Shell Scripting Cheatsheet

A quick reference guide to shell scripting, covering essential syntax, commands, and best practices for automating tasks in Unix-like environments.

Basic Syntax & Structure

Script Structure

Every shell script starts with a shebang line to specify the interpreter:
`#!/bin/bash`

Followed by comments, variables, and commands.

Comments are denoted by `#`.

Example:
```
# This is a comment
```

Variables are assigned using `=` without spaces.

Example:
```
NAME="John Doe"
echo $NAME
```

Variables

`$VAR` or `${VAR}`	Accessing a variable's value. `${VAR}` is useful for variable expansion. **Example:** `echo "Hello, ${NAME}!"`
`$0`	Name of the script.
`$1, $2, ...`	Arguments passed to the script.
`$#`	Number of arguments passed to the script.
`$@` or `$*`	All arguments as a single string or separate words.
`$?`	Exit status of the last executed command.

Input/Output

`echo` - Displays text.

Example:
`echo "Hello, world!"`

`read` - Reads input from the user.

Example:
`read -p "Enter your name: " NAME`

`>` - Redirects output to a file (overwrites).

Example:
`echo "Hello" > file.txt`

`>>` - Redirects output to a file (appends).

Example:
`echo "Hello" >> file.txt`

`<` - Redirects input from a file.

Example:
`wc -l < file.txt`

Control Flow

Conditional Statements

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

Example:

```
if [ $AGE -gt 18 ]; then
  echo "Adult"
elif [ $AGE -lt 13 ]; then
  echo "Child"
else
  echo "Teenager"
fi
```

Looping

`for VAR in item1 item2 ...; do commands done`

Iterates over a list of items.

Example:

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

`while [ condition ]; do commands done`

Executes commands while a condition is true.

Example:

```
i=0
while [ $i -lt 5 ];
do
  echo $i
  i=$((i+1))
done
```

`until [ condition ]; do commands done`

Executes commands until a condition is true.

Example:

```
i=0
until [ $i -ge 5 ];
do
  echo $i
  i=$((i+1))
done
```

Case Statements

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

Example:

```
case $OS in
  "Linux") echo "Linux OS" ;;
  "Windows") echo "Windows OS" ;;
  *) echo "Other OS" ;;
esac
```

Functions & Commands

Function Definition

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

Example:

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

Passing arguments to functions:
`function_name arg1 arg2`
Access arguments inside the function using `$1`, `$2`, etc.

Essential Commands

`ls`	List directory contents.
`cd`	Change directory.
`mkdir`	Create directory.
`rm`	Remove files or directories.
`cp`	Copy files or directories.
`mv`	Move files or directories.
`cat`	Concatenate and display files.
`grep`	Search for patterns in files.
`find`	Search for files based on criteria.

String Manipulation

`substring=${string:position:length}` - Extracts a substring.
Example:

```
string="Hello World"
substring=${string:0:5}  # Hello
```

`length=${#string}` - Gets the length of a string.
Example:

```
string="Hello"
length=${#string}  # 5
```

`string//pattern/replacement` - Replaces all occurrences of a pattern.
Example:

```
string="Hello World"
new_string=${string//World/Universe}  # Hello Universe
```

Advanced Techniques

Error Handling

`set -e` - Exit immediately if a command exits with a non-zero status.

Example:

```
set -e
command_that_might_fail
echo "This will not be executed if the command fails"
```

`||` - Execute a command only if the previous command fails.

Example:

```
command_that_might_fail || echo "Command failed"
```

`&&` - Execute a command only if the previous command succeeds.

Example:

```
command_that_must_succeed && echo "Command succeeded"
```

Process Substitution

`<(command)` - Provides the output of a command as if it were a file.

Example:

```
diff <(ls dir1) <(ls dir2)
```

`>(command)` - Redirects output to a command.

Example:

```
ls > >(tee output.txt)
```

Debugging

`set -x` - Display commands and their arguments as they are executed.
`set +x` - Disable command tracing.

`echo "Debugging message" >&2` - Print debugging messages to stderr.