Anki Cloze Cards — Common Linux Command-Line Tools ?

Below are **Anki cloze (deletion) notes** you can paste into Anki (note type: **Cloze**).
Each note has **1–4 blanks** and is designed to be *memorable*, *practical*, and *cross-distro*.
I also added a **few high-value extras** (e.g., `du`, `df`, `chmod` meaning, safer `find`, `rsync`, `ssh` tips) that fit perfectly with the topic.

---

1) Orientation & navigation ?

1. `pwd` prints the {{c1::current working directory}}.
2. `ls -l` shows a {{c1::long listing}} (permissions, owner, size, time).
3. `ls -a` includes {{c1::hidden files (dotfiles)}}.
4. `ls -lah` combines {{c1::-l}}, {{c2::-a}}, and {{c3::human-readable sizes (-h)}}.
5. `cd ..` goes {{c1::up one directory}}.
6. `cd ~` goes to your {{c1::home directory}}.
7. `cd -` switches to the {{c1::previous directory}}.
8. `tree -L 2` shows a directory tree limited to {{c1::depth 2}}. *(Often needs installing.)*
9. To display your username and hostname quickly: `{{c1::whoami}}` and `{{c2::hostname}}`. *(Extra, useful baseline.)*

---

2) Create / move / copy / delete (file ops) ?

10. `touch file.txt` {{c1::creates an empty file}} or {{c2::updates its timestamp}}.
11. `mkdir -p a/b/c` creates directories {{c1::including parents as needed}}.
12. `cp a.txt b.txt` {{c1::copies a file}}.
13. `cp -r src/ dst/` copies a directory {{c1::recursively}}.
14. `cp -a src/ dst/` uses {{c1::archive mode}} to preserve {{c2::permissions and timestamps}}.
15. `mv old new` {{c1::renames}} or {{c2::moves}} files/directories.
16. `rm -r folder/` deletes a directory {{c1::recursively}}.
17. `rm -rf folder/` means {{c1::recursive}} + {{c2::force}} → {{c3::dangerous (no prompts)}}.

18. `ln -s TARGET LINKNAME` creates a {{c1::symbolic link (symlink)}}.
19. A **hard link** shares the same {{c1::inode}} as the original file; a symlink stores a {{c2::path}}. *(Extra but core concept.)*
20. Use `rm -i` for {{c1::interactive confirmation}} (safer deletes). *(Extra.)*

---

3) Reading files fast ?

21. `cat` is best for {{c1::small files}}; for big files prefer {{c2::less}}.
22. In `less`, `q` {{c1::quits}}, `/text` {{c2::searches}}, and `n` jumps to {{c3::next match}}.
23. `head -n 20 file` shows the {{c1::first 20 lines}}.
24. `tail -n 50 file` shows the {{c1::last 50 lines}}.
25. `tail -f` {{c1::follows appended lines}} (great for logs).
26. `nl -ba file` shows line numbers; `-b a` means number {{c1::all lines}} (including blank). *(Good to remember.)*
27. Quick “watch a file change” tool: `{{c1::watch}} -n 1 'tail -n 20 file'`. *(Extra.)*

---

4) Help & discovery ?

28. `man ls` opens the command’s {{c1::manual page}}.
29. `man -k network` performs a {{c1::keyword search}} (apropos).
30. Many commands support `--help` for {{c1::quick usage info}}.
31. `which python` prints the {{c1::path}} to the executable found in `PATH`.
32. `type ls` can reveal whether `ls` is an {{c1::alias}}, {{c2::function}}, {{c3::builtin}}, or external command.
33. Use `{{c1::command -v}} name` for a portable “where is it?” check. *(Extra, common in scripts.)*

---

5) Searching inside files (grep/sed/awk) ?

34. `grep "listen" nginx.conf` searches for the {{c1::literal pattern}} in a file.
35. `grep -R "DB_HOST" .` searches {{c1::recursively}} in the current directory.
36. `grep -n "error" app.log` includes {{c1::line numbers}}.
37. `grep -i "warning" app.log` is {{c1::case-insensitive}}.
38. `grep -v PATTERN` prints lines that {{c1::do NOT match}}. *(Extra.)*
39. `grep -E` enables {{c1::extended regex}} (like `+`, `|`, `()` without heavy escaping). *(Extra.)*

40. `sed 's/OLD/NEW/g' file` does a substitution; `g` means {{c1::replace all matches per line}}.
41. `sed -n '1,20p' file` uses `-n` to {{c1::suppress auto-print}} and `p` to {{c2::print selected lines}}.
42. In regex, `^` matches {{c1::start of line}} and `\$` matches {{c2::end of line}}.
43. `sed '/^$/d' file` deletes lines that are {{c1::blank}}.
44. `sed -i.bak 's/old/new/g' file` edits {{c1::in place}} while keeping a {{c2::backup}}.
45. In `awk '{print \$1, \$9}'`, `\$1` is the {{c1::first field}} and `\$9` the {{c2::ninth field}}.
46. `awk -F: '{print \$1}' /etc/passwd` sets the field separator to {{c1::`:`}}. *(Extra, highly practical.)*
47. `awk 'NR==1{print}' file` uses `NR` as the {{c1::record (line) number}}. *(Extra.)*

---

6) Counting / sorting / unique (pipelines) ?

48. `wc -l file` counts {{c1::lines}}.
49. `sort -n numbers.txt` performs a {{c1::numeric sort}}.
50. `uniq` only removes {{c1::adjacent duplicates}} → often use it after {{c2::sort}}.
51. `uniq -c` prefixes each group with its {{c1::count}}.
52. `sort users.txt | uniq -c | sort -nr` gives a {{c1::frequency table}} sorted {{c2::descending}}.
53. `cut -d: -f1 /etc/passwd` uses delimiter {{c1::`:`}} and prints field {{c2::1}} (usernames).
54. `tr '[:lower:]' '[:upper:]'` converts {{c1::lowercase}} to {{c2::uppercase}}.
55. `tr -d '\r'` deletes {{c1::carriage returns}} (fix Windows CRLF). *(Extra, common pain point.)*

---

7) Composition: pipes, redirection, tee, xargs ?

56. The pipe `|` sends {{c1::stdout of one command}} into {{c2::stdin of another}}.
57. `>` {{c1::overwrites}} a file, while `>>` {{c2::appends}}.
58. `2>` redirects {{c1::stderr}} to a file.
59. `&>` redirects {{c1::stdout and stderr}} to the same file (Bash).
60. `tee -a file` {{c1::appends}} while still printing to the {{c2::terminal}}.
61. `xargs` turns {{c1::input lines}} into {{c2::command arguments}}.
62. Safer filenames: `find ... -print0 | xargs -0 ...` handles {{c1::spaces/newlines}} correctly.
63. Even safer: `find ... -exec cmd {} +` avoids some {{c1::xargs pitfalls}}. *(Extra.)*

8) Finding files (find/locate) ??

64. `find . -name "*.conf"` finds files by {{c1::name pattern}}.
65. `find /var/log -type f -mtime -7` finds regular files modified in the last {{c1::7 days}}.
66. `find . -type f -size +100M` finds files larger than {{c1::100 MB}}.
67. `locate nginx.conf` searches a {{c1::database index}} → results may be stale unless updated. *(Extra concept.)*
68. `find . -iname "*.jpg"` is case-{{c1::insensitive}} name matching. *(Extra.)*
69. `find . -maxdepth 1 -type f` limits search depth to {{c1::1}}. *(Extra.)*

9) Permissions & ownership ?

70. `chmod 644 file` corresponds to {{c1::rw-r–r–}}.
71. `chmod 755 script.sh` corresponds to {{c1::rwxr-xr-x}}.
72. In permissions, `r=4`, `w=2`, `x=1`, so `7` equals {{c1::rwx}}. *(Extra, core mental model.)*
73. `chmod -R 755 dir/` changes permissions {{c1::recursively}} (use carefully).
74. `chown user:group file` changes {{c1::owner}} and {{c2::group}}.
75. `chown -R www-data:www-data /var/www/site` applies ownership changes {{c1::recursively}}.
76. `umask` affects the {{c1::default permissions}} for newly created files/dirs.
77. Typical defaults: `umask 022` leads to new files being {{c1::644}} and dirs {{c2::755}} (before special cases). *(Extra.)*

10) Processes & system state ?

78. `ps aux` shows {{c1::all processes}} (common BSD-style format).
79. `top` provides a {{c1::live}} process view; `htop` is a more {{c2::interactive}} alternative (may need install).
80. `kill 1234` sends {{c1::SIGTERM}} by default (a polite request).
81. `kill -KILL 1234` sends {{c1::SIGKILL}} → cannot be {{c2::caught/ignored}} (last resort).
82. `pkill nginx` kills processes by {{c1::name/pattern}}.
83. `systemctl status nginx` shows {{c1::service status}} (systemd systems).
84. `systemctl enable nginx` enables start on {{c1::boot}}.
85. `journalctl -u nginx -f` follows logs for unit {{c1::nginx}} in {{c2::real time}}.
86. `journalctl -b` shows logs since the last {{c1::boot}}.
87. Check disk usage quickly: `{{c1::df -h}}` (filesystem free space) vs `{{c2::du -sh DIR}}` (directory size). *(Extra, very common.)*

11) Networking essentials ?

88. `ip a` shows {{c1::addresses/interfaces}}; `ip r` shows {{c2::routes}}.
89. `ss -tulpn` lists listening {{c1::TCP/UDP}} sockets with {{c2::process info}} (needs privileges for full details).
90. `ping -c 4 example.com` sends {{c1::4}} ICMP echo requests.
91. `curl -I URL` fetches only {{c1::HTTP headers}}.
92. `curl -v URL` enables {{c1::verbose}} output (connection/TLS details).
93. `dig example.com +short` returns a {{c1::short}} DNS answer list.
94. Quick port test: `{{c1::nc}} -vz host 443` (or `{{c2::curl}}` to an HTTP port). *(Extra.)*

---

12) Archives & compression ?

95. `tar -czf site.tar.gz site/` creates a {{c1::gzip-compressed}} tar archive.
96. `tar -xzf site.tar.gz` {{c1::extracts}} a gzip-compressed tar archive.
97. In `tar`, `c={{c1::create}}`, `x={{c2::extract}}`, `z={{c3::gzip}}`, `f={{c4::filename}}`.
98. `gzip large.log` compresses to {{c1::large.log.gz}}.
99. `xz -T0 bigfile` uses {{c1::all CPU cores}} (`-T0`) for compression.
.00. `zip -r project.zip project/` creates a zip archive {{c1::recursively}}.
.01. `unzip project.zip` {{c1::extracts}} a zip archive.

---

13) Remote access & transfer (SSH family) ?

.02. `ssh user@host` opens a {{c1::remote shell}} over SSH.
.03. `ssh -i ~/.ssh/id_ed25519 user@host` uses a specific {{c1::private key}}.
.04. `scp file user@host:/tmp/` copies a file over {{c1::SSH}}.
.05. `sftp user@host` provides an {{c1::interactive}} file-transfer session.
.06. Better large sync tool: `{{c1::rsync}} -avz SRC/ user@host:DST/` (preserves attrs, efficient). *(Extra, extremely practical.)*
.07. SSH config shortcut: define a host in `{{c1::~/.ssh/config}}` to avoid retyping usernames/keys. *(Extra.)*

---

14) Privilege escalation ??

.08. `sudo cmd` runs a command as {{c1::root}} (or another user).

.09. Use `sudo -u USER cmd` to run as {{c1::a specific user}}. *(Extra.)*
.10. Prefer editing protected files with `{{c1::sudoedit}}` rather than running a full editor as root. *(Extra safety.)*

---

15) Environment & shell basics ?

.11. `echo "\$HOME"` prints the value of the {{c1::HOME}} environment variable.
.12. `env` shows the process {{c1::environment}} (variables).
.13. `printenv PATH` prints the {{c1::PATH}} variable.
.14. `export VAR=value` sets an env var for the {{c1::current shell session}} and its {{c2::child processes}}.
.15. `history | tail` shows the {{c1::most recent}} commands.
.16. `VAR=value cmd` sets `VAR` only for {{c1::that command invocation}} (not permanently). *(Extra, very useful.)*

---

16) High-yield “extras” that pair well with the toolbox ?

.17. To run a command as another user without a shell: `sudo -u user {{c1::--}} cmd` prevents option confusion. *(Extra nuance.)*
.18. `chmod u+x script.sh` adds execute for the {{c1::user (owner)}}; symbolic modes can be clearer than octal. *(Extra.)*
.19. `ls -lt` sorts by {{c1::modification time}} (newest first). *(Extra.)*
.20. `sort -u` both sorts and removes duplicates in {{c1::one step}}. *(Extra.)*
.21. Safer grep in binaries: `grep -I` treats binary files as {{c1::non-matching}}. *(Extra.)*
.22. `curl -sS` means {{c1::silent}} but still show {{c2::errors}}. *(Extra.)*

Want these as a *single “deck-ready” TSV*? ?

Tell me your preferred import format:

1. **Plain cloze lines** (as above)
2. **TSV** with columns: `Text` + `Tags`
3. **Grouped by tags** (e.g., `linux::files`, `linux::network`, `linux::text`)

Revision #3
Created 2026-04-09 23:51:14 UTC by art10m
Updated 2026-04-10 02:55:20 UTC by art10m