UNIX SHELL ENGINEERING REFERENCE

Author: ddonle.com

Core Model

Three layers of wisdom:

- **UNIX** – everything is a file
- **Shell** – everything is a text stream
- **Script** – everything is a string

Layer	Examples
Hardware	CPU, disk, NIC
OS / Drivers	Linux, FreeBSD, Darwin
Shell	bash, zsh, fish
GUI / Apps	X11, executables

Options convention: `cmd [-h] [-v] [--opt=val] ARG`
Short: `-h -v` or combined `-al`. Long: `--help --verbose`.

Shell Interactivity

`cmd` – run a command.
`./script` – run script in a subshell (changes discarded).
`. script` or `source script` – run in current shell (changes kept).

`Ctrl-C`	stop running command
`Ctrl-Z`	pause (suspend) command
`fg`	foreground a paused command
`Ctrl-D`	exit shell / send EOF
`Ctrl-L`	clear screen
`Ctrl-R`	search command history
`Ctrl-U`	erase line before cursor
`Ctrl-W`	erase word before cursor
`Ctrl-←`	move one word left
`Ctrl-_`	undo last edit

Everyday File Commands

`ls -l` – long listing. `ls -a` – show hidden files.
`ls -h` – human-readable sizes. `ls -t` – sort by date. `ls -S` – sort by size.
`cd ..` – go up. `cd /` – filesystem root. `cd ~` – home.
`cp -r` – recursive copy. `cp -a` – archive (preserve metadata). `cp -i` – interactive.
`mv src dst` – move or rename.
`rm -r` – recursive. `rm -f` – force. `rm -i` – interactive.
`mkdir -p dir/sub` – create with parents.
`ln -s target link` – create symbolic link.
`rename "s/X/Y/" *.ext` – bulk rename via regex.
`chmod [ugoa][+-][rwx] file` – change permissions. `-R` for recursive.
`chown user[:group] file` – change owner. `-R` for recursive.

Viewing & Displaying Files

`cat file` – print entire file to stdout.
`less file` – paginated view. `q` quit, `/` search, `n` next match, arrows navigate.
`head -n 20 file` – first 20 lines.
`tail -n 20 file` – last 20 lines.
`tail -f file` – follow file as it grows (logs).
`wc -l file` – count lines. `wc -w` words. `wc -m` characters.

Text Streams

`cat A > B` – redirect (overwrite). `cat A >> B` – append.
`cat A B > C` – concatenate two files.
`cmd | cmd2` – pipe stdout of cmd into cmd2.

`cmd > out.txt`	redirect stdout
`cmd 2> err.log`	redirect stderr
`cmd &> all.txt`	redirect both
`cmd 2>&1 | ...`	redirect stderr into pipe
`cmd 2>/dev/null`	suppress errors
`... | tee -a out`	print and also redirect

`cat ~/.bashrc | grep alias | less` – pipeline example.

Filtering & Searching

`grep PATTERN file` – filter lines matching pattern.
`grep -v` – invert (exclude matching). `grep -E` – extended regex.
`grep -n` – show line numbers. `grep -A n` – n lines after.
`grep -B n` – n lines before.
`find . -name "x"` – find by name from current dir.
`find . -type f` – files only. `find . -type d` – dirs only.
`find ~ -name "*.sh" -exec ls -l {} \;` – exec on each found file.
`find /tmp -depth -name core -type f -delete` – delete found files.
`find -print0` – null-delimited output (safe for filenames with spaces).
`locate term` – fast filesystem name search (uses index).

Stream Manipulation

`sort file` – sort lines. `sort -r` reverse. `sort -n` numeric. `sort -f` ignore case.
`uniq` – remove consecutive duplicates (pipe after `sort`).
`cut -d: -f1` – extract field 1 using `:` as delimiter.
`cut -f2-4` – extract fields 2 through 4.
`sed "s/X/Y/g"` – replace all X with Y.
`sed "s,X,Y,g"` – alternate delimiter (useful when X contains /).
`sed -i "s/X/Y/g" file` – replace in-place.
`sed -e "s/A/B/" -e "s/C/D/"` – multiple substitutions.
`xargs cmd` – run cmd for each item of piped stream.
`xargs -L 1 cmd` – one item per invocation.
`xargs -I {} cmd {}` – use {} as placeholder.
Example – 10 most used commands:

```
history|cut -d" " -f5|sort|uniq -c|sort -rn|head -10
```

Chaining Commands

`A ; B`	run A then B unconditionally
`A && B`	run B only if A succeeds
`A || B`	run B only if A fails

Example – notify when build ends:

```
(cmake .. && make) ; zenity --info --text "Done"
```

Example – merge CSVs without duplicating headers:

```
head -n1 A.csv > out.csv && tail -n+2 -q *.csv >>
out.csv
```

Background & Detached Processes

`cmd &`	run in background (tied to session)
`Ctrl-Z, bg`	pause then push to background
`nohup cmd`	run detached (survives logout)

tmux – detachable terminal multiplexer:
`tmux` – start new session.
`Ctrl-b d` – detach from session.
`tmux attach` – reattach to session.

background = depends on the session.
detached = survives session end.

Process Management

`ps aux` – show all running processes.
`kill PID` – request process to stop (SIGTERM).
[DANGER] `kill -9 PID` – force kill (SIGKILL); cannot be caught.
`nice -n 10 cmd` – run with lower priority (+10).
`renice +5 PID` – change priority of running process.
`at 14:00` – schedule command to run at a time.
`sleep 5 && cmd` – wait 5 seconds before next command.
`time cmd` – measure wall clock and CPU time of cmd.

`$?`	exit code of last command (0 = OK)
`$!`	PID of last background process

Shell Script Basics

```
#!/bin/bash           # shebang: interpreter
x="hello"             # assign variable
printf "${x}\n"       # print variable
z="${x:-default}"     # use default if x unset
x=$(cmd)              # capture command output
x=$((1 + 2))          # integer arithmetic
```

`$0`	script name
`$1 $2`	positional arguments
`$*`	all arguments
`$#`	number of arguments
`$IFS`	Internal Field Separator

Arrays:

```
declare -a arr=()
arr+=("A" "B")
printf "${arr[0]}"   # access element
```

Conditions & Tests

```
if [[ "$x" == "$y" ]]; then ...  # string equal
if [[ "$x" -eq "$y" ]]; then ... # numeric equal
```

`-ne`	not equal (numeric)
`-gt / -ge`	greater than / or equal
`-lt / -le`	less than / or equal
`-n "$x"`	string is not empty
`-z "$x"`	string is empty
`-f file`	is a regular file
`-d file`	is a directory
`-r / -w / -x`	readable / writable / executable
`&& / || / !`	AND / OR / NOT

Loops

```
for x in $LIST; do   # iterate over split var
  echo "$x"
done

for ((i=0; i<10; i++)); do
  echo "$i"          # numeric counter
done

i=0
while [[ "$i" -ne 10 ]]; do
  i=$((i+1))
done
```

Read file line by line:

```
while IFS='' read -r line \
    || [[ -n "$line" ]]; do
  printf "$line\n"
done < "$filename"
```

Functions

```
alias f="cmd -opt"    # simple shortcut
function f() { ... }  # full function

f "A" "B"             # call function
if f "A"; then ...    # test exit code
```

Return values — by variable or stdout:

```
function g() {
  local str="${1}"
  if [[ -z "$str" ]]; then
    return 1          # non-zero = error
  fi
  ret="$str"          # return via variable
  printf "$str"       # return via stdout
  return 0
}
printf $(g "hello")   # capture stdout
if g "hi"; then
  printf "$ret"       # read variable
fi
```

Environment

`printenv` – show all environment variables.
`export KEY=value` – set and export variable to subprocesses.

`PATH`	where shell finds commands
`HOME`	user's home directory
`PWD`	current working directory
`LOGNAME`	current username
`SHELL`	current shell binary
`LANG`	locale / language
`EDITOR`	default text editor
`HISTIGNORE`	commands not saved to history

`export PATH="$PATH:/my/path"` – append to PATH.

~/.bashrc Configuration

`. ~/.bashrc` – reload config without restarting shell.

```
alias ..="cd .."
alias ll="ls -al"
alias lk="ls -lSr"          # sort by size

function md() {              # mkdir then cd
  mkdir $1 && cd $1
}

# Up/down arrows search history
bind '"\e[A": history-search-backward'
bind '"\e[B": history-search-forward'

alias upgrade="sudo apt update && \
  sudo apt dist-upgrade -y && \
  sudo apt --purge autoremove -y"

export PATH="$PATH:$HOME/.local/bin"
```

SSH & Remote

`ssh [user@]host [cmd]` – connect or run command remotely.
`ssh -X host` – enable X11 forwarding (graphical apps).
`ssh -L 8080:localhost:80 host` – set up a local tunnel.
`scp -r src user@host:/dst` – copy files over SSH.
`ssh-keygen` – generate SSH key pair.
`ssh-copy-id user@host` – install public key on remote.

~/.ssh/config example:

```
Host myserver
  Hostname example.com
  User alice
  IdentityFile ~/.ssh/id_ed25519

Host internal
  Hostname 192.168.1.10
  ProxyJump myserver
```

`ProxyJump` chains through a bastion host automatically.

Core Sysadmin

`sudo cmd` – execute as root.
`df -h` – disk free, human-readable.
`free -m` – free memory in MB.
`du -h -d 2 dir` – disk usage, 2 levels deep.
`ifconfig` – show ethernet interfaces.
`iwconfig` – show wifi interfaces.
`hostname` – show current hostname.
`curl URL` – download from web / call HTTP API.
`sha256sum file` – verify file fingerprint.
`time cmd` – measure execution time.
[DANGER] `rm -rf /` – destroys entire filesystem. Never run.

Archives & Compression

`tar cfz archive.tgz *` – create gzip-compressed tarball.
`tar xfz archive.tgz` – extract gzip tarball.
`tar ... j f.tar.bz2` – bzip2 compression.
`tar ... J f.tar.xz` – xz compression (best ratio).
`zip -r f.zip *` – create zip archive.
`unzip -d outdir f.zip` – extract zip to directory.
`dtrx archive` – smart extract any archive format.

Useful file utilities:
`date -I` – print sortable ISO date (YYYY-MM-DD).
`mktemp` – create a unique temporary file.
`mktemp -d` – create a unique temporary directory.

Wildcards & Globs

`*.txt`	files ending in `.txt`
`*A*`	files containing `A`
`A?.txt`	one character after A
`[abc].txt`	one of a, b, or c
`[0-9]*`	starts with a digit
`{a,b}.sh`	brace expansion: `a.sh` and `b.sh`

`**` – recursive glob (enable with `shopt -s globstar` in bash).

Additional Tools

`liquidprompt`	smart, informative shell prompt
`exa`	modern replacement for `ls`
`rsync`	efficient remote file sync
`colout`	add color to text streams
`autojump`	smart `cd` with frecency
`fzf`	fuzzy finder for files and history
`ripgrep (rg)`	fast modern replacement for `grep`
`bat`	`cat` with syntax highlighting
`fd`	fast modern replacement for `find`

Large-Scale Shell Practices

- Always quote variables: `"$var"` not `$var`.
- Use `[[ ]]` not `[ ]` for conditionals in bash.
- Use `set -euo pipefail` at top of scripts to fail fast.
- Prefer `printf` over `echo` for portable output.
- Use `local` inside functions to avoid polluting global scope.
- Use `tmux` or `screen` for any long-running remote tasks.
- Pin script interpreter explicitly with a shebang: `#!/usr/bin/env bash`.
- Use `shellcheck` to lint scripts before running in production.

Core concepts
Chaining / background
Processes / sysadmin
Remote / archives
Scripting / config
Files / workflow