Bash Scripting Cheatsheet

A concise reference for Bash scripting, covering essential syntax, commands, and best practices.

Bash Basics

Script Structure

- `#!/bin/bash` - Shebang, specifies the interpreter.
  It's important to include this at the beginning of your bash scripts.
- `# Comments` - Use `#` for single-line comments.
- `exit 0` - Exit the script with a success code (0).

Variables

`variable="value"`	Assign a value to a variable. No spaces around `=`.
`$variable` or `${variable}`	Access the value of a variable. Use `${}` for clarity.
`readonly variable`	Make a variable read-only.
`unset variable`	Unset a variable.
`export variable`	Export a variable to the environment.

Input/Output

`echo "message"`	Print a message to standard output.
`read variable`	Read input from standard input into a variable.
`cat file`	Display the contents of a file.
`> file`	Redirect output to a file (overwrite).
`>> file`	Redirect output to a file (append).

Control Flow

Conditional Statements

```
if [ condition ]; then
  # commands
elif [ condition ]; then
  # more commands
else
  # default commands
fi
```

Example:

```
if [ "$1" == "start" ]; then
  echo "Starting service..."
elif [ "$1" == "stop" ]; then
  echo "Stopping service..."
else
  echo "Invalid command"
fi
```

Looping

For loop:

```
for variable in item1 item2 ... itemN;
do
  # commands
done
```

Example:

```
for i in 1 2 3 4 5; do
  echo "Number: $i"
done
```

While loop:

```
while [ condition ]; do
  # commands
done
```

Example:

```
count=0
while [ $count -lt 5 ]; do
  echo "Count: $count"
  count=$((count + 1))
done
```

Until loop:

```
until [ condition ]; do
  # commands
done
```

Case Statements

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

Example:

```
case "$1" in
  start )
    echo "Starting service..."
    ;;
  stop )
    echo "Stopping service..."
    ;;
  * )
    echo "Invalid command"
    ;;
esac
```

Functions

Defining Functions

`function function_name() { # commands }`	Defines a function named `function_name`.
Or: `function_name() { # commands }`	
`return value`	Returns a value from the function. The value must be between 0 and 255.

Calling Functions

`function_name arg1 arg2`	Calls the function `function_name` with arguments `arg1` and `arg2`.
`$1`, `$2`, etc.	Access the arguments passed to the function.
`$#`	Number of arguments passed to the function.
`$@`	All arguments passed to the function.

Example Function

```
function greet() {
  echo "Hello, $1!"
  return 0
}

greet "World"
```

Command Line Arguments

Accessing Arguments

`$0`	The name of the script.
`$1`, `$2`, etc.	The first, second, etc., command-line argument.
`$#`	The number of command-line arguments.
`$@`	All the command-line arguments as a list.
`$*`	All the command-line arguments as a single string.

Argument Parsing

```
while [[ $# -gt 0 ]]; do
  case "$1" in
    -a|--argument1 )
      argument1_value="$2"
      shift 2
      ;;
    -b|--argument2 )
      argument2_value="$2"
      shift 2
      ;;
    * )
      echo "Unknown argument: $1"
      exit 1
      ;;
  esac
done
```

A common pattern for parsing command-line arguments.