Essential Linux Commands for Data Engineers

Ravi Sharma

Siddhesh Shinde

Introduction

Linux is the backbone of data engineering infrastructures

Provides powerful command-line tools for:

- Managing files and directories
- Automating tasks and scheduling jobs
- Monitoring system resources
- Operating on clusters and remote servers

This presentation covers essential Linux commands for Data Engineers, including:

- Syntax and usage examples
- Expected output and real-world use cases
- Best practices for efficiency and security

What is Linux??

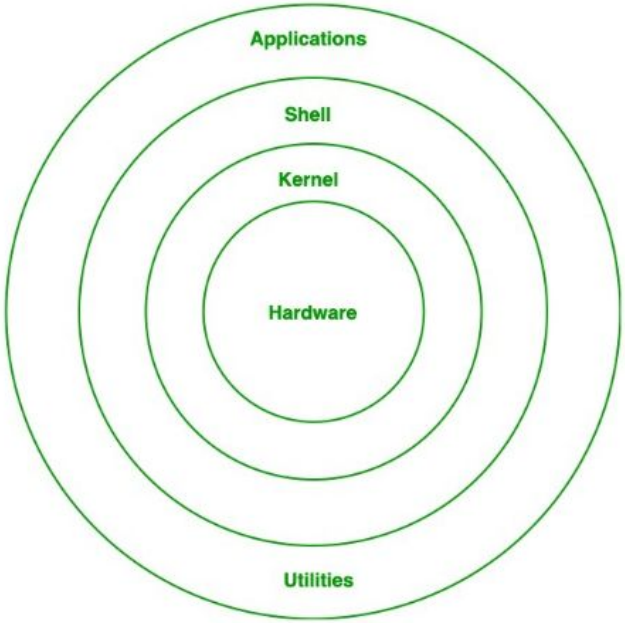
- Linux is a free and open-source operating system known for its resilience and flexibility.
- Created by Linus Torvalds in 1991, Linux allows anyone to access and modify its source code.
- Encourages global collaboration and continuous development.
- Used in a wide range of devices, from personal computers to supercomputers.
- Cost-effective, secure, and efficient, making it a preferred OS for many applications.

Linux Distributions

- A Linux distribution consists of the Linux kernel, supporting libraries, and software.
- Over 600+ distributions exist, catering to different user needs.
- Popular Linux distributions:
  - For Beginners: Ubuntu, Linux Mint
  - For Advanced Users: Arch Linux, Gentoo
  - For Developers: Fedora, Debian
  - For Servers: CentOS, Ubuntu Server
  - For Lightweight Systems: Lubuntu, Puppy Linux

Linux Architecture

- Kernel: Core of the OS, managing resources and process execution.
- System Library: Pre-written code enabling application functionality.
- Shell: User interface for command execution.
- Hardware Layer: Physical components such as RAM, CPU, and storage devices.
- System Utility: Tools for system management, networking, and software installation.



Linux Architecture

Advantages & Disadvantages

Advantages:

- Open-source: Free to use, modify, and distribute.
- Security: Less vulnerable to malware compared to other OSs.
- Stability: Rarely slows down or crashes.
- Performance: Efficient in handling multiple tasks and users.
- Privacy: Protects user data and security.
- Flexibility: Can install only required components.
- Community Support: Large and active developer community.

Disadvantages:

- Not very user-friendly for beginners.
- Limited peripheral hardware drivers compared to Windows.

Linux Important Commands

File and Directory Management

- ls - List directory contents
- cd - Change directory
- mkdir - Create a directory
- rm - Remove files or directories
- cp - Copy files or directories
- mv - Move or rename files or directories
- touch - Create an empty file
- find - Search for files and directories
- locate - Quickly find files using an indexed database
- tree - Display directories in a tree format

ls

> List directory contents. More information: https://www.gnu.org/software/coreutils/manual/html_node/ls-invocation.html.

- List files one per line:

```
ls -1
```

- List **a**ll files, including hidden files:

```
ls -a
```

- List files with a trailing symbol to indicate file type (directory/, symbolic_link@, executable*, ...):

```
ls -F
```

- List **a**ll files in **l**ong format (permissions, ownership, size, and modification date):

```
ls -la
```

- List files in **l**ong format with size displayed using **h**uman-readable units (KiB, MiB, GiB):

```
ls -lh
```

- List files in **l**ong format, sorted by [S]ize (descending) [R]ecursively:

```
ls -lSR
```

- List files in **l**ong format, sorted by **t**ime the file was modified and in **r**everse order (oldest first):

```
ls -ltr
```

- Only list **d**irectories:

```
ls -d */
```

cd

> Change the current working directory. More information: https://manned.org/cd.

- Go to the specified directory:

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

- Go up to the parent of the current directory:

```
cd ..
```

- Go to the home directory of the current user:

```
cd
```

- Go to the home directory of the specified user:

```
cd ~username
```

- Go to the previously chosen directory:

```
cd -
```

- Go to the root directory:

```
cd /
```

mkdir

> Create directories and set their permissions. More information: https://www.gnu.org/software/coreutils/manual/html_node/mkdir-invocation.html.

- Create specific directories:

```
mkdir path/to/directory1 path/to/directory2 ...
```

- Create specific directories and their parents if needed:

```
mkdir -p|--parents path/to/directory1 path/to/directory2 ...
```

- Create directories with specific permissions:

```
mkdir -m|--mode rwxrw-r-- path/to/directory1 path/to/directory2 ...
```

rm

> Remove files or directories. See also: `rmdir` . More information: https://www.gnu.org/software/coreutils/manual/html_node/rm-invocation.html.

- Remove specific files:

```
rm path/to/file1 path/to/file2 ...
```

- Remove specific files ignoring nonexistent ones:

```
rm -f path/to/file1 path/to/file2 ...
```

- Remove specific files interactively prompting before each removal:

```
rm -i path/to/file1 path/to/file2 ...
```

- Remove specific files printing info about each removal:

```
rm -v path/to/file1 path/to/file2 ...
```

- Remove specific files and directories recursively:

```
rm -r path/to/file_or_directory1 path/to/file_or_directory2 ...
```

cp

> Copy files and directories. More information: https://www.gnu.org/software/coreutils/manual/html_node/cp-invocation.html.

- Copy a file to another location:

```
cp path/to/source_file.ext path/to/target_file.ext
```

- Copy a file into another directory, keeping the filename:

```
cp path/to/source_file.ext path/to/target_parent_directory
```

- Recursively copy a directory's contents to another location (if the destination exists, the directory is copied inside it):

```
cp -R path/to/source_directory path/to/target_directory
```

- Copy a directory recursively, in verbose mode (shows files as they are copied):

```
cp -vR path/to/source_directory path/to/target_directory
```

- Copy multiple files at once to a directory:

```
cp -t path/to/destination_directory path/to/file1 path/to/file2 ...
```

- Copy text files to another location, in interactive mode (prompts user before overwriting):

```
cp -i *.txt path/to/target_directory
```

- Follow symbolic links before copying:

```
cp -L link path/to/target_directory
```

- Use the first argument as the destination directory (useful for `xargs ... | cp -t <DEST_DIR>`):

```
cp -t path/to/target_directory path/to/file_or_directory1 path/to/file_or_directory2 ...
```

mv

> Move or rename files and directories. More information: https://www.gnu.org/software/coreutils/manual/html_node/mv-invocation.html.

- Rename a file or directory when the target is not an existing directory:

```
mv path/to/source path/to/target
```

- Move a file or directory into an existing directory:

```
mv path/to/source path/to/existing_directory
```

- Move multiple files into an existing directory, keeping the filenames unchanged:

```
mv path/to/source1 path/to/source2 ... path/to/existing_directory
```

- Do not prompt (**f**) for confirmation before overwriting existing files:

```
mv --force path/to/source path/to/target
```

- Prompt for confirmation **i**nteractively before overwriting existing files, regardless of file permissions:

```
mv --interactive path/to/source path/to/target
```

- Do not overwrite (**n**) existing files at the target:

```
mv --no-clobber path/to/source path/to/target
```

- Move files in **v**erbose mode, showing files after they are moved:

```
mv --verbose path/to/source path/to/target
```

- Specify **t**arget directory so that you can use external tools to gather movable files:

```
find /var/log -type f -name '*.log' -print0 | xargs -0 mv --target-directory path/to/target_directory
```

touch

> Create files and set access/modification times. More information: https://manned.org/touch.

- Create specific files:

```
touch path/to/file1 path/to/file2 ...
```

- Set the file **a**ccess or **m**odification times to the current one and don't **c**reate file if it doesn't exist:

```
touch -c -a|m path/to/file1 path/to/file2 ...
```

- Set the file **t**ime to a specific value and don't **c**reate file if it doesn't exist:

```
touch -c -t YYYYMMDDHHMM.SS path/to/file1 path/to/file2 ...
```

- Set the files' timestamp to the **r**eference file's timestamp, and do not **c**reate the file if it does not exist:

```
touch -c -r path/to/reference_file path/to/file1 path/to/file2 ...
```

find

> Find files or directories under a directory tree, recursively. More information: https://manned.org/find.

- Find files by extension:

```
find root_path -name '*.ext'
```

- Find files matching multiple path/name patterns:

```
find root_path -path '*/path/*/*.ext' -or -name '*pattern*'
```

- Find directories matching a given name, in case-insensitive mode:

```
find root_path -type d -iname '*lib*'
```

- Find files matching a given pattern, excluding specific paths:

```
find root_path -name '*.py' -not -path '*/site-packages/*'
```

- Find files matching a given size range, limiting the recursive depth to "1":

```
find root_path -maxdepth 1 -size +500k -size -10M
```

- Run a command for each file (use `{}` within the command to access the filename):

```
find root_path -name '*.ext' -exec wc -l {} \;
```

- Find all files modified today and pass the results to a single command as arguments:

```
find root_path -daystart -mtime -1 -exec tar -cvf archive.tar {} \+
```

- Search for either empty files or directories and delete them verbosely:

```
find root_path -type f|d -empty -delete -print
```

locate

> Find filenames quickly. More information: https://manned.org/locate.

- Look for pattern in the database. Note: the database is recomputed periodically (usually weekly or daily):

```
locate pattern
```

- Look for a file by its exact filename (a pattern containing no globbing characters is interpreted as `*pattern*`):

```
locate '*/filename'
```

- Recompute the database. You need to do it if you want to find recently added files:

```
sudo updatedb
```

tree

> Show the contents of the current directory as a tree. More information: https://manned.org/tree.

- Print files and directories up to 'num' levels of depth (where 1 means the current directory):

```
tree -L num
```

- Print directories only:

```
tree -d
```

- Print hidden files too with colorization on:

```
tree -a -C
```

- Print the tree without indentation lines, showing the full path instead (use `-N` to not escape non-printable characters):

```
tree -i -f
```

- Print the size of each file and the cumulative size of each directory, in human-readable format:

```
tree -s -h --du
```

- Print files within the tree hierarchy, using a wildcard (glob) pattern, and pruning out directories that don't contain matching files:

```
tree -P '*.txt' --prune
```

- Print directories within the tree hierarchy, using the wildcard (glob) pattern, and pruning out directories that aren't ancestors of the wanted one:

```
tree -P directory_name --matchdirs --prune
```

- Print the tree ignoring the given directories:

```
tree -I 'directory_name1|directory_name2'
```

File Permissions & Ownership

- chmod - Change file permissions
- chown - Change file ownership
- umask - Set default file permissions
- lsattr - Display file attributes
- chattr - Change file attributes

chmod

> Change the access permissions of a file or directory. More information: https://www.gnu.org/software/coreutils/manual/html_node/chmod-invocation.html.

- Give the **u**ser who owns a file the right to e**x**ecute it:

```
chmod u+x path/to/file
```

- Give the **u**ser rights to **r**ead and **w**rite to a file/directory:

```
chmod u+rw path/to/file_or_directory
```

- Remove e**x**ecutable rights from the **g**roup:

```
chmod g-x path/to/file
```

- Give **a**ll users rights to **r**ead and e**x**ecute:

```
chmod a+rx path/to/file
```

- Give **o**thers (not in the file owner's group) the same rights as the **g**roup:

```
chmod o=g path/to/file
```

- Remove all rights from **o**thers:

```
chmod o= path/to/file
```

- Change permissions recursively giving **g**roup and **o**thers the ability to **w**rite:

```
chmod -R g+w,o+w path/to/directory
```

- Recursively give **a**ll users **r**ead permissions to files and e[X]ecute permissions to sub-directories within a directory:

```
chmod -R a+rX path/to/directory
```

chown

> Change user and group ownership of files and directories. More information: https://www.gnu.org/software/coreutils/manual/html_node/chown-invocation.html.

- Change the owner user of a file/directory:

```
chown user path/to/file_or_directory
```

- Change the owner user and group of a file/directory:

```
chown user:group path/to/file_or_directory
```

- Change the owner user and group to both have the name `user`:

```
chown user: path/to/file_or_directory
```

- Recursively change the owner of a directory and its contents:

```
chown -R user path/to/directory
```

- Change the owner of a symbolic link:

```
chown -h user path/to/symlink
```

- Change the owner of a file/directory to match a reference file:

```
chown --reference path/to/reference_file path/to/file_or_directory
```

umask

> Manage the read/write/execute permissions that are masked out (i.e. restricted) for newly created files by the user.
> More information: https://manned.org/umask.

- Display the current mask in octal notation:

```
umask
```

- Display the current mask in symbolic (human-readable) mode:

```
umask -S
```

- Change the mask symbolically to allow read permission for all users (the rest of the mask bits are unchanged):

```
umask a+r
```

- Set the mask (using octal) to restrict no permissions for the file's owner, and restrict all permissions for everyone else:

```
umask 077
```

lsattr

> List file attributes on a Linux filesystem. More information: https://manned.org/lsattr.

- Display the attributes of the files in the current directory:

```
lsattr
```

- List the attributes of files in a particular path:

```
lsattr path
```

- List file attributes recursively in the current and subsequent directories:

```
lsattr -R
```

- Show attributes of all the files in the current directory, including hidden ones:

```
lsattr -a
```

- Display attributes of directories in the current directory:

```
lsattr -d
```

chattr

> Change attributes of files or directories. More information: https://manned.org/chattr.

- Make a file or directory immutable to changes and deletion, even by superuser:

```
chattr +i path/to/file_or_directory
```

- Make a file or directory mutable:

```
chattr -i path/to/file_or_directory
```

- Recursively make an entire directory and contents immutable:

```
chattr -R +i path/to/directory
```

Text Processing & Manipulation

- cat - View file contents
- less - View large files page by page
- head - Display the first few lines of a file
- tail - Display the last few lines of a file
- grep - Search for text patterns in files
- awk - Process text by columns
- sed - Edit text in files
- cut - Extract specific columns from text files
- sort - Sort lines in a file
- uniq - Remove duplicate lines from a file
- diff - Compare two text files
- cmp - Compare binary files
- tr - Translate or delete characters
- wc - Count words, lines, and characters in a file

cat

> Print and concatenate files. More information: https://manned.org/cat.1posix.

- Print the contents of a file to `stdout`:

```
cat path/to/file
```

- Concatenate several files into an output file:

```
cat path/to/file1 path/to/file2 ... > path/to/output_file
```

- Append several files to an output file:

```
cat path/to/file1 path/to/file2 ... >> path/to/output_file
```

- Copy the contents of a file into an output file without buffering:

```
cat -u /dev/tty12 > /dev/tty13
```

- Write `stdin` to a file:

```
cat - > path/to/file
```

less

> Open a file for interactive reading, allowing scrolling and search. More information: https://greenwoodsoftware.com/less/.

- Open a file:

  `less source_file`

- Page down/up:

  `<Space> (down), b (up)`

- Go to end/start of file:

  `G (end), g (start)`

- Forward search for a string (press `n` / `N` to go to next/previous match):

  `/something`

- Backward search for a string (press `n` / `N` to go to next/previous match):

  `?something`

- Follow the output of the currently opened file:

  `F`

- Open the current file in an editor:

  `v`

- Exit:

  `q`

head

> Output the first part of files. More information: https://manned.org/head.1p.

- Output the first few lines of a file:

```
head -n count path/to/file
```

tail

> Display the last part of a file. See also: `head` . More information: https://www.gnu.org/software/coreutils/manual/html_node/tail-invocation.html.

- Show last 'count' lines in file:

```
tail --lines count path/to/file
```

- Print a file from a specific line number:

```
tail --lines +count path/to/file
```

- Print a specific count of bytes from the end of a given file:

```
tail --bytes count path/to/file
```

- Print the last lines of a given file and keep reading it until `Ctrl + C` :

```
tail --follow path/to/file
```

- Keep reading file until `Ctrl + C` , even if the file is inaccessible:

```
tail --retry --follow path/to/file
```

- Show last 'num' lines in 'file' and refresh every 'n' seconds:

```
tail --lines count --sleep-interval seconds --follow path/to/file
```

grep

> Find patterns in files using regular expressions. More information: https://www.gnu.org/software/grep/manual/grep.html.

- Search for a pattern within a file:

```
grep "search_pattern" path/to/file
```

- Search for an exact string (disables regular expressions):

```
grep -F|--fixed-strings "exact_string" path/to/file
```

- Search for a pattern in all files recursively in a directory, showing line numbers of matches, ignoring binary files:

```
grep -r|--recursive -n|--line-number --binary-files without-match "search_pattern" path/to/directory
```

- Use extended regular expressions (supports `?`, `+`, `{}`, `()`, and `|`), in case-insensitive mode:

```
grep -E|--extended-regexp -i|--ignore-case "search_pattern" path/to/file
```

- Print 3 lines of context around, before, or after each match:

```
grep --context|before-context|after-context 3 "search_pattern" path/to/file
```

- Print file name and line number for each match with color output:

```
grep -H|--with-filename -n|--line-number --color=always "search_pattern" path/to/file
```

- Search for lines matching a pattern, printing only the matched text:

```
grep -o|--only-matching "search_pattern" path/to/file
```

- Search `stdin` for lines that do not match a pattern:

```
cat path/to/file | grep -v|--invert-match "search_pattern"
```

awk

> A versatile programming language for working on files. More information: https://github.com/onetrueawk/awk.

- Print the fifth column (a.k.a. field) in a space-separated file:

```
awk '{print $5}' path/to/file
```

- Print the second column of the lines containing "foo" in a space-separated file:

```
awk '/foo/ {print $2}' path/to/file
```

- Print the last column of each line in a file, using a comma (instead of space) as a field separator:

```
awk -F ',' '{print $NF}' path/to/file
```

- Sum the values in the first column of a file and print the total:

```
awk '{s+=$1} END {print s}' path/to/file
```

- Print every third line starting from the first line:

```
awk 'NR%3==1' path/to/file
```

- Print different values based on conditions:

```
awk '{if ($1 == "foo") print "Exact match foo"; else if ($1 ~ "bar") print "Partial match bar"; else print
"Baz"}' path/to/file
```

- Print all the lines which the 10th column value is between a min and a max:

```
awk '($10 >= min_value && $10 <= max_value)'
```

- Print table of users with UID >=1000 with header and formatted output, using colon as separator (`%-20s` mean: 20 left-align string characters, `%6s` means: 6 right-align string characters):

```
awk 'BEGIN {FS=":";printf "%-20s %6s %25s\n", "Name", "UID", "Shell"} $4 >= 1000 {printf "%-20s %6d %25s\n",
$1, $4, $7}' /etc/passwd
```

sed

> Edit text in a scriptable manner. See also: `awk`, `ed`. More information: https://manned.org/sed.1posix.

- Replace all `apple` (basic regex) occurrences with `mango` (basic regex) in all input lines and print the result to `stdout`:

```
command | sed 's/apple/mango/g'
```

- Execute a specific script file and print the result to `stdout`:

```
command | sed -f path/to/script.sed
```

- Print just a first line to `stdout`:

```
command | sed -n '1p'
```

cut

> Cut out fields from `stdin` or files. More information: https://www.gnu.org/software/coreutils/manual/html_node/cut-invocation.html.

- Print a specific **c**haracter/**f**ield range of each line:

```
command | cut --characters|fields 1|1,10|1-10|1-|-10
```

- Print a **f**ield range of each line with a specific **d**elimiter:

```
command | cut --delimiter "delimiter" --fields 1|1,10|1-10|1-|-10
```

- Print a **c**haracter range of each line of the specific file:

```
cut --characters 1 path/to/file
```

- Print specific **f**ields of `NUL` terminated lines (e.g. as in `find . -print0`) instead of newlines:

```
command | cut --zero-terminated --fields 1
```

sort

> Sort lines of text files. More information: https://www.gnu.org/software/coreutils/manual/html_node/sort-invocation.html.

- Sort a file in ascending order:

```
sort path/to/file
```

- Sort a file in descending order:

```
sort --reverse path/to/file
```

- Sort a file in case-insensitive way:

```
sort --ignore-case path/to/file
```

- Sort a file using numeric rather than alphabetic order:

```
sort --numeric-sort path/to/file
```

- Sort `/etc/passwd` by the 3rd field of each line numerically, using ":" as a field separator:

```
sort --field-separator=: --key=3n /etc/passwd
```

- As above, but when items in the 3rd field are equal, sort by the 4th field by numbers with exponents:

```
sort -t : -k 3,3n -k 4,4g /etc/passwd
```

- Sort a file preserving only unique lines:

```
sort --unique path/to/file
```

- Sort a file, printing the output to the specified output file (can be used to sort a file in-place):

```
sort --output=path/to/file path/to/file
```

uniq

> Output the unique lines from a input or file. Since it does not detect repeated lines unless they are adjacent, we need to sort them first. More information: https://www.gnu.org/software/coreutils/manual/html_node/uniq-invocation.html.

- Display each line once:

```
sort path/to/file | uniq
```

- Display only unique lines:

```
sort path/to/file | uniq -u
```

- Display only duplicate lines:

```
sort path/to/file | uniq -d
```

- Display number of occurrences of each line along with that line:

```
sort path/to/file | uniq -c
```

- Display number of occurrences of each line, sorted by the most frequent:

```
sort path/to/file | uniq -c | sort -nr
```

diff

> Compare files and directories. More information: https://manned.org/diff.

- Compare files (lists changes to turn `old_file` into `new_file`):

```
diff old_file new_file
```

- Compare files, ignoring white spaces:

```
diff -w|--ignore-all-space old_file new_file
```

- Compare files, showing the differences side by side:

```
diff -y|--side-by-side old_file new_file
```

- Compare files, showing the differences in unified format (as used by `git diff`):

```
diff -u|--unified old_file new_file
```

- Compare directories recursively (shows names for differing files/directories as well as changes made to files):

```
diff -r|--recursive old_directory new_directory
```

- Compare directories, only showing the names of files that differ:

```
diff -r|--recursive -q|--brief old_directory new_directory
```

- Create a patch file for Git from the differences of two text files, treating nonexistent files as empty:

```
diff -a|--text -u|--unified -N|--new-file old_file new_file > diff.patch
```

- Compare files, showing output in color and try hard to find smaller set of changes:

```
diff -d|--minimal --color=always old_file new_file
```

cmp

> Compare two files byte by byte. More information: https://www.gnu.org/software/diffutils/manual/html_node/Invoking-cmp.html.

- Output char and line number of the first difference between two files:

```
cmp path/to/file1 path/to/file2
```

- Output info of the first difference: char, line number, bytes, and values:

```
cmp --print-bytes path/to/file1 path/to/file2
```

- Output the byte numbers and values of every difference:

```
cmp --verbose path/to/file1 path/to/file2
```

- Compare files but output nothing, yield only the exit status:

```
cmp --quiet path/to/file1 path/to/file2
```

tr

> Translate characters: run replacements based on single characters and character sets. More information: https://www.gnu.org/software/coreutils/manual/html_node/tr-invocation.html.

- Replace all occurrences of a character in a file, and print the result:

```
tr find_character replace_character < path/to/file
```

- Replace all occurrences of a character from another command's output:

```
echo text | tr find_character replace_character
```

- Map each character of the first set to the corresponding character of the second set:

```
tr 'abcd' 'jkmn' < path/to/file
```

- Delete all occurrences of the specified set of characters from the input:

```
tr -d 'input_characters' < path/to/file
```

- Compress a series of identical characters to a single character:

```
tr -s 'input_characters' < path/to/file
```

- Translate the contents of a file to upper-case:

```
tr "[:lower:]" "[:upper:]" < path/to/file
```

- Strip out non-printable characters from a file:

```
tr -cd "[:print:]" < path/to/file
```

wc

> Count lines, words, and bytes. More information: https://www.gnu.org/software/coreutils/manual/html_node/wc-invocation.html.

- Count all lines in a file:

```
wc --lines path/to/file
```

- Count all words in a file:

```
wc --words path/to/file
```

- Count all bytes in a file:

```
wc --bytes path/to/file
```

- Count all characters in a file (taking multi-byte characters into account):

```
wc --chars path/to/file
```

- Count all lines, words and bytes from `stdin`:

```
find . | wc
```

- Count the length of the longest line in number of characters:

```
wc --max-line-length path/to/file
```

Disk & System Monitoring

- df - Check disk space usage
- du - Analyze disk usage per directory
- free - Check system memory usage
- uptime - Show system running time
- top - Monitor real-time system resource usage
- htop - Enhanced process viewer
- ps - View active processes
- kill - Terminate a process
- nice - Adjust process priority
- iotop - Monitor disk I/O usage
- vmstat - Show system performance statistics
- iostat - Display CPU and I/O statistics

df

> Display an overview of the filesystem disk space usage. More information: https://manned.org/df.1posix.

- Display all filesystems and their disk usage using 512-byte units:

```
df
```

- Display the filesystem and its disk usage containing the given file or directory:

```
df path/to/file_or_directory
```

- Use 1024-byte units when writing space figures:

```
df -k
```

- Display information in a portable way:

```
df -P
```

du

> Disk usage: estimate and summarize file and directory space usage. More information: https://www.gnu.org/software/coreutils/manual/html_node/du-invocation.html.

- List the sizes of a directory and any subdirectories, in the given unit (B/KiB/MiB):

```
du -b|k|m path/to/directory
```

- List the sizes of a directory and any subdirectories, in human-readable form (i.e. auto-selecting the appropriate unit for each size):

```
du -h path/to/directory
```

- Show the size of a single directory, in human-readable units:

```
du -sh path/to/directory
```

- List the human-readable sizes of a directory and of all the files and directories within it:

```
du -ah path/to/directory
```

- List the human-readable sizes of a directory and any subdirectories, up to N levels deep:

```
du -h --max-depth=N path/to/directory
```

- List the human-readable size of all `.jpg` files in current directory, and show a cumulative total at the end:

```
du -ch ./*.jpg
```

- List all files and directories (including hidden ones) above a certain threshold size (useful for investigating what is actually taking up the space):

```
du --all --human-readable --threshold 1G|1024M|1048576K .[^.]* *
```

free

> Display amount of free and used memory in the system. More information: https://manned.org/free.

- Display system memory:

```
free
```

- Display memory in Bytes/KB/MB/GB:

```
free -b|k|m|g
```

- Display memory in human-readable units:

```
free -h
```

- Refresh the output every 2 seconds:

```
free -s 2
```

uptime

> Tell how long the system has been running and other information. More information: https://www.gnu.org/software/coreutils/manual/html_node/uptime-invocation.html.

- Print current time, uptime, number of logged-in users and other information:

```
uptime
```

- Show only the amount of time the system has been booted for:

```
uptime --pretty
```

- Print the date and time the system booted up at:

```
uptime --since
```

- Display version:

```
uptime --version
```

top

> Display dynamic real-time information about running processes. More information: https://manned.org/top.

- Start `top`:

```
top
```

- Do not show any idle or zombie processes:

```
top -i
```

- Show only processes owned by given user:

```
top -u username
```

- Sort processes by a field:

```
top -o field_name
```

- Show the individual threads of a given process:

```
top -Hp process_id
```

- Show only the processes with the given PID(s), passed as a comma-separated list. (Normally you wouldn't know PIDs off hand. This example picks the PIDs from the process name):

```
top -p $(pgrep -d ',' process_name)
```

- Display help about interactive commands:

```
?
```

htop

> Display dynamic real-time information about running processes. An enhanced version of `top`. More information: https://htop.dev/.

- Start `htop`:

```
htop
```

- Start `htop` displaying processes owned by a specific user:

```
htop --user username
```

- Display processes hierarchically in a tree view to show the parent-child relationships:

```
htop --tree
```

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

```
htop --sort sort_item
```

- Start `htop` with the specified delay between updates, in tenths of a second (i.e. 50 = 5 seconds):

```
htop --delay 50
```

- See interactive commands while running htop:

```
?
```

- Switch to a different tab:

```
tab
```

- Display help:

```
htop --help
```

ps

> Information about running processes. More information: https://manned.org/ps.

- List all running processes:

```
ps aux
```

- List all running processes including the full command string:

```
ps auxww
```

- Search for a process that matches a string (the brackets will prevent `grep` from matching itself):

```
ps aux | grep string
```

- List all processes of the current user in extra full format:

```
ps --user $(id -u) -F
```

- List all processes of the current user as a tree:

```
ps --user $(id -u) f
```

- Get the parent PID of a process:

```
ps -o ppid= -p pid
```

- Sort processes by memory consumption:

```
ps --sort size
```

kill

> Sends a signal to a process, usually related to stopping the process. All signals except for SIGKILL and SIGSTOP can be intercepted by the process to perform a clean exit. More information: https://manned.org/kill.1posix.

- Terminate a program using the default SIGTERM (terminate) signal:

```
kill process_id
```

- List available signal names (to be used without the `SIG` prefix):

```
kill -l
```

- Terminate a program using the SIGHUP (hang up) signal. Many daemons will reload instead of terminating:

```
kill -1|HUP process_id
```

- Terminate a program using the SIGINT (interrupt) signal. This is typically initiated by the user pressing `Ctrl + C`:

```
kill -2|INT process_id
```

- Signal the operating system to immediately terminate a program (which gets no chance to capture the signal):

```
kill -9|KILL process_id
```

- Signal the operating system to pause a program until a SIGCONT ("continue") signal is received:

```
kill -17|STOP process_id
```

- Send a `SIGUSR1` signal to all processes with the given GID (group id):

```
kill -SIGUSR1 -group_id
```

nice

> Execute a program with a custom scheduling priority (niceness). Niceness values range from -20 (the highest priority) to 19 (the lowest).
> More information: https://www.gnu.org/software/coreutils/manual/html_node/nice-invocation.html.

- Launch a program with altered priority:

```
nice -niceness_value command
```

- Define the priority with an explicit option:

```
nice -n|--adjustment niceness_value command
```

iotop

> Display a table of current I/O usage by processes or threads. More information: https://manned.org/iotop.

- Start top-like I/O monitor:

```
sudo iotop
```

- Show only processes or threads actually doing I/O:

```
sudo iotop --only
```

- Show I/O usage in non-interactive mode:

```
sudo iotop --batch
```

- Show only I/O usage of processes (default is to show all threads):

```
sudo iotop --processes
```

- Show I/O usage of given PID(s):

```
sudo iotop --pid=PID
```

- Show I/O usage of a given user:

```
sudo iotop --user=user
```

- Show accumulated I/O instead of bandwidth:

```
sudo iotop --accumulated
```

vmstat

> Report information about processes, memory, paging, block IO, traps, disks and CPU activity. More information: https://manned.org/vmstat.

- Display virtual memory statistics:

```
vmstat
```

- Display reports every 2 seconds for 5 times:

```
vmstat 2 5
```

iostat

> Report statistics for devices and partitions. More information: https://manned.org/iostat.

- Display a report of CPU and disk statistics since system startup:

```
iostat
```

- Display a report of CPU and disk statistics with units converted to megabytes:

```
iostat -m
```

- Display CPU statistics:

```
iostat -c
```

- Display disk statistics with disk names (including LVM):

```
iostat -N
```

- Display extended disk statistics with disk names for device "sda":

```
iostat -xN sda
```

- Display incremental reports of CPU and disk statistics every 2 seconds:

```
iostat 2
```

Networking & Server Access

- ping - Test network connectivity
- traceroute - Trace the path to a remote host
- netstat - Show network connections
- ss - Display active network connections
- curl - Fetch data from a web URL
- wget - Download files from the web
- scp - Securely copy files between systems
- rsync - Efficiently copy files between systems
- ssh - Securely log in to remote servers
- ftp - Transfer files using FTP
- nmap - Scan network ports
- dig - Query DNS records

ping

> Send ICMP ECHO_REQUEST packets to network hosts. More information: https://manned.org/ping.

- Ping host:

```
ping host
```

- Ping a host only a specific number of times:

```
ping -c count host
```

- Ping host, specifying the interval in seconds between requests (default is 1 second):

```
ping -i seconds host
```

- Ping host without trying to lookup symbolic names for addresses:

```
ping -n host
```

- Ping host and ring the bell when a packet is received (if your terminal supports it):

```
ping -a host
```

- Also display a message if no response was received:

```
ping -O host
```

- Ping a host with specific number of pings, per-packet response timeout (`-W`), and total time limit (`-w`) of the entire ping run:

```
ping -c count -W seconds -w seconds host
```

traceroute

> Print the route packets trace to network host. More information: https://manned.org/traceroute.

- Traceroute to a host:

```
traceroute example.com
```

- Disable IP address and host name mapping:

```
traceroute -n example.com
```

- Specify wait time in seconds for response:

```
traceroute --wait=0.5 example.com
```

- Specify number of queries per hop:

```
traceroute --queries=5 example.com
```

- Specify size in bytes of probing packet:

```
traceroute example.com 42
```

- Determine the MTU to the destination:

```
traceroute --mtu example.com
```

- Use ICMP instead of UDP for tracerouting:

```
traceroute --icmp example.com
```

netstat

> Display network-related information such as open connections, open socket ports, etc. See also: `ss`. More information: https://manned.org/netstat.

- List all ports:

`netstat --all`

- List all listening ports:

`netstat --listening`

- List listening TCP ports:

`netstat --tcp`

- Display PID and program names:

`netstat --program`

- List information continuously:

`netstat --continuous`

- List routes and do not resolve IP addresses to hostnames:

`netstat --route --numeric`

- List listening TCP and UDP ports (+ user and process if you're root):

`netstat --listening --program --numeric --tcp --udp --extend`

ss

> Utility to investigate sockets. More information: https://manned.org/ss.8.

- Show all TCP/UDP/RAW/UNIX sockets:

```
ss -a -t|-u|-w|-x
```

- Filter TCP sockets by states, only/exclude:

```
ss state/exclude bucket/big/connected/synchronized/...
```

- Show all TCP sockets connected to the local HTTPS port (443):

```
ss -t src :443
```

- Show all TCP sockets listening on the local 8080 port:

```
ss -lt src :8080
```

- Show all TCP sockets along with processes connected to a remote SSH port:

```
ss -pt dst :ssh
```

- Show all UDP sockets connected on specific source and destination ports:

```
ss -u 'sport == :source_port and dport == :destination_port'
```

- Show all TCP IPv4 sockets locally connected on the subnet 192.168.0.0/16:

```
ss -4t src 192.168/16
```

- Kill IPv4 or IPv6 Socket Connection with destination IP 192.168.1.17 and destination port 8080:

```
ss --kill dst 192.168.1.17 dport = 8080
```

curl

> Transfers data from or to a server. Supports most protocols, including HTTP, HTTPS, FTP, SCP, etc. More information: https://curl.se/docs/manpage.html.

- Make an HTTP GET request and dump the contents in `stdout`:

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

- Make an HTTP GET request, fo[L]low any `3xx` redirects, and [D]ump the reply headers and contents to `stdout`:

```
curl --location --dump-header - https://example.com
```

- Download a file, saving the [O]utput under the filename indicated by the URL:

```
curl --remote-name https://example.com/filename.zip
```

- Send form-encoded **d**ata (POST request of type `application/x-www-form-urlencoded`). Use `--data @file_name` or `--data @'-'` to read from `stdin`:

```
curl -X POST --data 'name=bob' http://example.com/form
```

- Send a request with an extra header, using a custom HTTP method and over a pro**x**y (such as BurpSuite), ignoring insecure self-signed certificates:

```
curl -k --proxy http://127.0.0.1:8080 --header 'Authorization: Bearer token' --request
GET|PUT|POST|DELETE|PATCH|... https://example.com
```

- Send data in JSON format, specifying the appropriate Content-Type [H]eader:

```
curl --data '{"name":"bob"}' --header 'Content-Type: application/json' http://example.com/users/1234
```

- Pass client certificate and key for a resource, skipping certificate validation:

```
curl --cert client.pem --key key.pem --insecure https://example.com
```

- Resolve a hostname to a custom IP address, with **v**erbose output (similar to editing the `/etc/hosts` file for custom DNS resolution):

```
curl --verbose --resolve example.com:80:127.0.0.1 http://example.com
```

wget

> Download files from the Web. Supports HTTP, HTTPS, and FTP. More information: https://www.gnu.org/software/wget.

- Download the contents of a URL to a file (named "foo" in this case):

```
wget https://example.com/foo
```

- Download the contents of a URL to a file (named "bar" in this case):

```
wget --output-document bar https://example.com/foo
```

- Download a single web page and all its resources with 3-second intervals between requests (scripts, stylesheets, images, etc.):

```
wget --page-requisites --convert-links --wait=3 https://example.com/somepage.html
```

- Download all listed files within a directory and its sub-directories (does not download embedded page elements):

```
wget --mirror --no-parent https://example.com/somepath/
```

- Limit the download speed and the number of connection retries:

```
wget --limit-rate=300k --tries=100 https://example.com/somepath/
```

- Download a file from an HTTP server using Basic Auth (also works for FTP):

```
wget --user=username --password=password https://example.com
```

- Continue an incomplete download:

```
wget --continue https://example.com
```

- Download all URLs stored in a text file to a specific directory:

```
wget --directory-prefix path/to/directory --input-file URLs.txt
```

scp

> Secure copy. Copy files between hosts using Secure Copy Protocol over SSH. More information: https://man.openbsd.org/scp.

- Copy a local file to a remote host:

```
scp path/to/local_file remote_host:path/to/remote_file
```

- Use a specific port when connecting to the remote host:

```
scp -P port path/to/local_file remote_host:path/to/remote_file
```

- Copy a file from a remote host to a local directory:

```
scp remote_host:path/to/remote_file path/to/local_directory
```

- Recursively copy the contents of a directory from a remote host to a local directory:

```
scp -r remote_host:path/to/remote_directory path/to/local_directory
```

- Copy a file between two remote hosts transferring through the local host:

```
scp -3 host1:path/to/remote_file host2:path/to/remote_directory
```

- Use a specific username when connecting to the remote host:

```
scp path/to/local_file remote_username@remote_host:path/to/remote_directory
```

- Use a specific SSH private key for authentication with the remote host:

```
scp -i ~/.ssh/private_key path/to/local_file remote_host:path/to/remote_file
```

- Use a specific proxy when connecting to the remote host:

```
scp -J proxy_username@proxy_host path/to/local_file remote_host:path/to/remote_file
```

rsync

> Transfer files either to or from a remote host (but not between two remote hosts), by default using SSH. To specify a remote path, use `user@host:path/to/file_or_directory`. More information: https://download.samba.org/pub/rsync/rsync.1.

- Transfer a file:

```
rsync path/to/source path/to/destination
```

- Use archive mode (recursively copy directories, copy symlinks without resolving, and preserve permissions, ownership and modification times):

```
rsync -a|--archive path/to/source path/to/destination
```

- Compress the data as it is sent to the destination, display verbose and human-readable progress, and keep partially transferred files if interrupted:

```
rsync -zvhP|--compress --verbose --human-readable --partial --progress path/to/source path/to/destination
```

- Recursively copy directories:

```
rsync -r|--recursive path/to/source path/to/destination
```

- Transfer directory contents, but not the directory itself:

```
rsync -r|--recursive path/to/source/ path/to/destination
```

- Use archive mode, resolve symlinks, and skip files that are newer on the destination:

```
rsync -auL|--archive --update --copy-links path/to/source path/to/destination
```

- Transfer a directory from a remote host running `rsyncd` and delete files on the destination that do not exist on the source:

```
rsync -r|--recursive --delete rsync://host:path/to/source path/to/destination
```

- Transfer a file over SSH using a different port than the default (22) and show global progress:

```
rsync -e|--rsh 'ssh -p port' --info=progress2 host:path/to/source path/to/destination
```

ssh

> Secure Shell is a protocol used to securely log onto remote systems. It can be used for logging or executing commands on a remote server. More information: https://man.openbsd.org/ssh.

- Connect to a remote server:

```
ssh username@remote_host
```

- Connect to a remote server with a specific identity (private key):

```
ssh -i path/to/key_file username@remote_host
```

- Connect to a remote server using a specific **p**ort:

```
ssh username@remote_host -p 2222
```

- Run a command on a remote server with a **t**ty allocation allowing interaction with the remote command:

```
ssh username@remote_host -t command command_arguments
```

- SSH tunneling: [D]ynamic port forwarding (SOCKS proxy on `localhost:1080`):

```
ssh -D 1080 username@remote_host
```

- SSH tunneling: Forward a specific port (`localhost:9999` to `example.org:80`) along with disabling pseudo-[T]ty allocation and executio[N] of remote commands:

```
ssh -L 9999:example.org:80 -N -T username@remote_host
```

- SSH [J]umping: Connect through a jumphost to a remote server (Multiple jump hops may be specified separated by comma characters):

```
ssh -J username@jump_host username@remote_host
```

- Close a hanged session:

```
<Enter> ~ .
```

ftp

> Tools to interact with a server via File Transfer Protocol. More information: https://manned.org/ftp.

- Connect to an FTP server:

```
ftp ftp.example.com
```

- Connect to an FTP server specifying its IP address and port:

```
ftp ip_address port
```

- Switch to binary transfer mode (graphics, compressed files, etc):

```
binary
```

- Transfer multiple files without prompting for confirmation on every file:

```
prompt off
```

- Download multiple files (glob expression):

```
mget *.png
```

- Upload multiple files (glob expression):

```
mput *.zip
```

- Delete multiple files on the remote server:

```
mdelete *.txt
```

- Rename a file on the remote server:

```
rename original_filename new_filename
```

nmap

> Network exploration tool and security/port scanner. Some features (e.g. SYN scan) activate only when `nmap` is run with root privileges.
> More information: https://nmap.org/book/man.html.

- Scan the top 1000 ports of a remote host with various **v**erbosity levels:

```
nmap -v1|2|3 ip_or_hostname
```

- Run a ping sweep over an entire subnet or individual hosts very aggressively:

```
nmap -T5 -sn 192.168.0.0/24|ip_or_hostname1,ip_or_hostname2,...
```

- Enable OS detection, version detection, script scanning, and traceroute of hosts from a file:

```
sudo nmap -A -iL path/to/file.txt
```

- Scan a specific list of ports (use `-p-` for all ports from 1 to 65535):

```
nmap -p port1,port2,... ip_or_host1,ip_or_host2,...
```

- Perform service and version detection of the top 1000 ports using default NSE scripts, writing results (`-oA`) to output files:

```
nmap -sC -sV -oA top-1000-ports ip_or_host1,ip_or_host2,...
```

- Scan target(s) carefully using `default and safe` NSE scripts:

```
nmap --script "default and safe" ip_or_host1,ip_or_host2,...
```

- Scan for web servers running on standard ports 80 and 443 using all available `http-*` NSE scripts:

```
nmap --script "http-*" ip_or_host1,ip_or_host2,... -p 80,443
```

- Attempt evading IDS/IPS detection by using an extremely slow scan (`-T0`), decoy source addresses (`-D`), **f**ragmented packets, random data and other methods:

```
sudo nmap -T0 -D decoy_ip1,decoy_ip2,... --source-port 53 -f --data-length 16 -Pn ip_or_host
```

dig

> DNS lookup utility. More information: https://manned.org/dig.

- Lookup the IP(s) associated with a hostname (A records):

```
dig +short example.com
```

- Get a detailed answer for a given domain (A records):

```
dig +noall +answer example.com
```

- Query a specific DNS record type associated with a given domain name:

```
dig +short example.com A|MX|TXT|CNAME|NS
```

- Specify an alternate DNS server to query and optionally use DNS over TLS (DoT):

```
dig +tls @1.1.1.1|8.8.8.8|9.9.9.9|... example.com
```

- Perform a reverse DNS lookup on an IP address (PTR record):

```
dig -x 8.8.8.8
```

- Find authoritative name servers for the zone and display SOA records:

```
dig +nssearch example.com
```

- Perform iterative queries and display the entire trace path to resolve a domain name:

```
dig +trace example.com
```

- Query a DNS server over a non-standard **p**ort using the TCP protocol:

```
dig +tcp -p port @dns_server_ip example.com
```

Job Scheduling & Automation

- cron - Schedule recurring tasks
- at - Schedule one-time tasks
- nohup - Run processes that ignore hangups
- screen - Keep a session running in the background
- tmux - Terminal multiplexer for managing multiple sessions
- watch - Run a command repeatedly at intervals

cron

> A system scheduler for running jobs or tasks unattended. The command to submit, edit or delete entries to `cron` is called `crontab`.

- View documentation for managing `cron` entries:

  ```
  tldr crontab
  ```

at

> Execute commands once at a later time. Results will be sent to the users mail. More information: https://manned.org/at.

- Start the `atd` daemon:

```
systemctl start atd
```

- Create commands interactively and execute them in 5 minutes (press `<Ctrl> + D` when done):

```
at now + 5 minutes
```

- Create commands interactively and execute them at a specific time:

```
at hh:mm
```

- Execute a command from `stdin` at 10:00 AM today:

```
echo "command" | at 1000
```

- Execute commands from a given file next Tuesday:

```
at -f path/to/file 9:30 PM Tue
```

nohup

> Allows for a process to live when the terminal gets killed. More information: https://www.gnu.org/software/coreutils/manual/html_node/nohup-invocation.htm

- Run a process that can live beyond the terminal:

```
nohup command argument1 argument2 ...
```

- Launch `nohup` in background mode:

```
nohup command argument1 argument2 ... &
```

- Run a shell script that can live beyond the terminal:

```
nohup path/to/script.sh &
```

- Run a process and write the output to a specific file:

```
nohup command argument1 argument2 ... > path/to/output_file &
```

screen

> Hold a session open on a remote server. Manage multiple windows with a single SSH connection. See also `tmux` and `zellij`. More information: https://manned.org/screen.

- Start a new screen session:

```
screen
```

- Start a new named screen session:

```
screen -S session_name
```

- Start a new daemon and log the output to `screenlog.x`:

```
screen -dmLS session_name command
```

- Show open screen sessions:

```
screen -ls
```

- Reattach to an open screen:

```
screen -r session_name
```

- Detach from inside a screen:

```
<Ctrl> + A, D
```

- Kill the current screen session:

```
<Ctrl> + A, K
```

- Kill a detached screen:

```
screen -X -S session_name quit
```

tmux

> Terminal multiplexer. It allows multiple sessions with windows, panes, and more. See also: `zellij`, `screen`. More information: https://github.com/tmux/tmux.

- Start a new session:

  `tmux`

- Start a new named session:

  `tmux new -s name`

- List existing sessions:

  `tmux ls`

- Attach to the most recently used session:

  `tmux attach`

- Detach from the current session (inside a tmux session):

  `<Ctrl>-B d`

- Create a new window (inside a tmux session):

  `<Ctrl>-B c`

- Switch between sessions and windows (inside a tmux session):

  `<Ctrl>-B w`

- Kill a session by name:

  `tmux kill-session -t name`

watch

> Execute a program periodically and monitor the output in full-screen mode. More information: https://manned.org/watch.

- Repeatedly run a command and show the result:

```
watch command
```

- Re-run a command every 60 seconds:

```
watch --interval 60 command
```

- Monitor disk space, highlighting differences as they appear:

```
watch --differences df
```

- Repeatedly run a pipeline and show the result:

```
watch "command_1 | command_2 | command_3"
```

- Exit `watch` if the visible output changes:

```
watch --chgexit lsblk
```

- Interpret terminal control characters:

```
watch --color ls --color=always
```

Log Management & Debugging

- dmesg - Display kernel logs
- journalctl - View system logs
- tail -f /var/log/syslog - Monitor system logs in real-time
- grep ERROR /var/log/messages - Filter error messages from logs
- strace - Trace system calls of a process
- lsof - List open files
- uptime - Show system running time
- who - Show logged-in users

dmesg

> Write the kernel messages to `stdout`. More information: https://manned.org/dmesg.

- Show kernel messages:

```
sudo dmesg
```

- Show kernel error messages:

```
sudo dmesg --level err
```

- Show kernel messages and keep reading new ones, similar to `tail -f` (available in kernels 3.5.0 and newer):

```
sudo dmesg -w
```

- Show how much physical memory is available on this system:

```
sudo dmesg | grep -i memory
```

- Show kernel messages 1 page at a time:

```
sudo dmesg | less
```

- Show kernel messages with a timestamp (available in kernels 3.5.0 and newer):

```
sudo dmesg -T
```

- Show kernel messages in human-readable form (available in kernels 3.5.0 and newer):

```
sudo dmesg -H
```

- Colorize output (available in kernels 3.5.0 and newer):

```
sudo dmesg -L
```

journalctl

> Query the systemd journal. More information: https://manned.org/journalctl.

- Show all messages with priority level 3 (errors) from this **b**oot:

```
journalctl -b --priority=3
```

- Delete journal logs which are older than 2 days:

```
journalctl --vacuum-time=2d
```

- Show only the last N li**n**es and **f**ollow new messages (like `tail -f` for traditional syslog):

```
journalctl --lines N --follow
```

- Show all messages by a specific **u**nit:

```
journalctl --unit unit
```

- Show logs for a given unit since the last time it started:

```
journalctl _SYSTEMD_INVOCATION_ID=$(systemctl show --value --property=InvocationID unit)
```

- Filter messages within a time range (either timestamp or placeholders like "yesterday"):

```
journalctl --since now|today|yesterday|tomorrow --until "YYYY-MM-DD HH:MM:SS"
```

- Show all messages by a specific process:

```
journalctl _PID=pid
```

- Show all messages by a specific executable:

```
journalctl path/to/executable
```

strace

> Troubleshooting tool for tracing system calls. More information: https://manned.org/strace.

- Start tracing a specific **p**rocess by its PID:

```
strace -p pid
```

- Trace a **p**rocess and filt**e**r output by system call:

```
strace -p pid -e system_call,system_call2,...
```

- Count time, calls, and errors for each system call and report a summary on program exit:

```
strace -p pid -c
```

- Show the [T]ime spent in every system call and specify the maximum string **s**ize to print:

```
strace -p pid -T -s 32
```

- Start tracing a program by executing it:

```
strace program
```

- Start tracing file operations of a program:

```
strace -e trace=file program
```

- Start tracing network operations of a program as well as all its **f**orked and child processes, saving the **o**utput to a file:

```
strace -f -e trace=network -o trace.txt program
```

lsof

> Lists open files and the corresponding processes. Note: Root privileges (or sudo) is required to list files opened by others. More information: https://manned.org/lsof.

- Find the processes that have a given file open:

```
lsof path/to/file
```

- Find the process that opened a local internet port:

```
lsof -i :port
```

- Only output the process ID (PID):

```
lsof -t path/to/file
```

- List files opened by the given user:

```
lsof -u username
```

- List files opened by the given command or process:

```
lsof -c process_or_command_name
```

- List files opened by a specific process, given its PID:

```
lsof -p PID
```

- List open files in a directory:

```
lsof +D path/to/directory
```

- Find the process that is listening on a local IPv6 TCP port and don't convert network or port numbers:

```
lsof -i6TCP:port -sTCP:LISTEN -n -P
```

uptime

> Tell how long the system has been running and other information. More information: https://www.gnu.org/software/coreutils/manual/html_node/uptime-invocation.html.

- Print current time, uptime, number of logged-in users and other information:

```
uptime
```

- Show only the amount of time the system has been booted for:

```
uptime --pretty
```

- Print the date and time the system booted up at:

```
uptime --since
```

- Display version:

```
uptime --version
```

Archiving & Compression

- tar - Archive multiple files
- zip - Compress files into a .zip archive
- unzip - Extract files from a .zip archive
- gzip - Compress a single file
- gunzip - Decompress a .gz file
- bzip2 - Compress files using Burrows-Wheeler block sorting
- xz - Highly efficient file compression

tar

> Archiving utility. Often combined with a compression method, such as `gzip` or `bzip2`. More information: https://www.gnu.org/software/tar.

- **C**reate an archive and write it to a **f**ile:

```
tar cf path/to/target.tar path/to/file1 path/to/file2 ...
```

- **C**reate a g**z**ipped archive and write it to a **f**ile:

```
tar czf path/to/target.tar.gz path/to/file1 path/to/file2 ...
```

- **C**reate a g**z**ipped (compressed) archive from a directory using relative paths:

```
tar czf path/to/target.tar.gz --directory=path/to/directory .
```

- E**x**tract a (compressed) archive **f**ile into the current directory **v**erbosely:

```
tar xvf path/to/source.tar[.gz|.bz2|.xz]
```

- E**x**tract a (compressed) archive **f**ile into the target directory:

```
tar xf path/to/source.tar[.gz|.bz2|.xz] --directory=path/to/directory
```

- **C**reate a compressed archive and write it to a **f**ile, using the file extension to **a**utomatically determine the compression program:

```
tar caf path/to/target.tar.xz path/to/file1 path/to/file2 ...
```

- Lis**t** the contents of a tar **f**ile **v**erbosely:

```
tar tvf path/to/source.tar
```

- E**x**tract files matching a pattern from an archive **f**ile:

```
tar xf path/to/source.tar --wildcards "*.html"
```

zip

> Package and compress (archive) files into a Zip archive. See also: `unzip`. More information: https://manned.org/zip.

- Add files/directories to a specific archive (**r**ecursively):

```
zip -r path/to/compressed.zip path/to/file_or_directory1 path/to/file_or_directory2 ...
```

- Remove files/directories from a specific archive (**d**elete):

```
zip -d path/to/compressed.zip path/to/file_or_directory1 path/to/file_or_directory2 ...
```

- Archive files/directories e**x**cluding specified ones:

```
zip -r path/to/compressed.zip path/to/file_or_directory1 path/to/file_or_directory2 ... -x
path/to/excluded_files_or_directories
```

- Archive files/directories with a specific compression level (`0` - the lowest, `9` - the highest):

```
zip -r -0..9 path/to/compressed.zip path/to/file_or_directory1 path/to/file_or_directory2 ...
```

- Create an **e**ncrypted archive with a specific password:

```
zip -r -e path/to/compressed.zip path/to/file_or_directory1 path/to/file_or_directory2 ...
```

- Archive files/directories to a multi-part **s**plit Zip archive (e.g. 3 GB parts):

```
zip -r -s 3g path/to/compressed.zip path/to/file_or_directory1 path/to/file_or_directory2 ...
```

- Print a specific archive contents:

```
zip -sf path/to/compressed.zip
```

unzip

> Extract files/directories from Zip archives. See also: `zip`. More information: https://manned.org/unzip.

- Extract all files/directories from specific archives into the current directory:

```
unzip path/to/archive1.zip path/to/archive2.zip ...
```

- Extract files/directories from archives to a specific path:

```
unzip path/to/archive1.zip path/to/archive2.zip ... -d path/to/output
```

- Extract files/directories from archives to `stdout` alongside the extracted file names:

```
unzip -c path/to/archive1.zip path/to/archive2.zip ...
```

- Extract an archive created on Windows, containing files with non-ASCII (e.g. Chinese or Japanese characters) filenames:

```
unzip -O gbk path/to/archive1.zip path/to/archive2.zip ...
```

- List the contents of a specific archive without extracting them:

```
unzip -l path/to/archive.zip
```

- Extract a specific file from an archive:

```
unzip -j path/to/archive.zip path/to/file1_in_archive path/to/file2_in_archive ...
```

gzip

> Compress/uncompress files with `gzip` compression (LZ77). More information: https://www.gnu.org/software/gzip/manual/gzip.html.

- Compress a file, replacing it with a `gzip` archive:

```
gzip path/to/file
```

- Decompress a file, replacing it with the original uncompressed version:

```
gzip -d|--decompress path/to/file.gz
```

- Compress a file, keeping the original file:

```
gzip -k|--keep path/to/file
```

- Compress a file, specifying the output filename:

```
gzip -c|--stdout path/to/file > path/to/compressed_file.gz
```

- Decompress a `gzip` archive specifying the output filename:

```
gzip -c|--stdout -d|--decompress path/to/file.gz > path/to/uncompressed_file
```

- Specify the compression level. 1 is the fastest (low compression), 9 is the slowest (high compression), 6 is the default:

```
gzip -1..9 -c|--stdout path/to/file > path/to/compressed_file.gz
```

- Display the name and reduction percentage for each file compressed or decompressed:

```
gzip -v|--verbose -d|--decompress path/to/file.gz
```

gunzip

> Extract files from a `gzip` (`.gz`) archive. More information: https://manned.org/gunzip.

- Extract a file from an archive, replacing the original file if it exists:

```
gunzip archive.tar.gz
```

- Extract a file to a target destination:

```
gunzip --stdout archive.tar.gz > archive.tar
```

- Extract a file and keep the archive file:

```
gunzip --keep archive.tar.gz
```

- List the contents of a compressed file:

```
gunzip --list file.txt.gz
```

- Decompress an archive from `stdin` :

```
cat path/to/archive.gz | gunzip
```

bzip2

> A block-sorting file compressor. More information: https://manned.org/bzip2.

- Compress a file:

```
bzip2 path/to/file_to_compress
```

- **D**ecompress a file:

```
bzip2 -d path/to/compressed_file.bz2
```

- **D**ecompress a file to `stdout`:

```
bzip2 -dc path/to/compressed_file.bz2
```

- Test the integrity of each file inside the archive file:

```
bzip2 --test path/to/compressed_file.bz2
```

- Show the compression ratio for each file processed with detailed information:

```
bzip2 --verbose path/to/compressed_files.bz2
```

- Decompress a file overwriting existing files:

```
bzip2 --force path/to/compressed_file.bz2
```

- Display help:

```
bzip2 -h
```

xz

> Compress or decompress XZ and LZMA files. More information: https://manned.org/xz.

- Compress a file using xz:

```
xz path/to/file
```

- Decompress an XZ file:

```
xz --decompress path/to/file.xz
```

- Compress a file using lzma:

```
xz --format=lzma path/to/file
```

- Decompress an LZMA file:

```
xz --decompress --format=lzma path/to/file.lzma
```

- Decompress a file and write to `stdout` (implies `--keep`):

```
xz --decompress --stdout path/to/file.xz
```

- Compress a file, but don't delete the original:

```
xz --keep path/to/file
```

- Compress a file using the fastest compression:

```
xz -0 path/to/file
```

- Compress a file using the best compression:

```
xz -9 path/to/file
```

Security & User Management

- whoami - Check current user
- id - Show user and group IDs
- who - Show logged-in users
- passwd - Change user password
- sudo - Run commands as another user
- su - Switch to another user
- adduser - Create a new user
- deluser - Remove a user
- groups - Show groups of a user
- chmod - Modify file permissions
- chown - Change file ownership

whoami

> Display details about the current user. More information: https://learn.microsoft.com/windows-server/administration/windows-commands/whoami.

- Display the username of the current user:

```
whoami
```

- Display the groups that the current user is a member of:

```
whoami /groups
```

- Display the privileges of the current user:

```
whoami /priv
```

- Display the user principal name (UPN) of the current user:

```
whoami /upn
```

- Display the logon ID of the current user:

```
whoami /logonid
```

- Display all information for the current user:

```
whoami /all
```

id

> Display current user and group identity. More information: https://www.gnu.org/software/coreutils/manual/html_node/id-invocation.html.

- Display current user's ID (UID), group ID (GID) and groups to which they belong:

```
id
```

- Display the current user identity:

```
id -un
```

- Display the current user identity as a number:

```
id -u
```

- Display the current primary group identity:

```
id -gn
```

- Display the current primary group identity as a number:

```
id -g
```

- Display an arbitrary user's ID (UID), group ID (GID) and groups to which they belong:

```
id username
```

passwd

> Change a user's password. More information: https://manned.org/passwd.

- Change the password of the current user interactively:

```
passwd
```

- Change the password of a specific user:

```
passwd username
```

- Get the current status of the user:

```
passwd -S|--status
```

- Make the password of the account blank (it will set the named account passwordless):

```
passwd -d|--delete
```

sudo

> Executes a single command as the superuser or another user. More information: https://www.sudo.ws/sudo.html.

- Run a command as the superuser:

```
sudo less /var/log/syslog
```

- Edit a file as the superuser with your default editor:

```
sudo --edit /etc/fstab
```

- Run a command as another user and/or group:

```
sudo --user=user --group=group id -a
```

- Repeat the last command prefixed with `sudo` (only in Bash, Zsh, etc.):

```
sudo !!
```

- Launch the default shell with superuser privileges and run login-specific files (`.profile`, `.bash_profile`, etc.):

```
sudo --login
```

- Launch the default shell with superuser privileges without changing the environment:

```
sudo --shell
```

- Launch the default shell as the specified user, loading the user's environment and reading login-specific files (`.profile`, `.bash_profile`, etc.):

```
sudo --login --user=user
```

- List the allowed (and forbidden) commands for the invoking user:

```
sudo --list
```

su

> Switch shell to another user. More information: https://manned.org/su.

- Switch to superuser (requires the root password):

```
su
```

- Switch to a given user (requires the user's password):

```
su username
```

- Switch to a given user and simulate a full login shell:

```
su - username
```

- Execute a command as another user:

```
su - username -c "command"
```

adduser

> User addition utility. More information: https://manned.org/adduser.

- Create a new user with a default home directory and prompt the user to set a password:

```
adduser username
```

- Create a new user without a home directory:

```
adduser --no-create-home username
```

- Create a new user with a home directory at the specified path:

```
adduser --home path/to/home username
```

- Create a new user with the specified shell set as the login shell:

```
adduser --shell path/to/shell username
```

- Create a new user belonging to the specified group:

```
adduser --ingroup group username
```

deluser

> Delete a user from the system. More information: https://manned.org/deluser.

- Remove a user:

```
sudo deluser username
```

- Remove a user and their home directory:

```
sudo deluser --remove-home username
```

- Remove a user and their home, but backup their files into a `.tar.gz` file in the specified directory:

```
sudo deluser --backup-to path/to/backup_directory --remove-home username
```

- Remove a user, and all files owned by them:

```
sudo deluser --remove-all-files username
```

groups

> Print group memberships for a user. See also: `groupadd`, `groupdel`, `groupmod`. More information: https://www.gnu.org/software/coreutils/manual/html_node/groups-invocation.html.

- Print group memberships for the current user:

```
groups
```

- Print group memberships for a list of users:

```
groups username1 username2 ...
```

Advanced Usage & Debugging

- alias - Create command shortcuts
- unalias - Remove a command alias
- history - View command history
- time - Measure execution time of a command
- watch - Re-run a command at intervals
- tee - Redirect output to both a file and terminal
- xargs - Build and execute commands from input
- env - Show environment variables
- export - Set environment variables
- source - Reload configuration files

alias

> Create aliases - words that are replaced by a command string. Aliases expire with the current shell session unless defined in the shell's configuration file, e.g. `~/.bashrc`. More information: https://tldp.org/LDP/abs/html/aliases.html.

- List all aliases:

  ```
  alias
  ```

- Create a generic alias:

  ```
  alias word="command"
  ```

- View the command associated to a given alias:

  ```
  alias word
  ```

- Remove an aliased command:

  ```
  unalias word
  ```

- Turn `rm` into an interactive command:

  ```
  alias rm="rm --interactive"
  ```

- Create `la` as a shortcut for `ls --all`:

  ```
  alias la="ls --all"
  ```

unalias

> Remove aliases. More information: https://manned.org/unalias.

- Remove an alias:

```
unalias alias_name
```

- Remove all aliases:

```
unalias -a
```

history

> Command-line history. More information: https://www.gnu.org/software/bash/manual/html_node/Bash-History-Builtins.html.

- Display the commands history list with line numbers:

```
history
```

- Display the last 20 commands (in Zsh it displays all commands starting from the 20th):

```
history 20
```

- Display history with timestamps in different formats (only available in Zsh):

```
history -d|f|i|E
```

- **C**lear the commands history list (only for current Bash shell):

```
history -c
```

- Over**w**rite history file with history of current Bash shell (often combined with `history -c` to purge history):

```
history -w
```

- **D**elete the history entry at the specified offset:

```
history -d offset
```

time

> Measure how long a command took to run. Note: `time` can either exist as a shell builtin, a standalone program or both. More information: https://manned.org/time.

- Run the `command` and print the time measurements to `stdout`:

```
time command
```

- Create a very simple stopwatch (only works in Bash):

```
time read
```

watch

> Execute a program periodically and monitor the output in full-screen mode. More information: https://manned.org/watch.

- Repeatedly run a command and show the result:

```
watch command
```

- Re-run a command every 60 seconds:

```
watch --interval 60 command
```

- Monitor disk space, highlighting differences as they appear:

```
watch --differences df
```

- Repeatedly run a pipeline and show the result:

```
watch "command_1 | command_2 | command_3"
```

- Exit `watch` if the visible output changes:

```
watch --chgexit lsblk
```

- Interpret terminal control characters:

```
watch --color ls --color=always
```

tee

> Read from `stdin` and write to `stdout` and files (or commands). More information: https://www.gnu.org/software/coreutils/manual/html_node/tee-invocation.html.

- Copy `stdin` to each file, and also to `stdout`:

```
echo "example" | tee path/to/file
```

- Append to the given files, do not overwrite:

```
echo "example" | tee -a path/to/file
```

- Print `stdin` to the terminal, and also pipe it into another program for further processing:

```
echo "example" | tee /dev/tty | xargs printf "[%s]"
```

- Create a directory called "example", count the number of characters in "example" and write "example" to the terminal:

```
echo "example" | tee >(xargs mkdir) >(wc -c)
```

xargs

> Execute a command with piped arguments coming from another command, a file, etc. The input is treated as a single block of text and split into separate pieces on spaces, tabs, newlines and end-of-file. More information: https://pubs.opengroup.org/onlinepubs/9699919799/utilities/xargs.html.

- Run a command using the input data as arguments:

```
arguments_source | xargs command
```

- Run multiple chained commands on the input data:

```
arguments_source | xargs sh -c "command1 && command2 | command3"
```

- Gzip all files with `.log` extension taking advantage of multiple threads (`-print0` uses a null character to split file names, and `-0` uses it as delimiter):

```
find . -name '*.log' -print0 | xargs -0 -P 4 -n 1 gzip
```

- Execute the command once per argument:

```
arguments_source | xargs -n1 command
```

- Execute the command once for each input line, replacing any occurrences of the placeholder (here marked as `_`) with the input line:

```
arguments_source | xargs -I _ command _ optional_extra_arguments
```

- Parallel runs of up to `max-procs` processes at a time; the default is 1. If `max-procs` is 0, xargs will run as many processes as possible at a time:

```
arguments_source | xargs -P max-procs command
```

env

> Show the environment or run a program in a modified environment. More information: https://www.gnu.org/software/coreutils/manual/html_node/env-invocation.html.

- Show the environment:

```
env
```

- Run a program. Often used in scripts after the shebang (#!) for looking up the path to the program:

```
env program
```

- Clear the environment and run a program:

```
env -i program
```

- Remove variable from the environment and run a program:

```
env -u variable program
```

- Set a variable and run a program:

```
env variable=value program
```

- Set one or more variables and run a program:

```
env variable1=value variable2=value variable3=value program
```

export

> Export shell variables to child processes. More information: https://manned.org/export.1posix.

- Set an environment variable:

```
export VARIABLE=value
```

- Append a pathname to the environment variable `PATH`:

```
export PATH=$PATH:path/to/append
```

source

> Execute commands from a file in the current shell. More information: https://manned.org/source.

- Evaluate contents of a given file:

```
source path/to/file
```

- Evaluate contents of a given file (alternatively replacing `source` with `.`):

```
. path/to/file
```

Best Practices and Troubleshooting Tips

- Always check system resources before running large data jobs
- Use screen/tmux for long-running remote jobs
- Automate monitoring with cron jobs
- Optimize disk and memory usage to avoid slowdowns
- These strategies enhance efficiency and prevent system failures.