Bash Scripting Reference

One-liners, idioms, and templates. Targeting Bash 5+. Set `set -euo pipefail` at the top of every script you write.

Script Skeleton

```
#!/usr/bin/env bash
set -euo pipefail
IFS=$'\n\t'

# Constants
readonly SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
readonly LOG="${SCRIPT_DIR}/run.log"

# Functions
log() { printf '[%s] %s\n' "$(date +%T)" "$*" | tee -a "$LOG"; }
die() { log "ERROR: $*"; exit 1; }

main() {
[[ $# -ge 1 ]] || die "usage: $0 <input>"
local input=$1
log "starting on $input"
# ... work ...
log "done"
}

main "$@"
```

Variables & Quoting

```
name="alice"
echo "Hello, $name" # always double-quote expansions
echo '$name' # single quotes = literal
echo "${name^^}" # uppercase → ALICE
echo "${name:0:3}" # substring → ali
echo "${file%.txt}" # strip suffix hello.txt → hello
echo "${path##*/}" # basename /a/b/c.txt → c.txt
echo "${path%/*}" # dirname /a/b/c.txt → /a/b
echo "${var:-default}" # default if unset
echo "${var:=default}" # set + use default

# Arrays
arr=(a b c)
echo "${arr[1]}" # b
echo "${#arr[@]}" # 3 — length
for x in "${arr[@]}"; do echo "$x"; done

# Associative array (Bash 4+)
declare -A m=([a]=1 [b]=2)
echo "${m[a]}"
```

Conditions & Loops

```
if [[ -f "$file" ]]; then ... ; fi # file exists
if [[ -d "$dir" ]]; then ... ; fi # dir exists
if [[ -z "$x" ]]; then ... ; fi # empty
if [[ "$a" == "$b" ]]; then ... ; fi # string equality
if [[ "$a" =~ ^[0-9]+$ ]]; then ... ; fi # regex
if (( a > b )); then ... ; fi # arithmetic

for f in *.log; do gzip "$f"; done
```

```
while IFS= read -r line; do
echo ">$line<"
done < file.txt

case "$1" in
start) ... ;;
stop|quit) ... ;;
*) die "unknown" ;;
esac
```

Useful One-Liners

`!!`	Re-run last command
`!$`	Last argument of previous command
`cd -`	Toggle to previous directory
`${PIPESTATUS[0]}`	Exit status of first command in pipe
`mapfile -t arr < file`	Read file lines into array
`readarray -t arr < <(cmd)`	Same, from process substitution
`diff <(cmd1) <(cmd2)`	Compare command outputs
`paste -d, <(seq 1 5) <(seq 6 10)`	CSV merge
`printf "%s\n" "${arr[@]}"`	Print array one per line
`shuf -i 1-100 -n 10`	Pick 10 random numbers 1–100
`jot -r 1 1 100`	Random number 1–100 (BSD)
`seq -w 1 5`	Zero-padded sequence: 1 2 3 4 5
`find . -name "*.bak" -delete`	Delete matching files
`find . -mtime -1 -newer ref`	Modified since reference file
`xargs -P 4 -I{} cmd {}`	Parallel: 4 jobs at once
`command -v foo >/dev/null`	Check if command exists
`trap cleanup EXIT`	Always run cleanup, even on error

Process Substitution & Redirection

```
cmd > file # stdout to file (overwrite)
cmd >> file # stdout to file (append)
cmd 2> err.log # stderr to file
cmd >file 2>&1 # both to same file
cmd &> file # bash shorthand for both
cmd < input # stdin from file
cmd <<< "string" # heredoc string as stdin

# Heredoc
cat <<EOF > config
server=$(hostname)
port=8080
EOF

# No expansion (literal $)
cat <<'EOF' > script
echo $HOME
EOF

# Process substitution
diff <(sort a) <(sort b)
tee >(gzip > out.gz) > /dev/null
```

Argument Parsing

```
# Positional
input=${1:?usage: $0 <input> [output]}
output=${2:-out.txt}

# getopts
```

```
while getopts ":hvf:" opt; do
case $opt in
h) usage ; exit 0 ;;
v) verbose=1 ;;
f) file=$OPTARG ;;
\?) die "unknown -$OPTARG" ;;
:) die "missing arg for -$OPTARG" ;;
esac
done
shift $((OPTIND-1))
```

Idioms & Gotchas

• Always quote variables: `"$var"` — unquoted breaks on spaces and globs.

• Use `[[ ]]` instead of `[ ]`: safer, supports regex and &&/||.

• Use `$( )` for command substitution, never backticks.

• `set -euo pipefail`: exit on error, unset vars are errors, pipe failures propagate.

• Run `shellcheck script.sh` on every script — catches 90% of common bugs.

• Test scripts with `bash -n script.sh` for syntax-check without execution.