Bash

Scripts, expansion, and everyday flow.

DevSheets

01 Run a Bash script

```
#!/usr/bin/env bash
name=Maya
printf 'Hello, %s!\n' "$name"
printf 'Bash: %s\n' "$BASH_VERSION"
exit 0
```

Save as hello.sh; run bash hello.sh. bash -n hello.sh checks syntax without running commands. Examples are independent Bash scripts, not portable sh. Zero exit status means success.

02 Variables & defaults

```
name="Maya Chen"
count=3
printf '%s: %s\n' "$name" "$count"
label=${title:-Untitled}
printf '%s\n' "$label"
unset count
```

Assignments have no spaces around =. ${var:-word} defaults unset or empty values; ${var-word} only defaults unset values. Scalar assignments store text by default.

03 Quoting & printf

```
name="Maya Chen"
literal='$name stays literal'
printf '%s\n' "$name"
printf '%s\n' "$literal"
printf '%s\n' '100% ready'
```

Double quotes expand variables without word splitting or globbing. Single quotes keep literal text. Quote expansions by default. Use a fixed printf format with values as arguments.

04 Script arguments

```
printf 'Script: %s\n' "$0"
printf 'Count: %s\n' "$#"
first=${1:-Guest}
printf 'First: %s\n' "$first"
for arg in "$@"; do
    printf 'Arg: %s\n' "$arg"
done
```

"$@" preserves each argument, including empty ones. "$*" joins them into one word. ${10} accesses argument ten. shift removes the first argument; check that one exists.

05 Tests & exit status

```
name=Maya
if [[ -n "$name" && "$name" == Maya ]]; then
    printf '%s\n' 'Name found'
fi
false; status=$?
printf 'Status: %s\n' "$status"
```

[[...]] is a Bash conditional: -n means nonempty, -z empty, -f regular file, -d directory. Quote the right side of == for a literal string. Capture $? before another command replaces it.

06 Integer arithmetic

```
count=3
((count += 1))
whole=$((7 / 2))
rest=$((7 % 2))
printf '%s\n' "$count" "$whole" "$rest"
```

$((...)) expands an integer. ((...)) returns status 0 for nonzero, 1 for zero. Division truncates. Use constants or validated numbers, never untrusted arithmetic text.

07 Branches & case

```
mode=dev
case "$mode" in
    dev|test) printf '%s\n' 'Local' ;;
    prod) printf '%s\n' 'Production' ;;
    *) printf '%s\n' 'Unknown' >&2 ;;
esac
```

case matches shell patterns; | separates alternatives and * catches anything. ;; ends a branch. if/elif/else branches on command status, including tests, and ends with fi.

08 Loops

```
for name in Maya Leo; do
    printf '%s\n' "$name"
done
left=2
while ((left > 0)); do
    left=$((left - 1))
done
```

for visits words or array items. while repeats on success; until repeats on failure. break exits a loop; continue skips ahead. Quote expanded values to preserve each item.

09 Functions & local values

```
greet() {
    local name=${1:-Guest}
    printf 'Hello, %s!\n' "$name"
}
greet "Maya Chen"
```

Functions use their own positional arguments. local limits variable scope inside functions. return reports status (0-255); use stdout for text results. exit ends the script.

Bash

Collections, commands, and safe I/O.

10 Indexed arrays

```
names=("Maya Chen" "Leo")
names+=("Sam")
printf 'Count: %s\n' "${#names[@]}"
printf 'First: %s\n' "${names[0]}"
printf '%s\n' "${names[@]}"
```

Arrays preserve separate elements. "${names[@]}" expands one word per item, including empty strings. unset 'names[1]' removes an element without reindexing the others.

11 Command substitution

```
if stamp=$(date +%F); then
    printf 'Today: %s\n' "$stamp"
else
    printf '%s\n' 'date failed' >&2
    exit 1
fi
```

$(command) captures stdout and removes trailing newlines. Quote its result. Assignment alone reports the substitution's status. Do not use substitution to split lists of filenames.

12 Pipelines & failures

```
set -o pipefail
if printf '%s\n' red blue | grep red; then
    printf '%s\n' 'Matched'
else
    status=$?
    printf 'Pipeline: %s\n' "$status" >&2
fi
```

pipefail reports the rightmost failure, or zero. Normally only the last command decides. Check failures explicitly; set -e has context-dependent exceptions.

13 Redirect input & output

```
printf '%s\n' 'standard output'
printf '%s\n' 'diagnostic' >&2
if printf '%s\n' quiet > /dev/null; then
    printf '%s\n' 'Write succeeded'
fi
```

> file creates/truncates; >> file appends; < file supplies input. 2> redirects stderr. > log 2>&1 sends both streams to log. Order matters: 2>&1 copies stdout's current destination.

14 Environment & scope

```
app_mode=dev
export APP_MODE="$app_mode"
bash -c 'printf "%s\n" "$APP_MODE"'
unset APP_MODE
printf '%s\n' "$app_mode"
```

export passes values to children. NAME=value command sets one command's environment. Children cannot change parent variables. source runs a file in this shell: trust it.

15 Read text safely

```
while IFS= read -r line ||
      [[ -n "$line" ]]; do
    printf '<%s>\n' "$line"
done <<'TEXT'
  spaced text
backslash: \n
TEXT
```

IFS= keeps whitespace; read -r keeps backslashes. The extra test handles a final unterminated line. Quoting the here-document delimiter prevents body expansion.

16 Globs & filenames

```
for path in ./*; do
    if [[ -f "$path" ]]; then
        printf '%s\0' "$path"
    fi
done
```

Globs and quoted variables preserve spaces/newlines. ./* skips dotfiles and prefixes dashes. NUL separates names safely. Use -- before paths when supported.

17 Temporary files & cleanup

```
work_dir=$(mktemp -d) || exit 1
trap 'rm -rf -- "$work_dir"' EXIT
note_file=$work_dir/note
printf 'draft\n' > "$note_file" || exit 1
cat -- "$note_file" || exit 1
```

This script removes only its newly created temporary directory. Keep that path unchanged. EXIT runs on normal shell exit; traps cannot handle SIGKILL. Check resource creation and writes explicitly.

18 Parse short options

```
verbose=false
while getopts ':v' flag; do
    case "$flag" in
        v) verbose=true ;;
        *) exit 2 ;;
    esac
done
```

Run with -v. A leading : enables silent error handling. v: would require an argument in $OPTARG. After parsing, shift "$((OPTIND - 1))" exposes remaining positional arguments.