Web Hosting

Top 100 Linux Commands (You Need To Know)

December 18, 2023 by Matt Stamp

Linux is the backbone of the internet. It powers nearly 97% of the world's top web servers. And 55.9% of professional developers lean on Linux for their development needs.

Yet, Linux has only a 2.68% desktop market share. Why this gap?

The core focus of Linux has never been its user interface. It was instead designed to give you complete control over your operating system through the command line.

That can make Linux seem intimidating to beginners — And the thousands of available commands only make this more difficult.

In this article, we cover the top 100 most useful Linux commands. Learning just a handful of these commands can help you boost your productivity as a Linux user. Let's dive right in!

DreamHost Glossary

Linux

Linux refers to a collection of open-source Operating Systems (OS). There's no single Linux OS. Instead, users can choose from a broad group of Linux distros, all of which provide different experiences.

Read More

What Are Linux Commands?

Linux commands allow you to control your system from the command line interface (CLI) instead of using your mouse or trackpad. They are text instructions entered into the terminal to tell your system exactly what to do.

Commands you enter on the Linux terminal are case-sensitive and follow a syntax like "`command -options arguments`." You can combine them for complex tasks using pipelines and redirection.

Some key things to know about Linux commands:

- **They are case-sensitive**; for example, "`ls`" and "LS" mean different things.
- **They follow a specific syntax** like "`command -options arguments`."
- **They can be combined** for complex operations using pipelines and redirection.
- **They give you fine-grained control** over your system, which is hard to achieve with graphical interfaces.
- **They allow you to automate tasks** through shell scripts and batch processing.
- **They can be used to access system resources** like the file system, network, memory, and CPU.
- **They form the basis of interaction with Linux** servers and operating systems.

If you're a programmer that's just learning to code, you can start practicing your Linux commands without leaving Windows using the Windows Subsystem for Linux. This lets you run Linux from within Windows without dual booting and get the best of both operating systems.

Top 100 Most Useful Linux Commands

Now that you have a basic understanding of what Linux commands are, let's dive into the top 100 most commonly used Linux commands.

We've organized them by category to cover areas like file management, system monitoring, network operations, user administration, and more.

File Management Commands In Linux

File management is a common task on the Linux command line. Here are essential file commands:

The `ls` command is one of the most frequently used Linux commands. It lists the contents of a directory, showing all files and subdirectories contained inside.

Without any options or arguments, `ls` will display the contents of the current working directory. You can pass a path name to list files and folders in that location instead.

Syntax:

```
ls [options] [directory]
```

Some of the most useful `ls` options include:

- `-l` – Display results in long format, showing extra details like permissions, ownership, size, and modification date for each file and directory.
- `-a` – Show hidden files and directories that start with . in addition to non-hidden items.
- `-R` – Recursively list all subdirectory contents, descending into child folders indefinitely.
- `-S` – Sort results by file size, largest first.
- `-t` – Sort by timestamp, newest first.

Example:

```
ls -l /home/user/documents
```

This would list the contents of the "**documents**" folder in long format.

Example output:

```
total 824
-rwxrwx--- 1 user user    8389 Jul 12 08:53 report.pdf
-rw-r--r-- 1 user user   10231 Jun 30 16:32
presentation.pptx
drwxr-xr-x 2 user user    4096 May 11 09:21 images
-rw-rw-r-- 1 user user     453 Apr 18 13:32 todo.txt
```

This output shows a detailed list with permissions, size, owner, and timestamp for each file and directory. The long listing format given by the `-l` option provides helpful file information at a glance.

The `ls` command gives you flexible control over directory content listing. It's one of the commands you'll find yourself using constantly when working on Linux.

2. cd – Change Directory

The `cd` command is used to navigate between directories. It allows you to move the current working directory to a new location in the filesystem.

When you run the `cd` command by itself, it will return you to the home directory. You can also pass a specific path to change into. For example:

- `cd /usr/local` – Changes to the /usr/local directory.
- `cd ..` – Moves up one level to the parent directory.
- `cd ~/pictures` – Changes to the pictures folder in your home directory.

Syntax:

```
cd [directory]
```

Example:

```
cd /home/user/documents
```

This would change the working directory to the "documents" folder under /home/user. Using `cd` is essential for being able to access and work with files in different locations conveniently.

3. mkdir – Create A New Directory

The `mkdir` command allows you to create a new folder. You simply pass the name of the directory to create.

Syntax:

```
mkdir [options] <directory>
```

This will create a directory called "newproject" in the current working directory.

Some useful `mkdir` options:

- `-p` – Creates parent directories recursively as needed.
- `-v` – Verbose output showing created directories.

Example:

```
mkdir -v ~/project/code
```

This would create the "code" subdirectory under "project" in the user's home folder, with verbose output showing the directory being created.

4. rmdir – Remove Directory

To delete an empty directory, use the `rmdir` command. Note that `rmdir` can only remove empty directories – we'll need the rm command to delete non-empty ones.

Syntax:

Some options for rmdir include:

- `-v` – Verbose output when deleting directories.
- `-p` – Remove parent directories recursively as needed.

Example:

```
rmdir -v ~/project/code
```

This would delete the "code" subdirectory under "project" while showing verbose output.

5. touch – Create A New Empty File

The `touch` command is used to create a new empty file instantly. This is useful when you need an empty file to populate with data later.

The basic syntax of touch is:

```
touch [options] filename
```

Some useful options for touch include:

- `-c` – Do not create the file if it already exists. This avoids accidentally overwriting existing files.
- `-m` – Instead of creating a new file, update the timestamp on an existing file. This can be used to change the modified time.

For example:

```
touch /home/user/newfile.txt
```

The above command creates a new empty file called "newfile.txt" in the user's /home/user directory. If newfile.txt already exists, it will update the access and modification times on the file instead.

6. cp – Copy Files And Directories

The basic syntax of cp is:

```
cp [options] source destination
```

Some useful cp options:

- `-r` – Copy directories recursively, descending into child directories to copy their contents as well. Necessary when copying directories.
- `-i` – Prompt before overwriting any existing files at the destination. It prevents accidentally overwriting data.
- `-v` – Display verbose output showing the details of each file as it is copied. Helpful to confirm exactly what was copied.

For example:

```
cp -r /home/user/documents /backups/
```

This would recursively copy the /home/user/documents directory and all its contents to the /backups/ directory. The `-r` option is needed to copy directories.

The `cp` command is one of the most frequently used file management utilities for copying files and directories in Linux. You'll find yourself using this command quite often.

7. mv – Move Or Rename Files And Directories

The `mv` command is used to move files or directories to a different location or rename them. Unlike copy, the files from the source path are deleted after they've been moved to the destination.

You can also use the `mv` command to rename files since you simply need to change the source and destination paths to the old and new name.

The syntax of mv is:

Useful mv options:

- `-i` – Prompt before overwriting any existing files at the destination location. This prevents accidentally overwriting data.
- `-v` – Produce verbose output showing each file or directory as it is moved. This is helpful for confirming exactly what was moved.

For example:

```
mv ~/folder1 /tmp/folder1
```

The above will move folder1 from the home (~) directory to the /tmp/ directory. Let's look at another example of using the `mv` command for renaming files.

```
mv folder1 folder2
```

Here, "folder1" is renamed to "folder2."

8. rm – Remove Files And Directories

The `rm` command deletes files and directories. Use caution because deleted files and directories cannot be recovered.

The syntax is:

```
rm [options] name
```

Useful rm options:

- `-r` – Recursively delete directories, including all contents inside them. This is necessary when deleting directories.
- `-f` – Force deletion and suppress all confirmation prompts. This is a dangerous command, as files cannot be recovered when they're gone!
- `-i` – Prompt for confirmation before deleting each file or directory, which provides safety against accidental removal.

For example:

This recursively deletes the "temp" directory and all its contents without prompting (`-f` overrides confirmations).

Note: The `rm` command permanently erases files and folders, so use it with extreme care. If used with sudo privileges, you could also delete the root directory completely, and Linux would no longer function after restarting your computer.

9. find – Search For Files In A Directory Hierarchy

The `find` command recursively searches directories for files matching given criteria.

The basic syntax of find is:

```
find [path] [criteria]
```

Some useful criteria options for find include:

- `-type f` – Search for only normal files, omitting directories.
- `-mtime +30` – Search for files modified over 30 days ago.
- `-user jane` – Search for files belonging to user "jane."

For example:

```
find . -type f -mtime +30
```

This will find all regular files over 30 days old under the current directory (*denoted by the dot*).

The find command allows searching for files based on all kinds of advanced conditions like name, size, permissions, timestamps, ownership, and more.

10. du – Estimate File Space Usage

The `du` command measures the file space usage for a given directory. When used without options, it displays disk usage for the current working directory.

```
du [options] [path]
```

Useful du options:

- `-h` – Display file sizes in human-readable format like K for Kilobytes rather than a byte count. Much easier to parse.
- `-s` – Only show the total size for a directory, rather than listing each subdirectory and file. Good for summary.
- `-a` – Show individual file sizes in addition to totals. Helps identify large files.

For example:

```
du -sh pictures
```

This will print a human-readable size total for the "pictures" directory.

The `du` command is helpful for analyzing disk usage for a directory tree and identifying files consuming excessive space.

Search And Filter Commands In Linux

Now, let's explore commands that allow you to search, filter, and manipulate text right from the Linux command line.

11. grep – Search Text Using Patterns

The `grep` command is used to search for text patterns within files or output. It prints any lines that match the given regular expression. `grep` is extremely powerful for searching, filtering, and pattern matching in Linux.

Here is the basic syntax:

```
grep [options] pattern [files]
```

For example:

```
grep -i "error" /var/log/syslog
```

Some useful grep options:

- `-i` – Ignore case distinctions in patterns
- `-R` – Recursively search subdirectories
- `-c` – Print only a count of matching lines
- `-v` – Invert match, print non-matching lines

`grep` allows you to search files and output for keywords or patterns quickly. It's invaluable for parsing logs, searching source code, matching regexes, and extracting data.

12. awk – Pattern Scanning And Processing Language

The awk command allows more advanced text processing based on specified patterns and actions. It operates on a line-by-line basis, splitting each line into fields.

awk syntax is:

```
awk 'pattern { action }' input-file
```

For example:

```
awk '/error/ {print $1}' /var/log/syslog
```

This prints the first field of any line containing "error." awk can also use built-in variables like NR (*number of records*) and NF (*number of fields*).

Advanced awk capabilities include:

- Mathematical computations on fields
- Conditional statements
- Built-in functions for manipulating strings, numbers, and dates
- Output formatting control

This makes awk suitable for data extraction, reporting, and transforming text output. awk is extremely powerful since it is an independent programming

13. sed – Stream Editor For Filtering And Transforming Text

The `sed` command allows filtering and transformation of text. It can perform operations like search/replace, deletion, transposition, and more. However, unlike `awk`, `sed` was designed for editing lines on a per-line basis as per the instructions.

Here's the basic syntax is:

```
sed options 'commands' input-file
```

For example:

```
sed 's/foo/bar/' file.txt
```

This replaces "foo" with "bar" in file.txt.

Some useful sed commands:

- `s` – Search and replace text
- `/pattern/d` – Delete lines matching a pattern
- `10,20d` – Delete lines 10-20
- `1,3!d` – Delete all except lines 1-3

`sed` is ideal for tasks like bulk find/replace, selective line deletion, and other text stream editing operations.

14. sort – Sort Lines Of Text Files

When you're working with a lot of text or data or even large outputs from other commands, sorting it is a great way to make things manageable. The `sort` command will sort the lines of a text file alphabetically or numerically.

Basic sort syntax:

```
sort [options] [file]
```

- `-n` – Sort numerically instead of alphabetically
- `-r` – Reverse the sort order
- `-k` – Sort based on a specific field or column

For example:

```
sort -n grades.txt
```

This numerically sorts the contents of grades.txt. `sort` is handy for ordering the contents of files for more readable output or analysis.

15. uniq – Report Or Omit Repeated Lines

The `uniq` command filters duplicate adjacent lines from input. This is often used in conjunction with sort.

Basic syntax:

```
uniq [options] [input]
```

Options:

- `-c` – Prefix unique lines with count of occurrences.
- `-d` – Only show duplicated lines, not unique ones.

For example:

```
sort data.txt | uniq
```

This will remove any duplicated lines in data.txt after sorting. `uniq` gives you control over filtering repeated text.

16. diff – Compare Files Line By Line

The `diff` command compares two files line-by-line and prints the differences. It's commonly used to show changes between versions of files.

Syntax:

Options:

- `-b` – Ignore changes in whitespace.
- `-B` – Show differences inline, highlighting changes.
- `-u` – Output differences with three lines of context.

For example:

```
diff original.txt updated.txt
```

This will output the lines that differ between original.txt and updated.txt. `diff` is invaluable for comparing revisions of text files and source code.

17. wc – Print Line, Word, And Byte Counts

The `wc` (word count) command prints counts of lines, words, and bytes in a file.

Syntax:

```
wc [options] [file]
```

Options:

- `-l` – Print only the line count.
- `-w` – Print only the word count.
- `-c` – Print only the byte count.

For example:

```
wc report.txt
```

This command will print the number of lines, words, and bytes in report.txt.

Redirection Commands In Linux

Redirection commands are used to control input and output sources in Linux, allowing you to send and append output streams to files, take input from files, connect multiple commands, and split output to multiple destinations.

The > redirection operator redirects the standard output stream from the command to a file instead of printing to the terminal. Any existing contents of the file will be overwritten.

For example:

```
ls -l /home > homelist.txt
```

This will execute **ls -l** to list the contents of the /home directory.

Then, instead of printing that output to the terminal, the **>** symbol captures that standard output and writes it to homelist.txt, overwriting any existing file contents.

Redirecting standard output is helpful for saving command results to files for storage, debugging, or chaining commands together.

19. >> – Append Standard Output

The >> operator appends standard output from a command to a file without overwriting existing contents.

For example:

```
tail /var/log/syslog >> logfile.txt
```

This will append the last 10 lines of the syslog log file onto the end of logfile.txt. Unlike >, **>>** adds the output without erasing the current logfile.txt contents.

Appending is helpful in collecting command output in one place without losing existing data.

20. < – Redirect Standard Input

The < redirection operator feeds a file's contents as standard input to a command, instead of taking input from the keyboard.

For example:

```
wc -l < myfile.txt
```

This sends the contents of myfile.txt as input to the wc command, which will count lines in that file instead of waiting for keyboard input.

Redirecting input is useful for batch-processing files and automating workflows.

21. | – Pipe Output To Another Command

The pipe `|` operator sends the output from one command as input to another command, chaining them together.

For example:

```
ls -l | less
```

This pipes the output of `ls -l` into the less command, which allows scrolling through the file listing.

Piping is commonly used to chain together commands where the output of one feeds the input of another. This allows building complex operations out of smaller single-purpose programs.

22. tee – Read From Standard Input And Write To Standard Output And Files

The `tee` command splits standard input into two streams.

It writes the input to standard output (shows the output of the main command) while also saving a copy to a file.

For example:

```
cat file.txt | tee copy.txt
```

This displays file.txt contents to the terminal while simultaneously writing it to copy.txt.

you've redirected the output to.

Archive Commands

Archiving commands allow you to bundle multiple files and directories into compressed archive files for easier portability and storage. Common archive formats in Linux include `.tar`, `.gz`, and `.zip`.

23. tar – Store And Extract Files From An Archive

The `tar` command helps you work with tape archive (.tar) files. It helps you bundle multiple files and directories into a single compressed .tar file.

Syntax:

```
tar [options] filename
```

Useful tar options:

- `-c` – Create a new .tar archive file.
- `-x` – Extract files from a .tar archive.
- `-f` – Specify archive filename rather than stdin/stdout.
- `-v` – Verbose output showing archived files.
- `-z` – Compress or uncompress archive with gzip.

For example:

```
tar -cvzf images.tar.gz /home/user/images
```

This creates a gzip-compressed tar archive called images.tar.gz containing the /home/user/images folder.

24. gzip – Compress Or Expand Files

The gzip command compresses files using LZ77 coding to reduce size for storage or transmission. With gzip, you work with .gz files.

Syntax:

Useful gzip options:

- `-c` – Write output to stdout instead of file.
- `-d` – Decompress file instead of compressing.
- `-r` – Recursively compress directories.

For example:

```
gzip -cr documents/
```

The above command recursively compresses the documents folder and outputs to stdout.

25. gunzip – Decompress Files

The `gunzip` command is used for decompressing .gz files.

Syntax:

```
gunzip filename.gz
```

Example:

```
gunzip documents.tar.gz
```

The above command will extract the original uncompressed contents of documents.tar.gz.

26. zip – Package And Compress Files

The `zip` command creates .zip archived files containing compressed file contents.

Syntax:

```
zip [options] archive.zip filenames
```

Useful zip options:

- `-e` – Encrypt contents with a password.

Example:

```
zip -re images.zip pictures
```

This encrypts and compresses the pictures folder into images.zip.

27. unzip – Extract Files From ZIP Archives

Similar to `gunzip`, the `unzip` command extracts and uncompresses files from .zip archives.

Syntax:

```
unzip archive.zip
```

Example:

```
unzip images.zip
```

The above example command extracts all files from images.zip in the current directory.

File Transfer Commands

File transfer commands allow you to move files between systems over a network. This is useful for copying files to remote servers or downloading content from the internet.

28. scp – Secure Copy Files Between Hosts

The `scp` (secure copy) command copies files between hosts over an SSH connection. All data transfer is encrypted for security.

scp syntax copies files from a source path to a destination defined as user@host:

```
scp source user@host:destination
```

For example:

```
scp image.jpg user@server:/uploads/
```

This securely copies image.jpg to the /uploads folder on **server** as **user**.

`scp` works like the `cp` command but for remote file transfer. It leverages SSH (Secure Shell) for data transfer, providing encryption to ensure that no sensitive data, such as passwords, are exposed over the network. Authentication is typically handled using SSH keys, though passwords can also be used. Files can be copied both to and from remote hosts.

29. rsync – Synchronize Files Between Hosts

The `rsync` tool synchronizes files between two locations while minimizing data transfer using delta encoding. This makes it faster to sync large directory trees.

rsync syntax syncs source to destination:

```
rsync [options] source destination
```

For example:

```
rsync -ahv ~/documents user@server:/backups/
```

The above example command recursively syncs the documents folder to server:/backups/, showing verbose, human-readable output.

Useful rsync options:

- `-a` – Archive mode syncs recursively and preserves permissions, times, etc.
- `-h` – Human-readable output.
- `-v` – Verbose output.

`rsync` is ideal for syncing files and folders to remote systems and keeping things decentrally backed up and secure.

30. sftp – Secure File Transfer Program

FTP but encrypted. It can transfer files to/from remote systems.

sftp connects to a host then accepts commands like:

```
sftp user@host

get remotefile localfile

put localfile remotefile
```

This retrieves `remotefile` from the server and copies `localfile` to the remote host.

`sftp` has an interactive shell for navigating remote file systems, transferring files and directories, and managing permissions and properties.

31. wget – Retrieve Files from the Web

The `wget` tool downloads files over HTTP, HTTPS, and FTP connections. It's useful for retrieving web resources directly from the terminal.

For example:

```
wget https://example.com/file.iso
```

This downloads the file.iso image from the remote server.

Useful wget options:

- `-c` – Resume interrupted download.
- `-r` – Download recursively.
- `-O` – Save to specific filename.

`wget` is ideal for scripting automatic downloads and mirroring websites.

32. curl – Transfer Data From Or To A Server

The `curl` command transfers data to or from a network server using supported protocols. This includes REST, HTTP, FTP, and more.

```
curl -L https://example.com
```

The above command retrieves data from the HTTPS URL and outputs it.

Useful curl options:

- `-o` – Write output to file.
- `-I` – Show response headers only.
- `-L` – Follow redirects.

`curl` is designed to transfer data across networks programmatically.

File Permissions Commands

File permissions commands allow you to modify access rights for users. This includes setting read/write/execute permissions, changing ownership, and default file modes.

33. chmod – Change File Modes Or Access Permissions

The `chmod` command is used to change the access permissions or modes of files and directories. The permission modes represent who can read, write, or execute the file.

For example:

```
chmod 755 file.txt
```

There are three sets of permissions—owner, group, and public. **Permissions are set using numeric modes from 0 to 7:**

- 7 – read, write, and execute.
- 6 – read and write.
- 4 – read only.
- 0 – no permission.

This sets the owner permissions to 7 (rwx), group to 5 (r-x), and public to 5 (r-x).

```
chmod g+w file.txt
```

The g+w syntax adds group write permission to the file.

Setting proper file and directory permissions is crucial for Linux security and controlling access. chmod gives you flexible control to configure permissions precisely as needed.

34. chown – Change File Owner And Group

The chown command changes ownership of a file or directory. Ownership has two components—the user who is the owner, and the group it belongs to.

For example:

```
chown john:developers file.txt
```

The above example command will set the owner user to "john" and the owner group to "developers."

Only the root superuser account can use chown to change file owners. It's used to fix permission problems by modifying the owner and group as needed.

35. umask – Set Default File Permissions

The umask command controls the default permissions given to newly created files. It takes an octal mask as input, which subtracts from 666 for files and 777 for directories.

For example:

```
umask 007
```

New files will default to permissions 750 instead of 666, and new directories to 700 instead of 777.

Setting a umask lets you configure default file permissions rather than relying on system defaults. The umask command is useful for restricting permissions on

restrictions.

Process Management Commands

These commands allow you to view, monitor, and control processes running on your Linux system. This is useful for identifying resource usage and stopping misbehaving programs.

36. ps – Report A Snapshot Of Current Processes

The `ps` command displays a snapshot of currently running processes, including their PID, TTY, stat, start time, etc.

For example:

```
ps aux
```

This shows every process running as all users with additional details like CPU and memory usage.

Some useful ps options:

- `aux` – Show processes for all users
- `--forest` – Display tree of parent/child processes

`ps` gives you visibility into what's currently running on your system.

37. top – Display Linux Processes

The `top` command shows real-time Linux process information, including PID, user, CPU %, memory usage, uptime, and more. Unlike `ps`, it updates the display dynamically to reflect current usage.

For example:

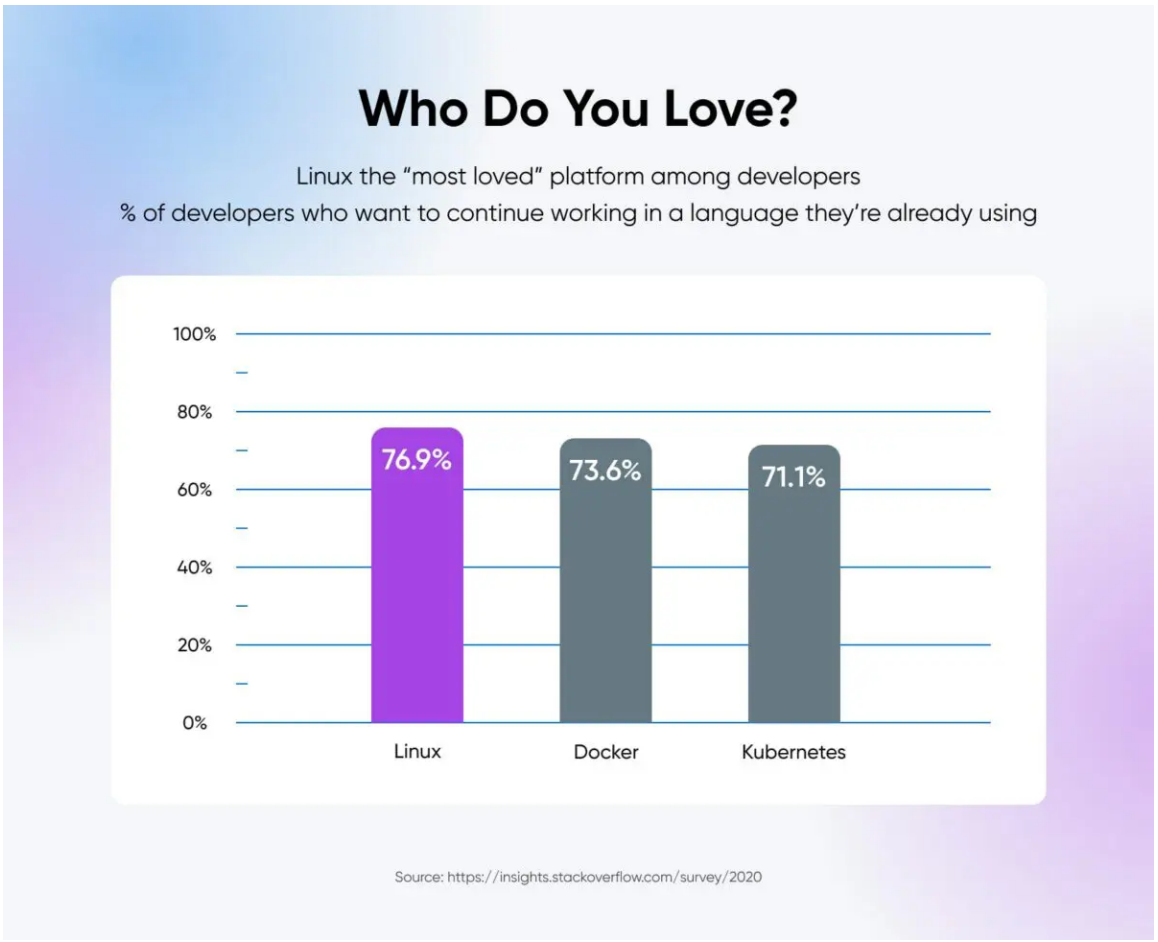
```
top -u mysql
```

The above command monitors processes just for the "mysql" user. It becomes quite helpful in identifying resource-intensive programs.

The `htop` command is an interactive process viewer replacing the top command. It shows system processes along with CPU/memory/swap usage graphs, allows sorting by columns, killing programs, and more.

Simply type in **`htop`** in the command line to view your processes.

`htop` has enhanced UI elements with colors, scrolling, and mouse support for easier navigation compared to **top**. Excellent for investigating processes.



39. kill – Send A Signal To A Process

The `kill` command sends a signal to a process to terminate or kill it. Signals allow graceful shutdown if the process handles them.

For example:

```
kill -15 12345
```

40. pkill – Send A Signal To A Process Based On Name

The `pkill` command kills processes by name instead of PID. It can make things easier than finding the PID first.

For example:

```
pkill -9 firefox
```

This forcibly stops all Firefox processes with SIGKILL (9). `pkill` targets processes by matching name, user, and other criteria instead of the PID.

41. nohup – Run A Command Immune To Hangups

The `nohup` command runs processes immune to hangups, so they keep running if you log out or get disconnected.

For example:

```
nohup python script.py &
```

The above example command will launch script.py detached in the background and immune to hangups. `nohup` is generally used to start persistent background daemons and services.

Performance Monitoring Commands

These commands provide valuable system performance statistics to help analyze resource utilization, identify bottlenecks, and optimize efficiency.

42. vmstat – Report Virtual Memory Statistics

The `vmstat` command prints detailed reports on memory, swap, I/O, and CPU activity. This includes metrics like memory used/free, swap in/out, disk blocks read/written, and CPU time spent on processes/idle.

For example:

Other useful vmstat options:

- `-a` – Show active and inactive memory
- `-s` – Display event counters and memory stats
- `-S` – Output in KB instead of blocks
- `5` – Output refreshed every 5 seconds.

The example above outputs memory and CPU data every 5 seconds until interrupted, which is useful for monitoring live system performance.

43. iostat – Report CPU And I/O Statistics

The `iostat` command monitors and displays CPU utilization and disk I/O metrics. This includes CPU load, IOPS, read/write throughput, and more.

For example:

```
iostat -d -p sda 5
```

Some iostat options:

- `-c` – Display CPU utilization info
- `-t` – Print timestamp for each report
- `-x` – Show extended stats like service times and wait counts
- `-d` – Show detailed stats per disk/partition instead of aggregate totals
- `-p` – Display stats for specific disk devices

This shows detailed per-device I/O stats for sda every 5 seconds.

`iostat` helps analyze disk subsystem performance and identify hardware bottlenecks.

44. free – Display Amount Of Free And Used Memory

The `free` command shows the total, used and free amounts of physical and swap memory on the system. This gives an overview of available memory.

For example:

```
free -h
```

Some options for the free command:

- `-b` – Display output in bytes
- `-k` – Show output in KB instead of default bytes
- `-m` – Show output in MB instead of bytes
- `-h` – Print statistics in human-readable format like GB, MB instead of bytes.

This prints memory statistics in human-readable format (GB, MB, etc). It's useful when you want a quick overview of memory capacity.

45. df – Report File System Disk Space Usage

The `df` command displays disk space usage for file systems. It shows the filesystem name, total/used/available space, and capacity.

For example:

```
df -h
```

The above command will print the disk utilization in a human-readable format. You can also run it without arguments to get the same data in block sizes.

46. sar – Collect And Report System Activity

The `sar` tool collects and logs system activity information on CPU, memory, I/O, network, and more over time. This data can be analyzed to identify performance issues.

For example:

```
sar -u 5 60
```

This samples CPU usage every 5 seconds for a duration of 60 samples.

`sar` provides detailed historical system performance data not available in real-time tools.

When using multi-user systems, you may need commands that help you manage users and groups for access control and permissions. Let's cover those commands here.

47. useradd – Create A New User

The `useradd` command creates a new user account and home directory. It sets the new user's UID, group, shell, and other defaults.

For example:

```
useradd -m john
```

Useful useradd options:

- `-m` – Create the user's home directory.
- `-g` – Specify the primary group instead of the default.
- `-s` – Set the user's login shell.

The above command will create a new user, "john," with a generated UID and home folder created at /home/john.

48. usermod – Modify A User Account

The `usermod` command modifies the settings of an existing user account. This can change the username, home dir, shell, group, expiry date, etc.

For example:

```
usermod -aG developers john
```

With this command, you add a user **john** to an additional group—"developers." The `-a` appends to the existing list of groups that the user is added to.

49. userdel – Delete A User Account

The `userdel` command deletes a user account, home directory, and mail spool.

```
userdel -rf john
```

Helpful userdel options:

- `-r` – Remove the user's home directory and mail spool.
- `-f` – Force deletion even if the user is still logged in.

This forces the removal of user "john," deleting associated files.

Specifying options like `-r` and `-f` with `userdel` ensures the user account is entirely deleted even if the user is logged in or has active processes.

50. groupadd – Add A Group

The `groupadd` command creates a new user group. Groups represent teams or roles for permissions purposes.

For example:

```
groupadd -r sysadmin
```

Useful groupadd options:

- `-r` – Create a system group used for core system functions.
- `-g` – Specify the new group's GID instead of using next available.

The above command creates a new "sysadmin" group with system privileges. When creating new groups, the `-r` or `-g` help set them up correctly.

51. passwd – Update User's Authentication Tokens

The `passwd` command sets or updates a user's authentication password/tokens. This allows changing your login password.

For example:

```
passwd john
```

This prompts user "john" to enter a new password interactively. If you've lost the

privileges and change the password using the same method.

Networking Commands

These commands are used for monitoring connections, troubleshooting network issues, routing, DNS lookups, and interface configuration.

52. ping – Send ICMP ECHO_REQUEST To Network Hosts

The `ping` command verifies connectivity to a remote host by sending ICMP echo request packets and listening for echo responses.

For example:

```
ping google.com
PING google.com (142.251.42.78): 56 data bytes
64 bytes from 142.251.42.78: icmp_seq=0 ttl=112 time=8.590
ms
64 bytes from 142.251.42.78: icmp_seq=1 ttl=112 time=12.486
ms
64 bytes from 142.251.42.78: icmp_seq=2 ttl=112 time=12.085
ms
64 bytes from 142.251.42.78: icmp_seq=3 ttl=112 time=10.866
ms
--- google.com ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 8.590/11.007/12.486/1.518
ms
```

Useful ping options:

- `-c [count]` – Limit packets sent.
- `-i [interval]` – Wait interval seconds between pings.

With the above command, you ping google.com and outputs round-trip stats indicating connectivity and latency. Generally, `ping` is used to check if a system you're trying to connect to is alive and connected to the network.

53. ifconfig – Configure Network Interfaces

The `ifconfig` command displays and configures network interface settings, including IP address, netmask, broadcast, MTU, and hardware MAC address.

For example:

```
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
inet6 fe80::a00:27ff:fe1e:ef1d  prefixlen 64  scopeid
0x20<link>
ether 08:00:27:1e:ef:1d  txqueuelen 1000  (Ethernet)
RX packets 23955654  bytes 16426961213 (15.3 GiB)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 12432322  bytes 8710937057 (8.1 GiB)
TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Running `ifconfig` with no other arguments will give you a list of all the network interfaces available for use, along with IP and additional network information. `ifconfig` can also be used to set addresses, enable/disable interfaces, and change options.

54. netstat – Network Statistics

The `netstat` command shows you the network connections, routing tables, interface stats, masquerade connections, and multicast memberships.

For example:

```
netstat -pt tcp
```

This command will output all the active TCP connections and the processes using them.

55. ss – Socket Statistics

The `ss` command dumps socket statistical information similar to netstat. It can

For example:

```
ss -t -a
```

This prints all open TCP sockets. More efficient than `netstat`.

56. traceroute – Trace Route To Host

The `traceroute` command prints the route packets take to a network host, showing each hop along the way and transit times. Useful for network debugging.

For example:

```
traceroute google.com
```

This traces the path to reach google.com and outputs each network hop.

57. dig - DNS Lookup

The `dig` command performs DNS lookups and returns information about DNS records for a domain.

For example:

```
dig google.com
; <<>> DiG 9.10.6 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60290
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0,
ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1280
;; QUESTION SECTION:
;google.com. IN A
:: ANSWER SECTION:
```

```
;; Query time: 6 msec
;; SERVER:
2405:201:2:e17b::c0a8:1d01#53(2405:201:2:e17b::c0a8:1d01)
;; WHEN: Wed Nov 15 01:36:16 IST 2023
;; MSG SIZE  rcvd: 55
```

This queries DNS servers for records related to google.com and prints details.

58. nslookup – Query Internet Name Servers Interactively

The `nslookup` command queries DNS servers interactively to perform name resolution lookups or display DNS records.

It enters an interactive shell, allowing you to manually lookup hostnames, reverse IP addresses, find DNS record types, and more.

For example, some common nslookup usage. Type nslookup on your command line:

```
nslookup
```

Next, we'll set Google's 8.8.8.8 DNS server for lookups.

```
> server 8.8.8.8
```

Now, let's query the A record of stackoverflow.com to find its IP address.

```
> set type=A
> stackoverflow.com
Server: 8.8.8.8
Address: 8.8.8.8#53
Non-authoritative answer:
Name: stackoverflow.com
Address: 104.18.32.7
Name: stackoverflow.com
Address: 172.64.155.249
```

```
> set type=MX
> github.com
Server: 8.8.8.8
Address: 8.8.8.8#53
Non-authoritative answer:
github.com mail exchanger = 1 aspmx.l.google.com.
github.com mail exchanger = 5 alt1.aspmx.l.google.com.
github.com mail exchanger = 5 alt2.aspmx.l.google.com.
github.com mail exchanger = 10 alt3.aspmx.l.google.com.
github.com mail exchanger = 10 alt4.aspmx.l.google.com.
```

The interactive queries make `nslookup` very useful for exploring DNS and troubleshooting name resolution issues.

59. iptables – IPv4 Packet Filtering And NAT

The `iptables` command allows configuring Linux netfilter firewall rules to filter and process network packets. It sets up policies and rules for how the system will handle different types of inbound and outbound connections and traffic.

For example:

```
iptables -A INPUT -s 192.168.1.10 -j DROP
```

The above command will block all input from IP 192.168.1.10.

`iptables` provides powerful control over the Linux kernel firewall to handle routing, NAT, packet filtering, and other traffic control. It is a critical tool for securing Linux servers.

60. ip – Manage Network Devices And Routing

The `ip` command allows managing and monitoring various network device related activities like assigning IP addresses, setting up subnets, displaying link details, and configuring routing options.

-

```
ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
fq_codel state UP mode DEFAULT group default qlen 1000
link/ether 08:00:27:8a:5c:04 brd ff:ff:ff:ff:ff:ff
```

The above command shows all the network interfaces, their status, and other information.

This command aims to replace ifconfig with more modern Linux network management. `ip` can control network devices, routing tables, and other network stack settings.

Package Management Commands

Package managers allow easy installation, update and removal of software on Linux distributions. Popular package managers include `APT`, `YUM`, `DNF`, `Pacman`, and `Zypper`.

61. apt – Debian/Ubuntu Package Manager

The `apt` command manages packages on Debian/Ubuntu systems using the APT repository. It allows installing, updating, and removing packages.

For example:

```
apt update
```

This command fetches the latest package versions and metadata from the repositories.

```
apt install nginx
```

You can install the nginx package from the configured APT repositories using the above command.

And this command upgrades packages and dependencies to newer versions.

APT makes installing software easy by fetching packages from repositories.

62. pacman – Arch Linux Package Manager

`pacman` manages packages on Arch Linux from the Arch User Repository. It can install, upgrade, and remove packages.

For example:

```
pacman -S nmap
```

This installs the nmap package from the configured repositories.

```
pacman -Syu
```

This synchronizes with repositories and upgrades all packages.

`pacman` keeps Arch Linux up-to-date and allows easy management of packages.

63. dnf – Fedora Package Manager

`dnf` installs, updates, and removes packages on Fedora Linux distributions using RPM packages. It replaces Yum as the next-gen package manager.

For example:

```
dnf install util-linux
```

This installs the util-linux package.

```
dnf upgrade
```

This upgrades all installed packages to the latest versions.

`dnf` makes Fedora package management fast and efficient.

64. yum – Red Hat Package Manager

`yum` manages packages on RHEL and CentOS Linux distributions using RPM packages. It fetches from Yum repositories to install and update.

For example:

```
yum update
```

This updates all installed packages to the latest versions.

```
yum install httpd
```

The above command installs the Apache httpd package. `yum` has been the major package manager for keeping Red Hat distributions updated.

65. zypper – OpenSUSE Package Manager

`zypper` manages packages on SUSE/openSUSE Linux. It can add repositories, search, install, and upgrade packages.

For example:

```
zypper refresh
```

The refresh command for `zypper` refreshes repository metadata from added repositories.

```
zypper install python
```

This installs the Python package from configured repositories. `zypper` makes the package management experience effortless on SUSE/openSUSE systems.

66. flatpak – Flatpak Application Package Manager

The `flatpak` command helps you manage Flatpak applications and runtimes. `flatpak` allows sandboxed desktop application distribution across Linux.

For example:

For instance, the above command will install LibreOffice from the Flathub repository.

```
flatpak run org.libreoffice.LibreOffice
```

And this one launches the sandboxed LibreOffice Flatpak application. `flatpak` provides a centralized cross-distro Linux application repository so you're no longer limited to packages available with a specific distro's package library.

67. appimage – AppImage Application Package Manager

`AppImage` packages are self-contained applications that run on most Linux distributions. The appimage command runs existing AppImages.

For example:

```
chmod +x myapp.AppImage
./myapp.AppImage
```

This allows running the AppImage binary file directly.

`AppImages` allow application deployment without system-wide installation. Think of them like small containers that include all the files to enable the app to run without too many external dependencies.

68. snap – Snappy Application Package Manager

The `snap` command manages snaps—containerized software packages. Snaps auto-update and work across Linux distributions similar to Flatpak.

For example:

```
snap install vlc
```

This simple command installs the VLC media player snap.

```
snap run vlc
```

by using the above command. Snaps isolate apps from the base system for portability and allow cleaner installs.

System Information Commands

These commands allow you to view details about your Linux system hardware, kernel, distributions, hostname, uptime, and more.

69. uname – Print System Information

The `uname` command prints detailed information about the Linux system kernel, hardware architecture, hostname, and operating system. This includes version numbers and machine info.

For example:

```
uname -a
Linux hostname 5.4.0-48-generic x86_64 GNU/Linux
```

`uname` is useful for querying these core system details. Some options include:

- `-a` – Print all available system info
- `-r` – Print just the kernel release number

The above command printed extended system information, including kernel name/version, hardware architecture, hostname, and OS.

```
uname -r
```

This will print only the kernel release number. The `uname` command shows details about your Linux system's core components.

70. hostname – Show Or Set The System's Host Name

The `hostname` command prints or sets the hostname identifier for your Linux system on the network. With no arguments it displays the current hostname. Passing a name will update the hostname.

```
hostname
linuxserver
```

This prints linuxserver — the configured system hostname.

```
hostname UbuntuServer
```

`hostnames` identify systems on a network. `hostname` gets or configures the identifying name of your system on the network. The second command helps you change the local hostname to UbuntuServer.

71. uptime – How Long The System Has Been Running

The `uptime` command shows how long the Linux system has been running since it was last rebooted. It prints the uptime and current time.

Simply run the following command to get your system uptime data:

```
uptime
23:51:26 up 2 days, 4:12, 1 user, load average: 0.00, 0.01,
0.05
```

This prints the system uptime showing how long the system has been on since last boot.

72. whoami – Print Active User ID

The `whoami` command prints the effective username of the current user logged into the system. It displays the privilege level you are operating at.

Type the command in your terminal to get the ID:

```
whoami
john
```

This prints the effective username that the current user is logged in and operating as and is useful in scripts or diagnostics to identify what user account actions are being performed as.

73. id – Print Real And Effective User And Group IDs

The `id` command prints detailed user and group information about the effective IDs and names of the current user. This includes:

- Real user ID and name.
- Effective user ID and name.
- Real group ID and name.
- Effective group ID and name.

To use the id command, simply type:

```
id
uid=1000(john) gid=1000(john)
groups=1000(john),10(wheel),998(developers)
```

The `id` command prints the current user's real and effective user and group IDs. `id` displays user and group details useful for determining file access permissions.

74. lscpu – Display CPU Architecture Information

The `lscpu` command shows detailed CPU architecture information, including:

- Number of CPU cores
- Number of sockets
- Model name
- Cache sizes
- CPU frequency
- Address sizes

To use the lscpu command, simply type:

```
lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              16
```

`lscpu` details the CPU architecture like the number of cores, sockets, model name, caches, and more.

75. lsblk – List Block Devices

The `lsblk` command lists information about all available block devices, including local disks, partitions, and logical volumes. The output includes device names, labels, sizes, and mount points.

```
lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
sda       8:0    0   1.8T  0 disk
|-sda1    8:1    0   512M  0 part  /boot
|-sda2    8:2    0    16M  0 part
`-sda5    8:5    0   1.8T  0 part
`-lvm1 254:0    0   1.8T  0 lvm   /
```

`lsblk` lists all the block devices, including disks, partitions, and logical volumes. Gives an overview of storage devices.

76. lsmod – Show The Status of Modules In The Linux Kernel

The `lsmod` command prints currently loaded kernel modules like device drivers. This includes networking, storage, and other hardware-related modules being used by the Linux kernel to interface with internal and external devices.

```
lsmod
Module                  Size  Used by
ipv6                  406206  27
evdev                  17700  0
crct10dif_pclmul       16384  1
crc32_pclmul           16384  0
ghash_clmulni_intel    16384  0
aesni_intel           399871  0
aes_x86_64             20274  1 aesni_intel
```

As you can see, it lists the currently loaded kernel modules like device drivers. In this case, it shows the use of networking, input, cryptographic and encryption modules.

77. dmesg – Print Or Control The Kernel Ring Buffer

The `dmesg` command dumps messages from the kernel ring buffer. This includes essential system events recorded by the kernel during start-up and operation.

```
dmesg | grep -i error
[   12.345678] Error receiving batched read response: -110
[   23.456789] tplink_mdio 0000:03:00.0: Direct firmware
load for tplink-mdio/leap_p8_v1_0.bin failed with error -2
[   40.567890] iwlwifi 0000:09:00.0: Direct firmware load for
iwlwifi-ty-a0-gf-a0-59.ucode failed with error -2
```

Grepping for "error" shows issues loading specific firmware. This prints buffered kernel log messages, including system events like start-up, errors, warnings etc.

System Administration Commands

System admin commands help you run programs as other users, shut down or reboot the system, and manage init systems and services.

78. sudo – Execute A Command As Another User

The `sudo` command allows you to run commands as another user, typically the superuser. After entering the `sudo` order, it will prompt you for your password to authenticate.

This provides elevated access for tasks like installing packages, editing system files, administering services etc.

For example:

```
sudo adduser bob
```

User 'bob' has been added to the system.

This uses `sudo` to create a new user, 'bob'. Regular users typically cannot add users without `sudo`.

79. su – Change User ID Or Become Superuser

The `su` command allows you to switch to another user account including the superuser. You must provide the target user's password to authenticate. This gives direct access to run commands in another user's environment.

For example:

```
su bob
Password:
bob@linux:~$
```

After inputting bob's password, this command switches the current user to the user 'bob'. The shell prompt will reflect the new user.

80. shutdown – Shutdown Or Restart Linux

The `shutdown` command schedules a system power off, halt or reboot after a specified timer or immediately. It's required to reboot or shutdown multi-user Linux systems safely.

For example:

```
shutdown -r now
Broadcast message from root@linux Fri 2023-01-20 18:12:37
CST:
The system is going down for reboot NOW!
```

This reboots the system instantly with a warning to users.

81. reboot – Reboot Or Restart System

The `reboot` command restarts the Linux operating system, logging all users off

down cleanly before restarting.

For example:

```
reboot
Restarting system.
```

This immediately reboots the OS. reboot is a simple alternative to shutdown `-r`.

82. systemctl – Control The systemd System And Service Manager

The `systemctl` command allows you to manage systemd services like starting, stopping, restarting, or reloading them. `Systemd` is the new init system used in most modern Linux distros, replacing SysV init.

For example:

```
systemctl start apache2
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-
units ===
Authentication is required to start 'apache2.service'.
Authenticating as: User Name
Password:
==== AUTHENTICATION COMPLETE ===
```

This starts the apache2 service after authenticating.

83. service – Run A System V Init Script

The `service` command runs System V init scripts for controlling services. This allows starting, stopping, restarting, and reloading services managed under traditional SysV init.

For example:

```
service iptables start
```

The above command starts the `iptables` firewall service using its SysV init script.

Other Linux Commands To Try

84. **`mount`** – Mount or "attach" drives to the system.
85. **`umount`** – Umount or "remove" drives from the system.
86. **`xargs`** – Builds and executes commands provided through standard input.
87. **`alias`** – Create shortcuts for long or complex commands.
88. **`jobs`** – List programs currently running jobs in the background.
89. **`bg`** – Resume a stopped or paused background process.
90. **`killall`** – Terminate processes by program name rather than PID.
91. **`history`** – Display previously used commands within the current terminal session.
92. **`man`** – Access help manuals for commands right within the terminal.
93. **`screen`** – Manage multiple terminal sessions from a single window.
94. **`ssh`** – Establish secure encrypted connections to remote servers.
95. **`tcpdump`** – Capture network traffic based on specific criteria.
96. **`watch`** – Repeat a command at intervals and highlight output differences.
97. **`tmux`** – Terminal multiplexer for persistent sessions and splitting.
98. **`nc`** – Open TCP or UDP connections for testing and data transfer.
99. **`nmap`** – Host discovery, port scanning, and OS fingerprinting.
100. **`strace`** – Debug processes by tracing operating system signals and calls.

7 Key Tips For Using Linux Commands

1. **Know your shell**: Bash, zsh, fish? Different shells have unique features. Pick the one that suits your needs the best.
2. **Master the core utils**: `ls`, `cat`, `grep`, `sed`, awk, etc form the core of a Linux toolkit.
3. **Stick with pipelines**: Avoid excessive uses of temporary files. Pipe programs together cleverly.
4. **Verify before overwriting**: Always double check before overwriting files with > and >>.
5. **Track your workflows**: Document complex commands and workflows to

6. **Make your own tools**: Write simple shell scripts and aliases for frequent tasks.
7. **Start without sudo**: Use a standard user account initially to understand permissions.

And remember to keep testing out new commands over virtual machines or VPS servers so they become second nature to you before you start using them on production servers.

DreamHost Glossary

VPS Hosting

A Virtual Private Server (VPS) is a virtual platform that stores data. Many web hosts offer VPS hosting plans, which give site owners a dedicated, private space on a shared server.

Read More

Better Linux Hosting With DreamHost

After you master the essential Linux commands, you also need a hosting and server provider that gives you full control to take advantage of Linux's power and flexibility.

That's where DreamHost shines.

DreamHost's optimized Linux infrastructure is perfect for running your apps, sites, and services:

- Fast web hosting on modern Linux servers.
- SSH shell access for command line control.
- Customizable PHP versions including PHP 8.0.

- Managed MySQL, PostgreSQL, Redis databases.
- 1-click installs of apps like WordPress and Drupal.
- SSD-accelerated NVMe storage for speed.
- Free Let's Encrypt SSL auto-renewal.

DreamHost's experts can help you get the most out of the Linux platform. Our servers are meticulously configured for security, performance, and reliability.

Launch your next project on a Linux hosting platform you can trust. Get started with robust, scalable hosting at DreamHost.com.

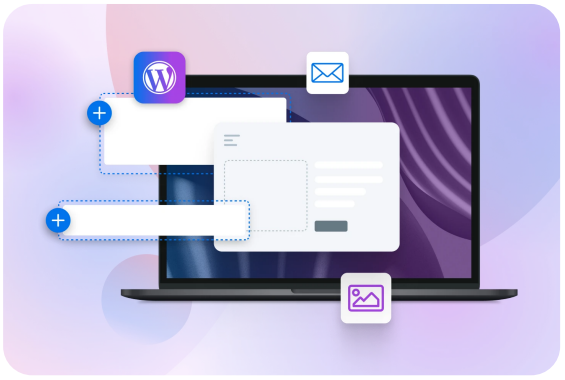
Matt Stamp

Matt is a DevOps Engineer at DreamHost. He is responsible for infrastructure automation, system monitoring and documentation. In his free time he enjoys 3D printing and camping. Follow Matt on LinkedIn: Lhttps://www.linkedin.com/in/matt-stamp-7a8b3a10a

Related Articles **Most Recent**

Work Smarter, Not Harder With These Game-Changing Chrome Extensions

September 11, 2024 By Bailey Abbott



Decoding WordPress: Working With Block Patterns

September 9, 2024 By Jason Cosper

Your Key to Building a

Map

September 6, 2024 By Alex Brown

Get Content Delivered Straight to Your Inbox

Subscribe to our blog and receive great content just like this delivered straight to your inbox.

Enter your email address

Sign Me Up!

Terms of Service

WHOIS Lookup

Webmail

RESOURCES

Contact

Knowledge Base

System Status

Glossary

Business Name Generator

Connect with us