Common Linux Command-Line Tools (Portable Across Distros) ?

Below is a **practical, cross-distro** “core toolbox” of commands you’ll use constantly. I’ll group them by job, give a **plain meaning**, and include **examples** you can try.

> Notes:
>
> - Examples assume a typical Bash-like shell.
> - Many commands have lots of options—this is the “most useful baseline.”

---

1) Getting oriented (where am I? what’s here?)

`pwd` — print working directory

Shows your current directory.

```
pwd
# /home/alex/projects
```

`ls` — list directory contents

```
ls
ls -l        # long listing (permissions, owner, size, time)
ls -a        # include hidden files (dotfiles)
ls -lah      # long + all + human-readable sizes
```

`cd` — change directory

```
cd /etc
cd ..        # up one directory
cd ~         # home directory
cd -         # previous directory
```

`tree` — show directory tree (often needs installing)

```
tree
tree -L 2    # limit depth
```

---

2) Creating, moving, copying, deleting (file operations)

`touch` — create an empty file / update timestamp

```
touch notes.txt
```

`mkdir` — make directories

```
mkdir logs
mkdir -p a/b/c   # create parents as needed
```

`cp` — copy files/directories

```
cp a.txt b.txt
cp -r src/ backup-src/     # copy directory recursively
cp -a src/ backup-src/     # archive mode: preserve permissions/times (very common)
```

`mv` — move/rename files/directories

```
mv oldname.txt newname.txt
mv file.txt /tmp/
```

`rm` — remove files/directories (destructive)

```
rm file.txt
rm -r folder/       # recursive delete directory
rm -rf folder/      # force + recursive (use extreme caution)
```

`ln` — create links (symlinks are very common)

```
ln -s /var/www/site/current/public public   # symlink
```

---

3) Reading files quickly

`cat` — print file contents

Good for small files; for big ones prefer `less`.

```
cat /etc/hostname
```

`less` — page through text interactively

Keys: `q` quit, `/text` search, `n` next match.

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

`head` / `tail` — show start/end of file

```
head -n 20 access.log
tail -n 50 error.log
tail -f error.log     # follow appended log lines (real-time)
```

`nl` — show file with line numbers

```
nl -ba nginx.conf
```

---

4) Help & discovery

`man` — manual pages

```
man ls
man -k network    # search man page descriptions (keyword)
```

`--help` — quick built-in help (common convention)

```
grep --help
```

`which` / `type` — where a command comes from

```
which python
type ls            # also tells if it’s an alias/function/builtin
```

---

5) Searching text (the everyday “find inside files” tools)

`grep` — search text for patterns

```
grep "listen" nginx.conf
grep -R "DB_HOST" .           # recursive search
grep -n "error" app.log        # include line numbers
grep -i "warning" app.log      # case-insensitive
```

`sed` — stream editor (common for simple substitutions)

```
sed 's/http:/https:/g' urls.txt
sed -n '1,20p' file.txt        # print specific lines
```

sed

□□ `sed` is a **stream editor**: it reads text **one line at a time**, applies one or more editing "commands" you give it, and then (by default) **prints the result**. It's especially useful in pipelines because it can transform text *as it flows through*—without you opening an editor. □□

1. **How `sed` thinks (the mental model)**
   1. `sed` takes input from:
      - a file: `sed '...' file.txt`
      - *or* stdin (a pipe): `some_command | sed '...'`
   2. For each line, it:
      - loads the line into a temporary buffer (often called the *pattern space*)
      - runs your commands on that line
      - prints the line (unless you tell it not to)
2. **The most common command: substitution (`s/…/…/`)**
   The form is:

   ```
   sed 's/PATTERN/REPLACEMENT/FLAGS'
   ```

   Example from your notes:

   ```
   sed 's/http:/https:/g' urls.txt
   ```

   What it means:
   - `s` = "substitute"
   - `http:` = the text (or regex) to find
   - `https:` = what to replace it with
   - `g` = "global" on the line (replace **all** matches in that line, not just the first)

   Concrete example:
   - Input line: `http://a  http://b`
   - Without `g` (`sed 's/http:/https:/'`): `https://a  http://b`
   - With `g` (`sed 's/http:/https:/g'`): `https://a  https://b`
3. **Patterns are usually regular expressions (regex)**
   `sed` patterns are typically regex, so you can match more than literal text.
   - Replace any run of spaces with one space:

```
echo "a     b" | sed 's/  */ /g'
```

- Replace only at the start of the line using `^`:

```
echo "ERROR: bad" | sed 's/^ERROR:/WARN:/'
```

- Replace only at the end using `$`:

```
echo "file.tmp" | sed 's/\.tmp$/.txt/'
```

Notes:

- In basic `sed`, some regex operators need escaping (like `\+` instead of `+`). On many systems you can use extended regex with `-E`:

```
echo "a     b" | sed -E 's/ +/ /g'
```

4. **Printing vs suppressing output (`-n` and `p`)**
   By default, `sed` prints every (possibly modified) line. If you only want *some* lines, use `-n` (no auto-print) plus `p` (print).
   Your example:

```
sed -n '1,20p' file.txt
```

   Meaning:
   - `-n` = don't print everything automatically
   - `1,20p` = print only lines 1 through 20

   Similarly, to print only lines that match a pattern:

```
sed -n '/error/p' app.log
```

5. **Filtering/deleting lines (`d`)**
   Another very common use is deleting lines (i.e., filtering them out).
   - Delete blank lines:

```
sed '/^$/d' file.txt
```

   - Delete comment lines starting with `#`:

```
sed '/^#/d' file.txt
```

6. **In-place editing (`-i`)—powerful but easy to misuse**
   If you want to actually modify a file on disk (instead of just printing to the terminal), you can use `-i`.
   - Replace in a file *in place*:

```
sed -i 's/old/new/g' config.txt
```

Many people prefer making a backup:

```
sed -i.bak 's/old/new/g' config.txt
```

That will leave `config.txt.bak` behind as a safety net.

7. **Putting it in pipelines (why it's called a "stream" editor)**
   `sed` shines when transforming command output:

   ```
   ps aux | sed -n '1,5p'
   ```

   Or when cleaning data before feeding it to something else:

   ```
   cat access.log | sed 's/"//g' | cut -d' ' -f1 | sort | uniq -c | sort -nr
   ```

If you tell me what exactly felt unclear (e.g., the `s///g` syntax, regex characters, or `-n ... p`), I can tailor a few examples directly to your kind of files/logs.

`awk` — field/column processing

```
awk '{print $1, $9}' access.log   # e.g., IP and status code columns (depends on log format)
```

---

6) Counting, sorting, unique-ing (text pipelines)

`wc` — count lines/words/bytes

```
wc -l access.log     # number of lines
```

`sort` — sort lines

```
sort names.txt
sort -n numbers.txt  # numeric sort
```

`uniq` — collapse adjacent duplicates (usually used after `sort`)

```
sort users.txt | uniq
sort users.txt | uniq -c | sort -nr   # frequency count, descending
```

`cut` — extract columns/fields

```
cut -d: -f1 /etc/passwd     # usernames (field 1, delimiter ':')
```

`tr` — translate/delete characters

```
tr '[:lower:]' '[:upper:]' < file.txt
```

---

7) Composing commands (pipes, redirection, and “do this to each line”)

`|` (pipe) — send output of one command into another

```
ps aux | grep nginx
```

Redirection: `>`, `>>`, `2>`, `&>`

```
echo "hello" > out.txt      # overwrite
echo "more"  >> out.txt     # append
cmd 2> err.txt              # stderr to file
cmd &> all.txt              # stdout+stderr to file (bash)
```

`tee` — write output to file *and* keep showing it

```
echo "config" | tee -a notes.txt
```

`xargs` — turn input lines into arguments (batch operations)

```
find . -name "*.log" | xargs rm -f
# safer form when filenames may contain spaces:
find . -name "*.log" -print0 | xargs -0 rm -f
```

---

8) Finding files

`find` — search by name/type/time/size (very common)

```
find . -name "*.conf"
find /var/log -type f -mtime -7     # files modified in last 7 days
find . -type f -size +100M          # larger than 100MB
```

`locate` — fast filename search via database (needs updated index)

```
locate nginx.conf
```

---

9) Permissions & ownership (you'll use these constantly on servers)

`chmod` — change permissions

```
chmod 644 file.txt    # rw-r--r--
chmod 755 script.sh   # rwxr-xr-x
chmod -R 755 public/  # recursively (use carefully)
```

`chown` / `chgrp` — change owner / group

```
chown www-data:www-data -R /var/www/site
chgrp developers project/
```

`umask` — default permission mask for new files

```
umask
```

---

10) Processes & system state (baseline diagnostics)

`ps` — show processes

```
ps aux
ps aux | grep php-fpm
```

`top` / `htop` — live process viewer (`htop` may need install)

```
top
htop
```

`kill` / `pkill` — send signals to processes

```
kill 1234
pkill nginx
kill -TERM 1234   # graceful request
```

```
kill -KILL 1234   # force (last resort)
```

`systemctl` — manage services (systemd systems)

```
systemctl status nginx
systemctl restart nginx
systemctl enable nginx   # start on boot
```

`journalctl` — systemd logs

```
journalctl -u nginx
journalctl -u nginx -f       # follow
journalctl -b                # logs since boot
```

---

11) Networking essentials (developer/server workflow)

`ip` — network interfaces and routes

```
ip a          # addresses
ip r          # routes
```

`ss` — see listening ports and connections

```
ss -tulpn     # TCP/UDP listening + process names
```

`ping` — basic reachability test

```
ping -c 4 example.com
```

`curl` — make HTTP requests (huge for web debugging)

```
curl https://example.com
curl -I https://example.com          # headers only
curl -v https://example.com          # verbose (TLS + connection details)
```

`dig` / `nslookup` — DNS queries (often `dig`)

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

---

12) Archives & compression (moving code, backups)

`tar` — create/extract tar archives (the Linux standard)

```
tar -czf site.tar.gz site/     # create gzip-compressed archive
tar -xzf site.tar.gz           # extract
```

`gzip` / `xz` — compression tools (often used via `tar`)

```
gzip large.log
xz -T0 bigfile     # stronger compression; -T0 uses all cores
```

`zip` / `unzip` — common for cross-platform bundles

```
zip -r project.zip project/
unzip project.zip
```

---

13) Remote access & file transfer (core server skills)

`ssh` — remote shell

```
ssh user@server.example.com
ssh -i ~/.ssh/id_ed25519 user@server
```

`scp` — copy over SSH (simple, not always the best)

```
scp file.txt user@server:/tmp/
scp -r site/ user@server:/var/www/
```

`sftp` — interactive file transfer over SSH

```
sftp user@server
# then: put/get, ls, cd, etc.
```

---

14) Privilege escalation (admin work)

`sudo` — run as root (or another user)

```
sudo apt update
sudo systemctl restart nginx
```

---

15) “Environment” and shell basics you’ll constantly touch

`echo` — print text / variables

```
echo "Hello"
echo "$HOME"
```

`env` / `printenv` — show environment variables

```
env
printenv PATH
```

`export` — set environment variable for this shell session

```
export NODE_ENV=production
```

`history` — show command history

```
history | tail
```

If you want, I can turn this into a printable “cheat sheet”

Tell me your target context: **web server admin (WordPress)**, **Node/React dev**, or **general Linux**—and whether you’re on **Debian-like** or **RHEL-like** most often.

Revision #9
Created 2026-04-05 00:23:25 UTC by art10m
Updated 2026-04-05 17:22:05 UTC by art10m