Comprehensive Guide to Taking Input in Unix/Linux Shell Scripting (Bash)

This is a **complete, detailed reference** on how to accept and handle user input in Bash shell scripts. We'll cover everything from basic interactive prompts to advanced command-line argument parsing, reading from files/stdin, and secure input handling.

Everything here works in **Bash 3+** unless noted otherwise. I'll highlight portable options (for `sh`/dash) and best practices.

---

1. Interactive Input with `read` Command

The `read` builtin is the primary way to prompt the user and store input in variables.

A. Basic Syntax

```
Bash
read [options] variable_name1 variable_name2 ...
```

- If no variable is given, input goes to `$REPLY`.
- Reads from stdin (keyboard by default).

Simple example:

```
Bash
#!/bin/bash

echo "What is your name?"
read name
echo "Hello, $name!"
```

B. Common `read` Options

Option	Meaning	Example
`-p "prompt"`	Display prompt without newline	`read -p "Enter age: " age`
`-r`	Raw input (don't treat `\` as escape)	`read -r line` (important for paths)
`-s`	Silent (hide input – great for passwords)	`read -s -p "Password: " pass`
`-n N`	Read exactly N characters	`read -n 1 -p "Yes/No? " ans`
`-t N`	Timeout after N seconds (Bash 4+)	`read -t 10 -p "Enter in 10s: " input`
`-a array`	Read into indexed array	`read -a colors` → `$colors[0]` , etc.
`-d delim`	Use delimiter instead of newline	`read -d ':' user host`
`-e`	Use readline (arrow keys, history – Bash)	`read -e -p "Path: " path`
`-i default`	Pre-fill with default text (Bash 4+, readline)	`read -e -i "default" -p "Edit: " var`

C. Reading Multiple Variables

```
Bash
read -p "Enter first and last name: " first last
echo "Welcome, $first $last"
```

- Words are split by `$IFS` (default: space/tab/newline).

D. Secure Password Input

```
Bash

#!/bin/bash

read -s -p "Enter password: " password
echo   # newline after hidden input
echo "Password stored (not shown)."
# Do NOT echo $password in production!
```

E. Timeout Example (Bash 4+)

```
Bash

if read -t 5 -p "Press Enter in 5 seconds (or timeout): " ; then
    echo "You pressed Enter!"
else
    echo "Timed out."
fi
```

F. Reading into Arrays

```
Bash

read -p "Enter colors (space-separated): " -a colors
echo "You like: ${colors[0]}, ${colors[1]}, etc."
echo "All: ${colors[@]}"
echo "Count: ${#colors[@]}"
```

G. Reading Lines from File or Stdin

```
Bash

# From file
while IFS= read -r line; do
    echo "Line: $line"
done < /etc/passwd

# From piped input
echo "hello\nworld" | while IFS= read -r line; do
    echo "→ $line"
done
```

Pitfall: In pipelines, the `while` loop runs in a subshell → variables not preserved. Fix:

```
Bash

while IFS= read -r line; do
    echo "$line"
done < <(some_command)   # process substitution
```

---

2. Command-Line Arguments (Positional Parameters)

Scripts can accept arguments when invoked: `./script.sh arg1 arg2 "arg with space"`

A. Special Parameters

Parameter	Meaning	Example
`$0`	Script name	`echo "Running $0"`
`$1 .. $9`	First 9 arguments	`echo "First: $1"`
`${10}`	10th and beyond (use braces)	`echo "Tenth: ${10}"`
`$@`	All arguments as separate words	`for arg in "$@"; do ...`
`$*`	All arguments as single word	Rarely used
`$#`	Number of arguments	`if (( $# < 2 )); then echo "Need 2 args"; fi`
`$?`	Exit status of last command	Not input, but useful after commands
`$$`	Process ID	`echo "PID: $$"`

B. Safe Iteration Over Arguments

```bash
#!/bin/bash

echo "Script: $0"
echo "Arg count: $#"

for arg in "$@"; do   # Use "$@" not $*
    echo "Arg: $arg"
done
```

C. Shifting Arguments

```bash
echo "First: $1"
shift   # discard $1, $2 becomes $1, etc.
echo "Now first: $1"
shift 3 # discard next 3
```

Useful for processing options first.

---

3. Parsing Options/Flags (getopts)

For scripts with `-a -b --long` style options.

A. Basic `getopts`

```Bash
#!/bin/bash

while getopts "a:b:c" opt; do
    case $opt in
        a) echo "Option -a with value: $OPTARG" ;;
        b) echo "Flag -b" ;;
        c) echo "Flag -c" ;;
        ?) echo "Invalid option: -$OPTARG" ; exit 1 ;;
    esac
done

shift $((OPTIND-1))  # Move past processed options

echo "Remaining args: $@"
```

- `a:` → requires argument
- No colon → flag only
- `OPTARG` → value of option
- `OPTIND` → index of next argument

Usage: `./script.sh -a hello -b -c extra_arg`

B. Long Options? (Use manual parsing or `getopt`)

`getopts` doesn't support `--long` . Common manual way:

```bash
#!/bin/bash

while [[ $# -gt 0 ]]; do
    case $1 in
        -h|--help)    echo "Help message"; exit 0 ;;
        -f|--file)    file="$2"; shift 2 ;;
        --)           shift; break ;;  # end of options
        -* )          echo "Unknown option $1"; exit 1 ;;
        *)            positional+=("$1"); shift ;;
    esac
done

echo "File: $file"
echo "Other args: ${positional[@]}"
```

---

4. Advanced Input Techniques

A. Here Documents / Here Strings

```bash
cat << EOF
Multi-line
input
EOF

var=$(cat <<< "single line here string")
```

B. Default Values

```bash
read -p "Enter name [default: Guest]: " name
name=${name:-Guest}   # use default if empty
```

Or with readline default (Bash 4+):

```bash
read -e -i "default_value" -p "Prompt: " var
```

C. Validation Loop

```bash
while true; do
    read -p "Enter number 1-10: " num
    if [[ "$num" =~ ^[0-9]+$ ]] && (( num >=1 && num <=10 )); then
        break
    else
        echo "Invalid! Try again."
    fi
done
```

---

5. Best Practices & Common Pitfalls

- Always quote variables: `"$var"`
- Use `-r` with `read` when input might contain backslashes.
- Use `"$@"` for arguments, never `$*` unless intentional.
- For passwords: use `-s`, never echo them.
- Validate input! Never trust user data.
- For portable scripts: stick to basic `read` and positional params.
- Enable strict mode:

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

- Debug: `set -x` to trace.

This covers **every major way** to take input in Bash scripting — from simple prompts to robust CLI argument parsing.

If you have a specific script you're building (e.g., a menu, config file reader, or argument-heavy tool), share the details or code — I'll help you implement the perfect input handling! 😊