POSIX Shell Scripting Cheatsheet

A quick reference guide to POSIX shell scripting, covering syntax, commands, and best practices for writing portable and robust shell scripts.

Basic Syntax & Structure

Script Structure

`#!/bin/sh` - Shebang line, specifies the interpreter.

This line should be the first line of the script. It tells the system which interpreter to use to execute the script. Using `/bin/sh` ensures POSIX compliance.

`# Comment` - Comments start with a `#`.

Comments are used to explain the code and are ignored by the interpreter.

Commands are executed sequentially, one per line.

Each line typically contains a single command or a control structure.

Semicolons (`;`) can separate multiple commands on a single line.

Example:
`command1; command2`

Use `exit n` to exit the script with status `n`.

A status of `0` usually indicates success, while a non-zero status indicates failure.

Variables

Variable Assignment	`variable=value` (No spaces around `=`). **Example:** `name="John Doe"`
Variable Access	`$variable` or `${variable}` (safer). **Example:** `echo "Hello, $name!"`

Read-only Variables	`readonly variable`
	
	Example:
	`readonly name`

Unsetting Variables	`unset variable`
	
	Example:
	`unset name`

Special Variables	$0: Script name $1, $2, ...: Arguments $#: Number of arguments $?: Exit status of last command $$: Process ID $!: PID of last background command

Input and Output

`echo message` - Prints a message to standard output.

Example:
`echo "Hello, world!"`

`read variable` - Reads input from standard input and assigns it to a variable.

Example:
`read name`

`cat filename` - Displays the content of a file.

Example:
`cat myfile.txt`

`printf format arguments` - Formatted output (like C's printf).

Example:
`printf "Name: %s, Age: %d\n" "John" 30`

Control Structures

Conditional Statements (if/then/else/fi)

`if condition; then commands [elif condition; then commands] [else commands] fi`

Example:

```
if [ "$name" = "John" ]; then
  echo "Hello, John!"
else
  echo "Hello, stranger!"
fi
```

Conditions are often enclosed in square brackets `[ ]`. Note the spaces around the brackets and the condition.

Example:
`[ -f "myfile.txt" ]` (checks if the file exists)

String comparison: `=` (equal), `!=` (not equal)
Integer comparison: `-eq` (equal), `-ne` (not equal), `-lt` (less than), `-le` (less than or equal), `-gt` (greater than), `-ge` (greater than or equal)
File tests: `-f` (file exists), `-d` (directory exists), `-r` (readable), `-w` (writable), `-x` (executable)

Looping (for/while/until)

`for variable in word1 word2 ...; do commands done`

Example:

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

`while condition; do commands done`

Example:

```
i=1
while [ $i -le 3 ]; do
  echo "Number: $i"
  i=$((i + 1))
done
```

`until condition; do commands done`

Example:

```
i=1
until [ $i -gt 3 ]; do
  echo "Number: $i"
  i=$((i + 1))
done
```

`break` - Exits the loop.
`continue` - Skips the current iteration.

Case Statements

`case variable in pattern1) commands ;; pattern2) commands ;; *) commands ;; esac`

Example:

```
case "$1" in
  start) echo "Starting service" ;;
  stop) echo "Stopping service" ;;
  *) echo "Usage: $0 {start|stop}" ;;
esac
```

The `*)` pattern is the default case, similar to `default` in other languages.

Commands and Utilities

File Manipulation

`ls`	List directory contents
`mkdir directory`	Create a directory
`rm file`	Remove a file
`rmdir directory`	Remove an empty directory
`cp source destination`	Copy a file
`mv source destination`	Move or rename a file
`touch file`	Create an empty file or update its timestamp

Text Processing

`grep pattern file`	Search for a pattern in a file
`sed 's/old/new/g' file`	Replace text in a file
`awk '{print $1}' file`	Print the first field of each line in a file
`sort file`	Sort the lines in a file
`uniq file`	Remove duplicate lines from a file
`cut -d',' -f1 file`	Cut out sections of each line of a file

Process Control

`ps`	List running processes
`kill pid`	Terminate a process
`sleep seconds`	Pause execution for a specified number of seconds
command &	Run a command in the background
`wait`	Wait for all background processes to complete

Functions and Advanced Features

Functions

`function_name() { commands }` or `function function_name { commands }`

Example:

```
my_function() {
  echo "Hello from my_function!"
}

my_function # Call the function
```

Functions can accept arguments: `$1`, `$2`, etc.

Example:

```
greet() {
  echo "Hello, $1!"
}

greet "John"
```

`return value` - Returns a value from the function. The value should be between 0 and 255.

Local variables can be declared using `local`.

Example:

```
my_function() {
  local my_var="local value"
  echo $my_var
}
```

Command Substitution

`$(command)` or `command` (deprecated) - Executes a command and substitutes its output.

Example:

```
date_str=$(date +%Y-%m-%d)
echo "Today is $date_str"
```

Here Documents

`<<DELIMITER text DELIMITER` - Redirects multiple lines of input to a command.

Example:

```
cat <<EOF
Hello, this is a multi-line string.
EOF
```

Signal Handling

`trap 'command' SIGNAL` - Executes a command when a signal is received.

Example:

```
trap 'echo "Exiting..." ; exit 1' SIGINT
```

Common signals: `SIGINT` (Ctrl+C), `SIGTERM` (termination signal), `SIGKILL` (kill signal, cannot be trapped).