commit -m "Fix bug" --amend

# !^ - First argument
echo !^          # echo commit

# !$ - Last argument
echo !$          # echo --amend

# !* - All arguments
echo !*          # echo commit -m "Fix bug" --amend

# !:n - nth argument (0-based)
echo !:0         # echo git
echo !:1         # echo commit
echo !:2         # echo -m

# !:n-m - Range of arguments
echo !:1-3       # echo commit -m "Fix bug"
```

3. Modifiers

```
# Previous command: ls /home/user/documents/file.txt

# :h - Remove trailing pathname component (head)
echo !!:h        # echo /home/user/documents

# :t - Remove leading pathname components (tail)
echo !!:t        # echo file.txt

# :r - Remove trailing suffix
echo !!:r        # echo /home/user/documents/file

# :e - Remove all but trailing suffix
echo !!:e        # echo txt

# :s/old/new/ - Substitute first occurrence
!!:s/user/admin/    # ls /home/admin/documents/file.txt

# :gs/old/new/ - Global substitution
!!:gs/o/0/          # ls /h0me/user/d0cuments/file.txt
```

Practical Examples

1. Error Correction

```
# Typo in command
ct /etc/hosts
# Command not found

# Correct and re-run
cat /etc/hosts

# Then if you need to edit it
sudo vi !!    # Becomes: sudo vi /etc/hosts
```

2. Adding Missing Options

```
# Run command without verbose
tar -czf backup.tar.gz /home/user

# Add verbose flag
!!:s/-czf/-czvf/
# Becomes: tar -czvf backup.tar.gz /home/user

# Or simpler approach
tar -czvf backup.tar.gz /home/user
```

3. File Operations

```
# Create file
touch /tmp/test_file.txt

# Edit the same file
vi !!:$      # vi /tmp/test_file.txt

# Copy to different location
cp !!:$ /home/user/
# Becomes: cp /tmp/test_file.txt /home/user/

# Previous command: find /var/log -name "*.log" -size +10M
# Copy found files
cp !!:3 !!:4 !!:5 /backup/
# Using specific arguments from find command
```

4. Directory Navigation

```
# Change to a directory
cd /usr/local/bin

# List contents
ls !!:$      # ls /usr/local/bin

# Go back and then to related directory
cd /usr/local/src
# ... later ...
cd !!:h/bin    # cd /usr/local/bin
```

Advanced History Expansion

1. Complex Substitutions

```
# Previous: rsync -av /home/user/docs/
backup@server:/backup/user/docs/

# Change source directory
!!:s/docs/pictures/
# Becomes: rsync -av /home/user/pictures/
backup@server:/backup/user/docs/

# Change both source and destination
!!:s/docs/pictures/:s/user\/docs/user\/pictures/
# Global changes
!!:gs/docs/pictures/
```

2. Combining with Other Features

```
# Previous: find /var/log -name "*.log" -type f -exec ls -l {}
\;

# Modify to use different action
!!:s/-exec ls -l/-exec wc -l/
# Count lines instead of listing

# Extract just the find portion
!:0-4    # find /var/log -name "*.log" -type f

# Use arguments with different command
grep "error" !!:3  # grep "error" "*.log"
```

3. Working with Multiple Commands

```
# Pipeline command
ps aux | grep apache | grep -v grep

# Modify the grep pattern
!!:s/apache/nginx/
# Becomes: ps aux | grep nginx | grep -v grep

# Extract just part of pipeline
!:0-2      # ps aux | grep apache

# Add to existing pipeline
!! | wc -l    # Count the results
```

Interactive Features

1. History Verification

```
# Enable history verification (before execution)
set +H    # Disable history expansion
set -H    # Enable history expansion

# Show command before execution
shopt -s histverify
# Now !! will show the command and wait for Enter
```

2. History Search

```
# Ctrl+R - Reverse search
# Type to search through history
# Ctrl+R again to find previous matches

# Search for specific command
!?git commit?    # Find last command containing "git commit"
!?ssh?           # Find last command containing "ssh"
```

Configuration and Settings

1. History Settings

```
# History size
export HISTSIZE=1000        # Commands in memory
export HISTFILESIZE=2000    # Commands in file

# History control
export HISTCONTROL=ignoreboth    # Ignore duplicates and
spaces
export HISTCONTROL=ignoredups    # Ignore duplicates only

# History ignore patterns
export HISTIGNORE="ls:cd:cd -:pwd:exit:date:* --help"

# Timestamp in history
export HISTTIMEFORMAT="%F %T "
```

2. History Expansion Settings

```
# Check if history expansion is enabled
set +o | grep histexpand

# Enable history expansion
set -H
# or
set -o histexpand

# Disable history expansion
set +H
# or
set +o histexpand
```

Safety and Best Practices

1. Verification Before Execution

```
# Enable command verification
shopt -s histverify

# This makes !! show the command first, requiring Enter to
execute

# Check what command will be executed
history | tail -1    # See last command
echo !!              # See what !! would execute (without
running it)
```

2. Safe Practices

```
# Be careful with destructive commands
rm -rf /tmp/*
# Don't blindly run !! after such commands

# Use history to verify
history | tail -5    # Check recent commands

# For critical operations, type commands fully
# Don't rely on history expansion for:
# - rm commands
# - chmod commands
# - System configuration changes
```

3. Debugging

```
# See history expansion in action
set -x     # Enable command tracing
!!         # You'll see the expanded command
set +x     # Disable tracing

# Check history before using
history 10    # Show last 10 commands
!-2           # Run 2nd to last command
```

Common Patterns and Shortcuts

1. Frequent Combinations

```
# Add sudo to last command
sudo !!

# Redirect last command output
!! > output.txt
!! 2>&1 | tee log.txt

# Background last command
!! &

# Time last command
time !!

# Run last command in different directory
(cd /tmp && !!)
```

2. File and Directory Operations

```
# Previous: vi /etc/apache2/sites-available/default

# Test the configuration
apache2ctl -t

# Copy the file
cp !!:$ !!:$:r.backup    # cp /etc/apache2/sites-
available/default /etc/apache2/sites-available/default.backup

# Edit related file
vi !!:h/sites-enabled/default    # vi /etc/apache2/sites-
enabled/default
```

3. Network and System Commands

```
# Check service status
systemctl status apache2

# Restart if needed
sudo !!:s/status/restart/    # sudo systemctl restart apache2

# Check multiple services
systemctl status nginx
!!:s/nginx/mysql/         # systemctl status mysql
!!:s/mysql/postgresql/    # systemctl status postgresql
```

Integration with Scripts

1. History in Scripts

```bash
#!/bin/bash
# Note: History expansion doesn't work in scripts by default
# Enable it explicitly if needed

set -H    # Enable history expansion in script

# Use variables instead of history expansion in scripts
last_command="$1"
echo "Re-running: $last_command"
eval "$last_command"
```

2. Interactive Scripts

```bash
#!/bin/bash
# Interactive script using history concepts

while true; do
    read -p "Command: " cmd

    if [ "$cmd" = "!!" ]; then
        echo "Re-running: $last_cmd"
        eval "$last_cmd"
    elif [ "$cmd" = "exit" ]; then
        break
    else
        eval "$cmd"
        last_cmd="$cmd"
    fi
done
```

Alternatives and Related Commands

1. fc (Fix Command)

```
# Edit last command in editor
fc

# Edit specific command number
fc 123

# List recent commands
fc -l

# Re-run range of commands
fc -s 100 105
```

2. history command

```
# Show command history
history

# Show last 10 commands
history 10

# Execute specific command number
!123

# Search and execute
history | grep git
!456
```

Troubleshooting

849

1. History Expansion Not Working

```
# Check if enabled
set +o | grep histexpand

# Enable it
set -H

# Check in current shell
echo $-     # Should contain 'H'
```

2. Unexpected Expansions

```
# Escape ! to prevent expansion
echo "The price is \$5\!"

# Use single quotes
echo 'The price is $5!'

# Disable temporarily
set +H
echo "Commands with ! work normally"
set -H
```

3. Complex Command Issues

```
# For very complex commands, use variables
complex_cmd="find /var/log -name '*.log' -exec grep 'error' {} \;"
eval "$complex_cmd"

# Then modify variable instead of using history expansion
complex_cmd="${complex_cmd/error/warning}"
eval "$complex_cmd"
```

Important Notes

851

- **Interactive Only**: History expansion primarily works in interactive shells
- **Not in Scripts**: Usually disabled in scripts for safety
- **Shell Specific**: This is a bash/zsh feature, not available in all shells
- **Verification**: Use `histverify` option for safety with destructive commands
- **Case Sensitive**: History expansion is case-sensitive
- **Immediate Execution**: !! executes immediately; use caution with destructive commands

The `!!` command and history expansion features are powerful tools for efficient command-line work, but they require understanding and careful use to avoid mistakes.

For more details, check the bash manual: `man bash` (search for "History Expansion")

The `tty` command

The `tty` command prints the filename of the terminal connected to standard input. It shows which terminal device you're currently using and can determine if the input is coming from a terminal or being redirected from a file or pipe.

Syntax

853

```
tty [options]
```

Key Features

854

- **Terminal Identification**: Shows current terminal device
- **Redirection Detection**: Determines if input is from terminal or file
- **Session Information**: Helps identify terminal sessions
- **Scripting Support**: Exit codes indicate terminal vs non-terminal

Basic Usage

Show Current Terminal

```
# Display current terminal device
tty

# Example outputs:
# /dev/pts/0    (pseudo-terminal)
# /dev/tty1     (console terminal)
# /dev/ttys000  (macOS terminal)
```

Check If Terminal

```
# Silent mode - just check if it's a terminal
tty -s

# Check exit code
tty -s && echo "Running in terminal" || echo "Not in terminal"

# Example usage in scripts
if tty -s; then
    echo "Interactive mode"
else
    echo "Non-interactive mode"
fi
```

Common Options

856

Basic Options

```
# -s: Silent mode (no output, just exit code)
tty -s

# --help: Show help information
tty --help

# --version: Show version information
tty --version
```

Understanding Terminal Types

1. Physical Terminals

```
# Console terminals (virtual consoles)
# /dev/tty1, /dev/tty2, /dev/tty3, etc.

# Switch to virtual console and check
# Ctrl+Alt+F1 (then login)
tty
# Output: /dev/tty1
```

2. Pseudo Terminals

```
# Most common in desktop environments
# /dev/pts/0, /dev/pts/1, /dev/pts/2, etc.

# In terminal emulator
tty
# Output: /dev/pts/0

# Each new terminal window gets new pts number
```

3. Serial Terminals

```
# Serial console connections
# /dev/ttyS0, /dev/ttyS1, etc.

# USB serial devices
# /dev/ttyUSB0, /dev/ttyUSB1, etc.
```

Practical Examples

858

1. Session Identification

```
# Identify current session
echo "Current session: $(tty)"

# Show user and terminal
echo "User: $(whoami) on $(tty)"

# Show all users and their terminals
who

# Show current user's terminals
who | grep $(whoami)
```

2. Multi-Terminal Scripts

```bash
#!/bin/bash
# Script that behaves differently based on terminal

current_tty=$(tty)
echo "Running on: $current_tty"

case "$current_tty" in
    /dev/tty[1-6])
        echo "Running on virtual console"
        # Console-specific behavior
        ;;
    /dev/pts/*)
        echo "Running in terminal emulator"
        # Terminal emulator behavior
        ;;
    *)
        echo "Unknown terminal type"
        ;;
esac
```

3. Interactive vs Non-Interactive Detection

```bash
#!/bin/bash
# Detect if script is running interactively

if tty -s; then
    echo "Interactive mode - can prompt user"
    read -p "Enter your name: " name
    echo "Hello, $name!"
else
    echo "Non-interactive mode - using defaults"
    name="User"
    echo "Hello, $name!"
fi
```

4. Terminal-Specific Configuration

```bash
#!/bin/bash
# Configure based on terminal capabilities

current_tty=$(tty)

if [[ "$current_tty" =~ ^/dev/pts/ ]]; then
    # Modern terminal emulator
    echo -e "\e[32mGreen text\e[0m"
    echo -e "\e[1mBold text\e[0m"
elif [[ "$current_tty" =~ ^/dev/tty[1-6]$ ]]; then
    # Virtual console - limited capabilities
    echo "Plain text output"
else
    echo "Unknown terminal - safe output"
fi
```

Scripting Applications

1. Conditional User Interaction

```bash
#!/bin/bash
# Only prompt if running in terminal

ask_confirmation() {
    local message="$1"

    if tty -s; then
        read -p "$message (y/n): " response
        case "$response" in
            [Yy]|[Yy][Ee][Ss]) return 0 ;;
            *) return 1 ;;
        esac
    else
        # Non-interactive - assume yes
        echo "$message: Assumed yes (non-interactive)"
        return 0
    fi
}

# Usage
if ask_confirmation "Delete old files?"; then
    echo "Deleting files..."
else
    echo "Keeping files..."
fi
```

2. Progress Indicators

```bash
#!/bin/bash
# Show progress only in terminal

show_progress() {
    if tty -s; then
        echo -n "Processing: "
        for i in {1..10}; do
            echo -n "."
            sleep 0.5
        done
        echo " Done!"
    else
        echo "Processing... Done!"
    fi
}

show_progress
```

3. Logging Behavior

```bash
#!/bin/bash
# Different logging based on terminal

log_message() {
    local level="$1"
    local message="$2"
    local timestamp=$(date '+%Y-%m-%d %H:%M:%S')

    if tty -s; then
        # Terminal output - colored
        case "$level" in
            INFO)  echo -e "\e[32m[$timestamp] INFO:
$message\e[0m" ;;
            WARN)  echo -e "\e[33m[$timestamp] WARN:
$message\e[0m" ;;
            ERROR) echo -e "\e[31m[$timestamp] ERROR:
$message\e[0m" ;;
        esac
    else
        # Non-terminal output - plain
        echo "[$timestamp] $level: $message"
    fi
}


# Usage
log_message "INFO" "Script started"
log_message "WARN" "Low disk space"
log_message "ERROR" "Connection failed"
```

Terminal Session Management

1. Session Information

```bash
# Get detailed session info
get_session_info() {
    echo "=== Session Information ==="
    echo "Terminal: $(tty)"
    echo "User: $(whoami)"
    echo "Shell: $SHELL"
    echo "PID: $$"
    echo "PPID: $PPID"
    echo "Session ID: $(ps -o sid= -p $$)"
    echo "Process Group: $(ps -o pgid= -p $$)"
}

get_session_info
```

2. Multiple Terminal Detection

```bash
# Count user's active terminals
count_user_terminals() {
    local user=$(whoami)
    local count=$(who | grep "^$user " | wc -l)
    echo "User $user has $count active terminals"

    echo "Active sessions:"
    who | grep "^$user " | while read line; do
        echo "  $line"
    done
}

count_user_terminals
```

3. Terminal Communication

```bash
# Send message to specific terminal
send_to_terminal() {
    local target_tty="$1"
    local message="$2"

    if [ -w "$target_tty" ]; then
        echo "Message from $(tty): $message" > "$target_tty"
        echo "Message sent to $target_tty"
    else
        echo "Cannot write to $target_tty"
    fi
}

# Usage (if permissions allow)
# send_to_terminal "/dev/pts/1" "Hello from another terminal!"
```

Integration with Other Commands

1. Combining with ps

```
# Find processes in current terminal
current_tty=$(tty | sed 's|/dev/||')
ps -t "$current_tty"

# Show process tree for current terminal
pstree -p $(ps -t "$current_tty" -o pid --no-headers | head
-1)
```

2. Combining with who/w

```
# Show who is on the same terminal type
current_tty_type=$(tty | cut -d'/' -f3 | sed 's/[0-9]*$//')
who | grep "$current_tty_type"

# Detailed information about current session
w | grep "$(tty | sed 's|/dev/||')"
```

3. System Monitoring

```bash
# Monitor terminal activity
monitor_terminals() {
    echo "Active terminals:"
    ls -la /dev/pts/
    echo

    echo "Users and terminals:"
    who
    echo

    echo "Current session:"
    echo "  TTY: $(tty)"
    echo "  Uptime: $(uptime)"
}

monitor_terminals
```

Security and Permissions

1. Terminal Permissions

```bash
# Check terminal permissions
check_tty_permissions() {
    local current_tty=$(tty)
    echo "Terminal: $current_tty"
    ls -la "$current_tty"

    # Check if others can write to terminal
    if [ -w "$current_tty" ]; then
        echo "Terminal is writable"
    else
        echo "Terminal is not writable"
    fi
}

check_tty_permissions
```

2. Secure Terminal Check

```
# Verify secure terminal environment
check_secure_terminal() {
    if ! tty -s; then
        echo "WARNING: Not running in a terminal"
        return 1
    fi

    local current_tty=$(tty)
    local tty_perms=$(ls -la "$current_tty" | cut -d' ' -f1)

    if [[ "$tty_perms" =~ .*w.*w.* ]]; then
        echo "WARNING: Terminal is world-writable"
        return 1
    fi

    echo "Terminal security check passed"
    return 0
}

check_secure_terminal
```

Debugging and Troubleshooting

1. Terminal Issues

```
# Debug terminal problems
debug_terminal() {
    echo "=== Terminal Debug Information ==="
    echo "TTY: $(tty 2>/dev/null || echo "No TTY")"
    echo "TERM: $TERM"
    echo "Interactive: $(tty -s && echo "Yes" || echo "No")"
    echo "Standard input: $(file /proc/self/fd/0 | cut -d: -
f2-)"
    echo "Standard output: $(file /proc/self/fd/1 | cut -d: -
f2-)"
    echo "Standard error: $(file /proc/self/fd/2 | cut -d: -
f2-)"
}

debug_terminal
```

2. Redirection Detection

```bash
# Detect various input/output scenarios
detect_io_redirection() {
    echo "Input/Output Analysis:"

    if tty -s; then
        echo "  Standard input: Terminal ($(tty))"
    else
        echo "  Standard input: Redirected/Pipe"
    fi

    if [ -t 1 ]; then
        echo "  Standard output: Terminal"
    else
        echo "  Standard output: Redirected/Pipe"
    fi

    if [ -t 2 ]; then
        echo "  Standard error: Terminal"
    else
        echo "  Standard error: Redirected/Pipe"
    fi
}

detect_io_redirection
```

3. Session Recovery

```bash
# Help recover lost terminal sessions
find_my_sessions() {
    local user=$(whoami)
    echo "Finding sessions for user: $user"

    echo "Current TTY: $(tty)"
    echo

    echo "All active sessions:"
    who | grep "^$user "
    echo

    echo "Screen sessions:"
    screen -ls 2>/dev/null || echo "No screen sessions"
    echo

    echo "Tmux sessions:"
    tmux list-sessions 2>/dev/null || echo "No tmux sessions"
}

find_my_sessions
```

Advanced Usage

1. Terminal Multiplexing Integration

```bash
# Detect if running inside multiplexer
detect_multiplexer() {
    if [ -n "$TMUX" ]; then
        echo "Running inside tmux"
        echo "  Session: $(tmux display-message -p '#S')"
        echo "  Window: $(tmux display-message -p '#W')"
        echo "  Pane: $(tmux display-message -p '#P')"
    elif [ -n "$STY" ]; then
        echo "Running inside screen"
        echo "  Session: $STY"
    else
        echo "Not in a multiplexer"
    fi

    echo "TTY: $(tty)"
}

detect_multiplexer
```

2. Remote Session Detection

```bash
# Detect remote vs local sessions
detect_session_type() {
    local current_tty=$(tty)

    if [ -n "$SSH_CONNECTION" ]; then
        echo "Remote SSH session"
        echo "  From: $(echo $SSH_CONNECTION | cut -d' ' -f1,2)"
        echo "  To: $(echo $SSH_CONNECTION | cut -d' ' -f3,4)"
    elif [[ "$current_tty" =~ ^/dev/tty[1-6]$ ]]; then
        echo "Local console session"
    elif [[ "$current_tty" =~ ^/dev/pts/ ]]; then
        echo "Local terminal emulator"
    else
        echo "Unknown session type"
    fi

    echo "TTY: $current_tty"
}

detect_session_type
```

Best Practices

1. Safe Scripting

```bash
# Always check for terminal before interactive operations
safe_interactive() {
    if ! tty -s; then
        echo "Error: This script requires a terminal" >&2
        exit 1
    fi

    # Proceed with interactive operations
    read -p "Continue? (y/n): " response
}
```

2. Cross-Platform Compatibility

```bash
# Handle different systems
get_terminal_info() {
    if command -v tty >/dev/null 2>&1; then
        local terminal=$(tty 2>/dev/null)
        if [ $? -eq 0 ]; then
            echo "$terminal"
        else
            echo "not a tty"
        fi
    else
        echo "tty command not available"
    fi
}
```

3. Error Handling

```bash
# Robust terminal checking
check_terminal_safe() {
    local tty_output
    tty_output=$(tty 2>/dev/null)
    local exit_code=$?

    if [ $exit_code -eq 0 ]; then
        echo "Terminal: $tty_output"
        return 0
    else
        echo "Not a terminal (exit code: $exit_code)"
        return 1
    fi
}
```

Important Notes

- **Exit Codes**: tty returns 0 if stdin is a terminal, non-zero otherwise
- **Silent Mode**: Use `-s` for scripts that only need to check terminal status
- **Redirection**: Output changes when stdin is redirected from files or pipes
- **Security**: Be aware of terminal permissions and write access
- **Portability**: Available on most Unix-like systems
- **Session Management**: Useful for multiplexing and session tracking

The `tty` command is essential for scripts that need to detect terminal environments and behave appropriately in interactive vs non-interactive contexts.

For more details, check the manual: `man tty`

The `lspci` command

The `lspci` command lists all PCI (Peripheral Component Interconnect) devices connected to the system. It provides detailed information about hardware components including graphics cards, network adapters, sound cards, storage controllers, and other PCI-based devices.

Syntax

879

```
lspci [options]
```

Key Features

- **Hardware Detection**: Lists all PCI devices
- **Detailed Information**: Vendor, device, class, and capabilities
- **Tree View**: Shows device hierarchy and relationships
- **Filtering Options**: Search by vendor, device, or class
- **Verbose Output**: Multiple levels of detail

Basic Usage

Simple Device Listing

```
# List all PCI devices
lspci

# Example output:
# 00:00.0 Host bridge: Intel Corporation Device 9b61
# 00:01.0 PCI bridge: Intel Corporation Device 1901
# 00:02.0 VGA compatible controller: Intel Corporation Device
9bc4
# 00:14.0 USB controller: Intel Corporation Device a36d
```

Human-Readable Output

```
# Show device names instead of just IDs
lspci -nn

# More verbose output
lspci -v

# Very verbose output (includes everything)
lspci -vv

# Extremely verbose output
lspci -vvv
```

Common Options

Basic Options

```
# -v: Verbose (show detailed info)
lspci -v

# -vv: Very verbose
lspci -vv

# -n: Show numeric IDs instead of names
lspci -n

# -nn: Show both names and numeric IDs
lspci -nn

# -k: Show kernel drivers
lspci -k
```

Display Options

```
# -t: Tree format
lspci -t

# -tv: Tree format with verbose info
lspci -tv

# -x: Show hex dump of config space
lspci -x

# -xxx: Show full hex dump
lspci -xxx
```

Filtering and Selection

1. By Device Type

```
# Graphics cards
lspci | grep -i vga
lspci | grep -i display

# Network adapters
lspci | grep -i network
lspci | grep -i ethernet

# USB controllers
lspci | grep -i usb

# Sound cards
lspci | grep -i audio
lspci | grep -i sound

# Storage controllers
lspci | grep -i storage
lspci | grep -i sata
```

2. By Vendor

```
# Intel devices
lspci | grep -i intel

# AMD devices
lspci | grep -i amd

# NVIDIA devices
lspci | grep -i nvidia

# Broadcom devices
lspci | grep -i broadcom
```

3. Specific Device Selection

```
# Select specific device by bus ID
lspci -s 00:02.0

# Select multiple devices
lspci -s 00:02.0,00:14.0

# Select by vendor ID
lspci -d 8086:

# Select by vendor and device ID
lspci -d 8086:9bc4
```

Detailed Information

1. Verbose Device Information

```
# Show capabilities and features
lspci -v

# Example output includes:
# - Memory addresses
# - IRQ assignments
# - Capabilities (MSI, Power Management, etc.)
# - Kernel modules/drivers

# Very detailed information
lspci -vv
# Includes:
# - Extended capabilities
# - Configuration registers
# - Link status and speeds
```

2. Driver Information

```
# Show which kernel drivers are in use
lspci -k

# Example output:
# 00:02.0 VGA compatible controller: Intel Corporation Device
9bc4
#        Subsystem: Dell Device 097d
#        Kernel driver in use: i915
#        Kernel modules: i915

# Combine with verbose for more detail
lspci -vk
```

3. Configuration Space

```
# Show configuration space (hex dump)
lspci -x

# Full configuration space
lspci -xxx

# Specific device configuration
lspci -s 00:02.0 -xxx
```

Tree View and Topology

1. Device Hierarchy

```
# Show device tree structure
lspci -t

# Example output:
# -[0000:00]-+-00.0  Intel Corporation Device 9b61
#           +-01.0-[01]--
#           +-02.0  Intel Corporation Device 9bc4
#           +-14.0  Intel Corporation Device a36d

# Tree with device names
lspci -tv
```

2. Bus Information

```
# Show bridges and connections
lspci -tv | grep -E "(bridge|Bridge)"

# PCI Express information
lspci -vv | grep -A 5 -B 5 "Express"

# Link capabilities and status
lspci -vv | grep -E "(Link|Speed|Width)"
```

Hardware Analysis

1. Graphics Card Information

```bash
# List graphics devices
get_gpu_info() {
    echo "=== Graphics Cards ==="
    lspci | grep -i -E "(vga|display|3d)"
    echo

    echo "=== Detailed GPU Information ==="
    lspci -v | grep -A 10 -B 2 -i -E "(vga|display)"
    echo

    echo "=== GPU Drivers ==="
    lspci -k | grep -A 3 -B 1 -i -E "(vga|display)"
}

get_gpu_info
```

2. Network Interface Analysis

```
# Network adapter details
get_network_info() {
    echo "=== Network Adapters ==="
    lspci | grep -i -E "(network|ethernet|wireless)"
    echo

    echo "=== Network Driver Information ==="
    lspci -k | grep -A 3 -B 1 -i -E "(network|ethernet)"
    echo

    echo "=== Wireless Capabilities ==="
    lspci -vv | grep -A 20 -B 5 -i wireless
}

get_network_info
```

3. Storage Controller Information

```
# Storage device details
get_storage_info() {
    echo "=== Storage Controllers ==="
    lspci | grep -i -E "(storage|sata|ide|scsi|nvme)"
    echo

    echo "=== Storage Driver Information ==="
    lspci -k | grep -A 3 -B 1 -i -E "(storage|sata|ahci)"
    echo

    echo "=== SATA Capabilities ==="
    lspci -vv | grep -A 10 -B 2 -i sata
}

get_storage_info
```

System Analysis Scripts

1. Hardware Inventory

```bash
#!/bin/bash
# Complete hardware inventory using lspci

hardware_inventory() {
    echo "=== PCI Hardware Inventory ==="
    echo "Generated: $(date)"
    echo

    echo "--- System Overview ---"
    lspci | wc -l | xargs echo "Total PCI devices:"
    echo

    echo "--- Graphics ---"
    lspci | grep -i -E "(vga|display|3d)" || echo "No graphics
devices found"
    echo

    echo "--- Network ---"
    lspci | grep -i -E "(network|ethernet|wireless)" || echo
"No network devices found"
    echo

    echo "--- Storage ---"
    lspci | grep -i -E "(storage|sata|ide|nvme)" || echo "No
storage controllers found"
    echo

    echo "--- Audio ---"
    lspci | grep -i -E "(audio|sound|multimedia)" || echo "No
audio devices found"
    echo

    echo "--- USB Controllers ---"
    lspci | grep -i usb || echo "No USB controllers found"
    echo
```

```
    echo "--- Other Devices ---"
    lspci | grep -v -i -E
"(vga|display|3d|network|ethernet|wireless|storage|sata|ide|nv
me|audio|sound|multimedia|usb|bridge|host)" || echo "No other
devices"
}

hardware_inventory > hardware_inventory.txt
```

2. Driver Status Check

```
#!/bin/bash
# Check driver status for all PCI devices

check_drivers() {
    echo "=== PCI Driver Status ==="

    lspci | while read line; do
        device_id=$(echo "$line" | cut -d' ' -f1)
        device_name=$(echo "$line" | cut -d' ' -f2-)

        driver_info=$(lspci -k -s "$device_id" | grep "Kernel
driver in use")

        if [ -n "$driver_info" ]; then
            driver=$(echo "$driver_info" | cut -d':' -f2 |
xargs)
            echo "✓ $device_id: $device_name -> $driver"
        else
            echo "✗ $device_id: $device_name -> NO DRIVER"
        fi
    done
}

check_drivers
```

3. Performance Analysis

```bash
#!/bin/bash
# Analyze PCI device performance capabilities

analyze_performance() {
    echo "=== PCI Performance Analysis ==="

    echo "--- PCIe Link Speeds ---"
    lspci -vv | grep -A 1 -B 1 "Link.*Speed" | grep -E
"(:|Speed|Width)"
    echo

    echo "--- Memory Mappings ---"
    lspci -v | grep -E "(Memory at|I/O ports at)" | sort |
uniq -c | sort -nr
    echo

    echo "--- Power Management ---"
    lspci -vv | grep -c "Power Management" | xargs echo
"Devices with power management:"
    echo

    echo "--- MSI Capabilities ---"
    lspci -vv | grep -c "MSI:" | xargs echo "Devices with MSI
support:"
    echo

    echo "--- 64-bit Devices ---"
    lspci -vv | grep -c "64-bit" | xargs echo "64-bit capable
devices:"
}

analyze_performance
```

Troubleshooting

1. Device Detection Issues

```bash
# Check if device is detected
check_device_detection() {
    local search_term="$1"

    echo "Searching for: $search_term"

    devices=$(lspci | grep -i "$search_term")
    if [ -n "$devices" ]; then
        echo "✓ Device(s) found:"
        echo "$devices"
        echo

        echo "Driver information:"
        echo "$devices" | while read line; do
            device_id=$(echo "$line" | cut -d' ' -f1)
            lspci -k -s "$device_id" | grep -E
"(driver|module)"
        done
    else
        echo "✗ No devices found matching '$search_term'"
        echo "Try checking:"
        echo "  - Physical connections"
        echo "  - BIOS/UEFI settings"
        echo "  - Power supply"
    fi
}

# Usage examples
check_device_detection "graphics"
check_device_detection "network"
```

2. Driver Issues

```
# Find devices without drivers
find_missing_drivers() {
    echo "=== Devices Without Drivers ==="

    lspci | while read line; do
        device_id=$(echo "$line" | cut -d' ' -f1)
        device_name=$(echo "$line" | cut -d' ' -f2-)

        if ! lspci -k -s "$device_id" | grep -q "Kernel driver
in use"; then
            echo "Missing driver: $device_id - $device_name"

            # Try to find available modules
            modules=$(lspci -k -s "$device_id" | grep "Kernel
modules:" | cut -d':' -f2)
            if [ -n "$modules" ]; then
                echo "  Available modules:$modules"
            fi
        fi
    done
}

find_missing_drivers
```

3. Hardware Compatibility

```bash
# Check hardware compatibility
check_compatibility() {
    echo "=== Hardware Compatibility Check ==="

    echo "--- Unsupported Devices ---"
    lspci -nn | while read line; do
        if echo "$line" | grep -q "\[ffff:"; then
            echo "Possible unsupported device: $line"
        fi
    done
    echo

    echo "--- Legacy Devices ---"
    lspci | grep -i -E "(legacy|isa|parallel|serial|floppy)"
|| echo "No legacy devices found"
    echo

    echo "--- Vendor Support ---"
    echo "Intel devices: $(lspci | grep -i intel | wc -l)"
    echo "AMD devices: $(lspci | grep -i amd | wc -l)"
    echo "NVIDIA devices: $(lspci | grep -i nvidia | wc -l)"
    echo "Other vendors: $(lspci | grep -v -i -E
"(intel|amd|nvidia)" | wc -l)"
}

check_compatibility
```

Advanced Usage

1. Configuration Space Analysis

```
# Analyze specific device configuration
analyze_device_config() {
    local device_id="$1"

    echo "=== Configuration Analysis for $device_id ==="

    echo "--- Basic Information ---"
    lspci -s "$device_id" -v
    echo

    echo "--- Configuration Space ---"
    lspci -s "$device_id" -x
    echo

    echo "--- Capabilities ---"
    lspci -s "$device_id" -vv | grep -A 20 "Capabilities:"
}

# Usage: analyze_device_config "00:02.0"
```

2. Bandwidth Analysis

```bash
# Analyze PCIe bandwidth
analyze_bandwidth() {
    echo "=== PCIe Bandwidth Analysis ==="

    lspci -vv | grep -A 2 -B 2 "Express.*Root
Port\|Express.*Endpoint" | \
    while read line; do
        if echo "$line" | grep -q "Express"; then
            echo "Device: $line"
        elif echo "$line" | grep -q "Link.*Speed"; then
            echo "  $line"
        fi
    done
}

analyze_bandwidth
```

3. Power Management

```bash
# Check power management capabilities
check_power_management() {
    echo "=== Power Management Status ==="

    lspci -vv | grep -B 5 -A 10 "Power Management" | \
    grep -E "(^[0-9a-f]{2}:[0-9a-f]{2}\.[0-9]|Power
Management|PME)"
}

check_power_management
```

Integration with Other Tools

1. Combining with lsmod

```
# Match PCI devices with loaded modules
match_devices_modules() {
    echo "=== PCI Devices and Kernel Modules ==="

    lspci -k | grep -E "(^[0-9a-f]{2}:|Kernel driver|Kernel
modules)" | \
    while read line; do
        if [[ "$line" =~ ^[0-9a-f]{2}: ]]; then
            echo "$line"
        elif [[ "$line" =~ "Kernel driver in use:" ]]; then
            driver=$(echo "$line" | cut -d':' -f2 | xargs)
            echo "  Active driver: $driver"
            lsmod | grep "^$driver" && echo "    ✓ Module
loaded" || echo "    ✗ Module not found"
        fi
    done
}
```

2. Combining with udev

```bash
# Check udev rules for PCI devices
check_udev_rules() {
    local device_id="$1"

    echo "Checking udev rules for device: $device_id"

    # Get vendor and device IDs
    vendor_device=$(lspci -n -s "$device_id" | awk '{print
$3}')
    vendor_id=$(echo "$vendor_device" | cut -d':' -f1)
    device_id_hex=$(echo "$vendor_device" | cut -d':' -f2)

    echo "Vendor ID: $vendor_id, Device ID: $device_id_hex"

    # Search udev rules
    find /etc/udev/rules.d /lib/udev/rules.d -name "*.rules" -
exec grep -l "$vendor_id\|$device_id_hex" {} \; 2>/dev/null
}
```

Security Considerations

1. Hardware Security

```
# Check for security-relevant hardware
check_security_hardware() {
    echo "=== Security Hardware Check ==="

    echo "--- TPM Devices ---"
    lspci | grep -i tpm || echo "No TPM devices found"
    echo

    echo "--- Virtualization Support ---"
    lspci -vv | grep -i -E "(vt-x|amd-v|virtualization)" ||
echo "Check CPU flags for virtualization"
    echo

    echo "--- IOMMU Support ---"
    lspci -vv | grep -i iommu || echo "No explicit IOMMU
references found"
    echo

    echo "--- Hardware Security Modules ---"
    lspci | grep -i -E "(security|crypto|hsm)" || echo "No HSM
devices found"
}

check_security_hardware
```

Best Practices

1. Regular Hardware Monitoring

```bash
# Create hardware monitoring script
#!/bin/bash
# Monitor hardware changes

BASELINE_FILE="/var/lib/hardware_baseline.txt"
CURRENT_FILE="/tmp/current_hardware.txt"

# Generate current state
lspci -nn > "$CURRENT_FILE"

# Compare with baseline
if [ -f "$BASELINE_FILE" ]; then
    if ! diff -q "$BASELINE_FILE" "$CURRENT_FILE" >/dev/null;
then
        echo "Hardware configuration changed!"
        echo "Changes:"
        diff "$BASELINE_FILE" "$CURRENT_FILE"
    fi
else
    echo "Creating hardware baseline"
fi

# Update baseline
cp "$CURRENT_FILE" "$BASELINE_FILE"
```

2. Documentation

```bash
# Generate hardware documentation
document_hardware() {
    local output_file="hardware_documentation_$(date
+%Y%m%d).txt"

    {
        echo "Hardware Documentation"
        echo "Generated: $(date)"
        echo "Hostname: $(hostname)"
        echo "========================="
        echo

        echo "PCI Device Summary:"
        lspci | wc -l | xargs echo "Total devices:"
        echo

        echo "Detailed Device List:"
        lspci -nn
        echo

        echo "Driver Status:"
        lspci -k
        echo

        echo "Device Tree:"
        lspci -tv

    } > "$output_file"

    echo "Documentation saved to: $output_file"
}
```

Important Notes

- **Root Access**: Some detailed information requires root privileges
- **Hardware Detection**: Only shows devices connected to PCI bus
- **Driver Status**: Shows currently loaded drivers, not all available drivers
- **Updates**: Device information is read from kernel, may require hardware rescan
- **Vendor IDs**: Numeric IDs are standardized, names come from PCI ID database
- **Tree View**: Shows physical bus topology and device relationships

The `lspci` command is essential for hardware troubleshooting, driver management, and system analysis.

For more details, check the manual: `man lspci`

The `cfdisk` command

The `cfdisk` command is a curses-based disk partitioning tool for Linux. It provides a user-friendly, text-based interface for creating, deleting, and managing disk partitions. Unlike `fdisk`, `cfdisk` offers a more intuitive menu-driven approach.

Syntax

905

```
cfdisk [options] [device]
```

Key Features

906

- **Interactive Interface**: Menu-driven partition management
- **Real-time Display**: Shows current partition table
- **Safe Operations**: Requires explicit write command to save changes
- **Multiple File Systems**: Supports various partition types
- **Resize Support**: Can resize existing partitions

Basic Usage

Starting cfdisk

```
# Open cfdisk on primary disk
sudo cfdisk /dev/sda

# Open cfdisk on secondary disk
sudo cfdisk /dev/sdb

# Auto-detect and use first available disk
sudo cfdisk
```

Navigation

- **Up**/**Down Arrows**: Navigate between partitions
- **Left**/**Right Arrows**: Navigate between menu options
- **Enter**: Select menu option
- **Tab**: Move between interface elements

Menu Options

Main Operations

- **New**: Create a new partition
- **Delete**: Delete selected partition
- **Resize**: Resize selected partition
- **Type**: Change partition type
- **Write**: Write changes to disk
- **Quit**: Exit without saving (if no changes written)

Additional Options

- **Bootable**: Toggle bootable flag
- **Verify**: Check partition table consistency
- **Print**: Display partition information

Common Tasks

1. Creating a New Partition

```
# Start cfdisk
sudo cfdisk /dev/sdb

# Steps in cfdisk:
# 1. Select "New" from menu
# 2. Choose partition size
# 3. Select partition type (primary/extended)
# 4. Choose partition type (Linux, swap, etc.)
# 5. Select "Write" to save changes
# 6. Type "yes" to confirm
# 7. Select "Quit" to exit
```

2. Deleting a Partition

```
# In cfdisk interface:
# 1. Navigate to partition to delete
# 2. Select "Delete" from menu
# 3. Confirm deletion
# 4. Select "Write" to save changes
# 5. Type "yes" to confirm
```

3. Changing Partition Type

```
# In cfdisk interface:
# 1. Navigate to target partition
# 2. Select "Type" from menu
# 3. Choose new partition type from list
# 4. Select "Write" to save changes
```

Common Partition Types

Linux Partition Types

- **83**: Linux filesystem
- **82**: Linux swap
- **8e**: Linux LVM
- **fd**: Linux RAID autodetect

Other Common Types

- **07**: NTFS/HPFS
- **0c**: FAT32 LBA
- **ef**: EFI System Partition
- **01**: FAT12

Command Line Options

```
# Show help
cfdisk --help

# Show version
cfdisk --version

# Use alternative device
cfdisk /dev/sdc

# Start with specific partition table type
cfdisk -t gpt /dev/sdb
cfdisk -t dos /dev/sdb
```

Practical Examples

1. Partitioning a New USB Drive

```
# Insert USB drive (assume it's /dev/sdc)
# Check device name
lsblk

# Start cfdisk
sudo cfdisk /dev/sdc

# Create new partition table if needed
# Create partitions as needed
# Write changes and quit
```

2. Adding Swap Partition

```
# Start cfdisk on target disk
sudo cfdisk /dev/sda

# Create new partition
# Set type to "82" (Linux swap)
# Write changes

# Format as swap
sudo mkswap /dev/sda3

# Enable swap
sudo swapon /dev/sda3
```

3. Preparing Disk for LVM

```
# Start cfdisk
sudo cfdisk /dev/sdb

# Create partition
# Set type to "8e" (Linux LVM)
# Write changes

# Create physical volume
sudo pvcreate /dev/sdb1
```

Safety Features

914

1. Change Tracking

```
# cfdisk tracks all changes
# Shows asterisk (*) next to modified partitions
# Changes only applied when "Write" is selected
```

2. Confirmation Required

```
# Writing changes requires typing "yes"
# Provides final warning before applying changes
# Can quit without saving if no "Write" performed
```

3. Verification

```
# Built-in partition table verification
# Warns about potential issues
# Suggests corrections for problems
```

Working with Different Partition Tables

1. GPT (GUID Partition Table)

```
# Create GPT partition table
sudo cfdisk -t gpt /dev/sdb

# Features:
# - Supports >2TB disks
# - Up to 128 partitions
# - Backup partition table
# - 64-bit LBA addressing
```

2. MBR/DOS Partition Table

```
# Create MBR partition table
sudo cfdisk -t dos /dev/sdb

# Limitations:
# - 4 primary partitions maximum
# - 2TB disk size limit
# - Extended partitions for >4 partitions
```

Integration with Other Tools

1. After Partitioning

```
# Verify partition creation
lsblk /dev/sdb
fdisk -l /dev/sdb

# Format partitions
sudo mkfs.ext4 /dev/sdb1
sudo mkfs.xfs /dev/sdb2

# Mount partitions
sudo mkdir /mnt/partition1
sudo mount /dev/sdb1 /mnt/partition1
```

2. Backup Before Changes

```
# Backup partition table before changes
sudo sfdisk -d /dev/sdb > sdb_partition_backup.txt

# Restore if needed
sudo sfdisk /dev/sdb < sdb_partition_backup.txt
```

Troubleshooting

1. Permission Issues

```
# Must run as root or with sudo
sudo cfdisk /dev/sdb

# Check device permissions
ls -l /dev/sdb
```

2. Device Busy

```
# Unmount all partitions on device first
sudo umount /dev/sdb1
sudo umount /dev/sdb2

# Check for active processes
lsof /dev/sdb*
```

3. Partition Table Corruption

```
# Verify partition table
sudo cfdisk /dev/sdb

# If corrupted, recreate partition table
# (This will destroy all data!)
sudo cfdisk -t gpt /dev/sdb  # For GPT
sudo cfdisk -t dos /dev/sdb  # For MBR
```

Best Practices

1. Always Backup

```
# Backup important data before partitioning
# Create partition table backup
sudo sfdisk -d /dev/sdb > partition_backup.txt
```

2. Verify Device

```
# Double-check device name before starting
lsblk
fdisk -l

# Ensure you're working on correct disk
```

3. Plan Partition Layout

```
# Plan your partition scheme:
# - Root partition (/)
# - Swap partition
# - Home partition (/home)
# - Boot partition (/boot) if needed
```

4. Consider Alignment

```
# Modern cfdisk handles alignment automatically
# Uses 1MB alignment by default
# Optimal for SSDs and advanced format drives
```

Comparison with Other Tools

vs fdisk

- **cfdisk**: Menu-driven, user-friendly
- **fdisk**: Command-driven, more scriptable

vs parted

- **cfdisk**: Simpler interface, basic operations
- **parted**: More advanced features, command-line scriptable

vs gparted

- **cfdisk**: Text-based, lightweight
- **gparted**: Graphical interface, requires X11

Important Notes

921

- Always unmount partitions before modifying them
- Changes are not written until you explicitly choose "Write"
- Backup important data before making partition changes
- Some operations may require a system reboot to take effect
- Be extremely careful when working with system disks
- Consider using LVM for more flexible partition management

The `cfdisk` command provides an excellent balance between ease of use and functionality for disk partitioning tasks.

For more details, check the manual: `man cfdisk`

The ifplugstatus Command

The `ifplugstatus` command is a diagnostic utility used to check the **physical link status** of network interfaces on Linux systems. It reports whether an interface (such as eth0, enp0s3, or wlan0) has a network cable connected or not. This tool is particularly useful for troubleshooting wired network connectivity and detecting unplugged cables.

Syntax

923

```
ifplugstatus [options] [interface]
```

Parameters

- interface — the network interface to check (e.g., eth0, enp0s3, wlan0).

- options — optional flags for customizing output.

Installation

924

`ifplugstatus` is part of the `ifplugd` package. It can be installed using the following commands depending on your distribution:

```
# Ubuntu/Debian
sudo apt update
sudo apt install ifplugd

# CentOS/RHEL/Fedora
sudo yum install ifplugd
# or
sudo dnf install ifplugd
```

To verify installation:

```
which ifplugstatus
```

Basic Usage

1. Check the status of a single interface

```
ifplugstatus eth0
```

Sample Output:

```
eth0: link beat detected
```

or

```
eth0: link beat not detected
```

Explanation:

- link beat detected → cable is plugged in and active.

- link beat not detected → cable is unplugged or inactive. *Note: 'link beat' is a legacy term from older Ethernet (10/100 Mbps). Modern Gigabit or higher interfaces may use different link detection signals, but the command generally reports an active physical link.*

2. Check multiple interfaces

```
ifplugstatus eth0 wlan0
```

Sample Output:

```
eth0: link beat detected
wlan0: link beat not detected
```

This allows quick verification of all available interfaces.

Options

927

Some popular option flags include:

Option	Description
-a	Show all interfaces (default behavior).
-i	Specify a particular interface.
-s	Display a short summary only.
-v	Verbose output with more details.
-q	Quiet mode — minimal output.
-h	Display help information.

Practical Examples

1. Check if the main Ethernet interface is connected

```
ifplugstatus eth0
```

Displays the current cable connection status for eth0.

2. Check all available network interfaces

```
ifplugstatus -a
```

Lists link status for every detected interface.

3. Use verbose mode for detailed results

```
ifplugstatus -v eth0
```

Provides more descriptive information about the link state.

4. Quiet mode for scripting or automation

```
ifplugstatus -q eth0
```

Outputs only the essential information, suitable for shell scripts.

Understanding the Output

A typical output may look like this:

```
eth0: link beat detected
enp0s3: link beat not detected
```

- Interface Name: eth0, enp0s3, etc.

- Link Status: Indicates whether a physical link (cable or signal) is present.

When integrated into scripts, this helps automate detection of disconnected interfaces or faulty cabling.

Common Use Cases

1. Cable Connectivity Testing

```
ifplugstatus eth0
```

Quickly verify if a cable is physically connected to the network port.

2. Multi-Interface Monitoring

```
ifplugstatus eth0 wlan0
```

Check multiple interfaces simultaneously for link activity.

3. Continuous Monitoring

```
watch -n 5 ifplugstatus eth0
```

Updates the interface status every 5 seconds for real-time link monitoring.

Troubleshooting Common Issues

1. Interface Not Found

If an incorrect interface name is given:

```
eth5: No such device
```

Fix: Use the ip a or ifconfig command to list valid interfaces.

2. Permission Denied

Some systems may require elevated privileges:

```
sudo ifplugstatus eth0
```

3. Command Not Found

If the command is missing:

```
ifplugstatus: command not found
```

Fix: Install the ifplugd package using your package manager.

Tip: Always ensure the ifplugd package is installed and your user has permission to access network interfaces.

Related Commands

932

Command	Description
`ip a` (preferred) / `ifconfig`	Display all network interfaces and details. `ip a` is the modern tool; `ifconfig` is legacy.
`ethtool eth0`	Retrieve advanced information about the network interface.
`nmcli device status`	Check connection status via NetworkManager.
`ping`	Test network reachability after confirming cable connection.

Important Notes

933

- Works best with wired Ethernet interfaces.

- Wireless interfaces may not always report link status accurately.

- The tool is read-only — it does not modify system settings.

- Lightweight and safe for use in both desktop and server environments.

Manual Reference

934

For additional information and advanced usage:

```
man ifplugstatus
```

Conclusion 935

Congratulations! You've reached the end of the **101 Linux Commands eBook**. Throughout this journey, you've explored over 135 essential Linux commands that form the foundation of system administration, development, and everyday Linux usage.

What You've Learned

This eBook has covered a comprehensive range of topics including:

🗂 File and Directory Management

- Navigation commands (`cd`, `ls`, `pwd`)
- File manipulation (`cp`, `mv`, `rm`, `mkdir`)
- Content viewing (`cat`, `head`, `tail`, `less`)
- Search and find operations (`find`, `grep`, `locate`)

⚙ System Administration

- Process management (`ps`, `kill`, `top`, `htop`)
- User and group management (`useradd`, `usermod`, `chmod`, `chown`)
- System monitoring (`df`, `du`, `free`, `vmstat`)
- Service management (`systemctl`, `service`)

🌐 Networking and Communication

- Network configuration (`ip`, `ifconfig`, `netstat`)
- Remote access (`ssh`, `scp`, `rsync`)
- Network diagnostics (`ping`, `dig`, `whois`)

📦 Package Management and Archives

- Package managers (`apt`, `yum`, `rpm`)
- Compression tools (`tar`, `gzip`, `zip`)
- Archive manipulation

⚙ **Text Processing and Automation**

- Text editors (`vim`, `nano`)
- Text processing (`awk`, `sed`, `cut`, `sort`)
- Task scheduling (`crontab`)
- Shell scripting helpers (`xargs`, `nohup`)

Your Linux Journey Continues

Mastering these commands is just the beginning of your Linux journey. Here are some recommendations for continuing your learning:

Next Steps

1. **Practice Regularly**: The best way to master Linux commands is through consistent practice
2. **Explore System Administration**: Learn about server management, security, and deployment
3. **Dive into Scripting**: Start writing bash scripts to automate repetitive tasks
4. **Learn Version Control**: Master Git for code management and collaboration
5. **Explore Containerization**: Learn Docker and Kubernetes for modern application deployment

Advanced Topics to Explore

- **Shell Scripting**: Write complex automation scripts
- **System Security**: Learn about firewalls, SSL certificates, and security hardening
- **Performance Tuning**: Optimize system performance and troubleshoot issues
- **DevOps Tools**: Explore CI/CD pipelines, configuration management, and infrastructure as code
- **Cloud Computing**: Learn about AWS, Azure, Google Cloud, and cloud-native technologies

Additional Resources

- **Man Pages**: Use `man command` to get detailed information about any command
- **Linux Documentation Project**: Comprehensive guides and HOWTOs
- **Online Communities**: Join forums, Discord servers, and Reddit communities
- **Practice Environments**: Set up virtual machines or use cloud platforms for hands-on learning

Contributing to This Project

This eBook is open source and community-driven. We welcome contributions from developers and Linux enthusiasts around the world:

🤝 How to Contribute

- **Add New Commands**: Help us reach our goal of covering even more Linux commands
- **Improve Examples**: Add better examples or use cases for existing commands
- **Fix Issues**: Report bugs, typos, or outdated information
- **Translations**: Help translate this eBook into other languages
- **Share Knowledge**: Add tips, tricks, and best practices

Visit our GitHub repository to get started with contributing.

Acknowledgments

941

⬛ Special Thanks

- **Mohamed Said**: Creator of the original Ibis tool
- **Roberto Butti**: Maintainer of Ibis Next used to generate this eBook
- **The Hacktoberfest Community**: Contributors who helped expand and improve this resource
- **Open Source Community**: Everyone who has contributed examples, fixes, and improvements

⬛ Design and Tools

- **Cover Design**: Created by Suhail Kakar
- **eBook Generation**: Powered by Ibis Next
- **Hosting**: Thanks to DevDojo for hosting and support

Final Words

942

Linux is more than just an operating system—it's a philosophy of open collaboration, continuous learning, and technological empowerment. The commands you've learned in this eBook are tools that will serve you throughout your career in technology.

Whether you're a system administrator managing servers, a developer building applications, a DevOps engineer automating deployments, or simply a curious learner exploring technology, these Linux commands will be invaluable companions on your journey.

Remember: **The terminal is your friend**. Don't be afraid to experiment, make mistakes, and learn from them. Every Linux expert started exactly where you are now.

⬡ Keep Learning, Keep Growing

The world of Linux is vast and constantly evolving. Stay curious, keep practicing, and most importantly, have fun exploring the incredible power and flexibility that Linux offers.

Happy Linux-ing! ⬡

This eBook was generated using Ibis Next - a modern, multi-format eBook generation tool for Markdown content.

Download Other Formats

This eBook is available in multiple formats:

- **PDF**: Perfect for offline reading and printing
- **EPUB**: Compatible with e-readers and mobile devices
- **HTML**: Ideal for web browsing and sharing

Visit our <u>GitHub releases</u> to download your preferred format.