



# Your First Script – Shebang

A **shebang** tells the system which interpreter to run.

```
#!/usr/bin/env bash  
echo "Hello from a script"
```

Save as **hello.sh**, make executable, then run:

```
chmod +x hello.sh  
./hello.sh
```

**/usr/bin/env bash** finds **bash** via your PATH.

---

# Safer Defaults for Scripts

Use defensive options to catch errors early.

```
#!/usr/bin/env bash
set -Eeu -o pipefail
IFS=$'
```

- `-e` → exit on error
  - `-u` → treat unset variables as errors
  - `-o pipefail` → fail a pipeline if any command fails
  - `IFS` set to safe defaults for word-splitting
-

# Variables and Expansion

Assign and use variables:

```
name="Alice"  
echo "Hello, $name"
```

Braced expansion avoids ambiguity:

```
echo "File: ${name}.txt"
```

Length of a variable:

```
echo "Name length: ${#name}"
```

Default values and replacement:

```
echo "${user:-anonymous}"      # default if unset/empty  
echo "${path//:/ }"            # replace all ':' with space
```

Command substitution:

```
today="$(date +%F)"  
echo "Today is $today"
```

# Quoting Rules

Single quotes **prevent** expansion; double quotes **allow** it.

```
echo '$HOME literally'
echo "HOME is $HOME"
```

Use quotes to keep whitespace intact and avoid globbing surprises.

---

# Arithmetic

Use `(( ))` for integer arithmetic.

```
a=5; b=7
(( sum = a + b ))
echo "$sum"
```

Exit status of arithmetic context:

```
if (( sum > 10 )); then echo "big"; fi
```

---



## Case Statements

```
read -r ans
case "$ans" in
  yes|y|Y) echo "proceed" ;;
  no|n|N)  echo "abort" ;;
  *)      echo "unknown" ;;
esac
```

Great for parsing modes and flags.

---



# Loops – for / while / until

```
for i in 1 2 3; do echo "$i"; done
```

Read a file line-by-line safely:

```
while IFS= read -r line; do  
    printf '%s  
' "$line"  
done < input.txt
```

Repeat until condition becomes true:

```
count=0  
until (( count >= 3 )); do  
    echo "$count"  
    ((count++))  
done
```

---

# Functions, Scope, and Exit Codes

Define functions and return statuses.

```
greet() { printf 'Hi, %s\n' "$1"; }  
  
greet "Alice"
```

Check exit codes via `$?` or with `||` and `&&`:

```
cp source.txt dest.txt && echo "ok" || echo "copy failed"
```

Local variables:

```
sum() { local a="$1" b="$2"; echo $((a+b)); }
```

---

## Positional Parameters and \$@

```
# script.sh
echo "Script: $0"
echo "Arg count: $#"  
for arg in "$@"; do  
    echo "-> $arg"  
done
```

Quote "\$@" to preserve args with spaces.

---

# Reading Input – read and getopt

Read a line from stdin:

```
read -r name
echo "Hello, $name"
```

Parse flags with `getopts`:

```
#!/usr/bin/env bash
while getopts ":f:n:" opt; do
  case "$opt" in
    f) file="$OPTARG" ;;
    n) name="$OPTARG" ;;
    *) echo "Usage: $0 -f FILE -n NAME" >&2; exit 2 ;;
  esac
done
echo "file=$file name=$name"
```

---

# Arrays and Associative Arrays

Indexed arrays:

```
nums=(10 20 30)
echo "${nums[1]}"
echo "${#nums[@]}"      # length
echo "${nums[@]}"       # iterate values
```

Associative arrays (Bash 4+):

```
declare -A user=( [name]="Alice" [role]="admin" )
echo "${user[name]}"
for k in "${!user[@]}"; do echo "$k=${user[$k]}"; done
```

---

## printf – Safer Output

Prefer `printf` over `echo` for predictable formatting.

```
printf 'User: %s  Score: %03d\n' "Alice" 7
```

---

# Traps and Cleanup

Run code on exit or on specific signals.

```
tmp="$(mktemp -d)"
cleanup(){ rm -rf "$tmp"; }
trap cleanup EXIT INT TERM

echo "Working in $tmp"
sleep 1
```

This ensures resources are cleaned up even if interrupted.

---

# Here-Docs and Process Substitution

Here-doc to create files inline:

```
cat <<'EOF' > script.sh
#!/usr/bin/env bash
echo "inline"
EOF
```

Process substitution to compare streams without temp files:

```
diff <(sort a.txt) <(sort b.txt)
```

---



# Subshells vs Current Shell

Subshell inherits environment but not variable changes back to parent.

```
pwd; (cd /tmp; pwd); pwd
```

Use `{ }` for grouping without subshell:

```
{ echo one; echo two; } > out.txt
```

---

# Error Handling Patterns

Immediate exit on failure (already enabled via `set -e`).  
For guarded steps, use `||` with messages:

```
mkdir -p /data || { echo "cannot create /data" >&2; exit 1; }
```

Time a block:

```
start=$(date +%s); sleep 1; end=$(date +%s); echo "took $((end-start))s"
```

---

# Reusable Script Template

```
#!/usr/bin/env bash
set -Eeuo pipefail
IFS=$'\n'

usage(){ echo "Usage: $0 -i INPUT -o OUTPUT"; }

input="" output=""
while getopts ":i:o:" opt; do
    case "$opt" in
        i) input="$OPTARG" ;;
        o) output="$OPTARG" ;;
        *) usage; exit 2 ;;
    esac
done

[[ -f "$input" && -n "$output" ]] || { usage; exit 2; }

tmp="$(mktemp)"; trap 'rm -f "$tmp"' EXIT
cp "$input" "$tmp"
printf 'Processed %s -> %s\n' "$input" "$output"
mv "$tmp" "$output"
```

## Testing and Formatting

Lint scripts with **shellcheck**:

```
shellcheck script.sh
```

Auto-format with **shfmt**:

```
shfmt -w script.sh
```

Install via your package manager or from upstream releases.

---

## Recap

- Shebang and safe defaults make scripts portable and robust
  - Quoting, expansion, and command substitution are fundamentals
  - Use control flow, functions, and arrays for structure
  - Parse input with `getopts`; clean up with `trap`
  - Process substitution and here-docs enable elegant workflows
  - Use `shellcheck` and `shfmt` to maintain quality
-