Cheat Sheet: Introduction to Shell Scripting

Bash shebang

1. #!/bin/bash

Get the path to a command

1. which bash

Pipes, filters, and chaining

Chain filter commands together using the pipe operator:

1. Is | sort -r

Pipe the output of manual page for statement to display the first 20 lines:

1. man ls | head -20

Use a pipeline to extract a column of names from a csv and drop duplicate names:

1. cut -d "," -f1 names.csv | sort | uniq

Working with shell and environment variables:

List all shell variables:

1. set

Define a shell variable called my_planet and assign value Earth to it:

1. my_planet=Earth

Display value of a shell variable:

1. echo \$my_planet

Reading user input into a shell variable at the command line:

1. read first_name

Tip: Whatever text string you enter after running this command gets stored as the value of the variable first_name.

List all environment variables:

1. env

Environment vars: define/extend variable scope to child processes:

- 1. export my_planet
- 2. export my_galaxy='Milky Way'

Metacharacters

Comments #:

1. # The shell will not respond to this message

Command separator ::

1. echo 'here are some files and folders'; Is

File name expansion wildcard:

1. Is *.json

Single character wildcard ?:

1. Is file_2021-06-??.json

Quoting

Single quotes " - interpret literally:

1. echo 'My home directory can be accessed by entering: echo \$HOME'

Double quotes "" - interpret literally, but evaluate metacharacters:

1. echo "My home directory is \$HOME"

Backslash \(\cdot\) - escape metacharacter interpretation:

1. echo "This dollar sign should render: \\$"

I/O Redirection

Redirect output to file and overwrite any existing content:

1. echo 'Write this text to file x' > x

Append output to file:

1. echo 'Add this line to file x' >> x

Redirect standard error to file:

1. bad_command_1 2> error.log

Append standard error to file:

1. bad_command_2 2>> error.log

Redirect file contents to standard input:

1. $tr "[a-z]" "[A-Z]" < a_text_file.txt$

The input redirection above is equivalent to:

1. | \$cat a_text_file.txt | tr "[a-z]" "[A-Z]"

Command Substitution

Capture output of a command and echo its value:

- 1. THE_PRESENT=\$(date)
- 2. echo "There is no time like \$THE_PRESENT"

Capture output of a command and echo its value:

1. echo "There is no time like \$(date)"

Command line arguments

1. ./My_Bash_Script.sh arg1 arg2 arg3

Batch vs. concurrent modes

Run commands sequentially:

1. start=\$(date); ./MyBigScript.sh; end=\$(date)

Run commands in parallel:

1. /ETL_chunk_one_on_these_nodes.sh & ./ETL_chunk_two_on_those_nodes.sh

Scheduling jobs with cron

Open crontab editor:

1. crontab -e

Job scheduling syntax:

1. m h dom mon dow command

(minute, hour, day of month, month, day of week)

Tip: You can use the * wildcard to mean "any".

Append the date/time to a file every Sunday at 6:15 pm:

1. 15 18 * * 0 date >> sundays.txt

Run a shell script on the first minute of the first day of each month:

1. 101**./My_Shell_Script.sh

Back up your home directory every Monday at 3:00 am:

1. 0 3 * * 1 tar -cvf my_backup_path\my_archive.tar.gz \$HOME\

Deploy your cron job:

Close the crontab editor and save the file.

List all cron jobs:

1. crontab -l

Conditionals

ifthenelse syntax:

1. if [[\$# == 2]]

2. then

a. echo "number of arguments is equal to 2"

3. else

a. echo "number of arguments is not equal to 2"

4. fi

'and' operator &&:

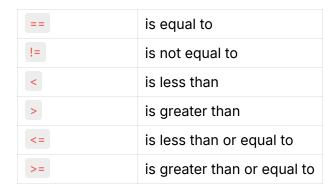
1. if [condition1] && [condition2]

'or' operator ||:

1. if [condition1] || [condition2]

Logical operators

Operator Definition



Arithmetic calculations

Integer arithmetic notation:

1. \$(())

Basic arithmetic operators:

Symbol	Operation
+	addition
-	subtraction
*	multiplication
1	division

Display the result of adding 3 and 2:

1. echo \$((3+2))

Negate a number:

1. echo \$((-1*-2))

Arrays

Declare an array that contains items 1, 2, "three", "four", and 5:

1. my_array=(1 2 "three" "four" 5)

Add an item to your array:

```
1. my_array+="six"
```

Declare an array and load it with lines of text from a file:

```
1. my_array=($(echo $(cat column.txt)))
```

for loops

Use a for loop to iterate over values from 1 to 5:

```
1. for i in \{0..5\}; do
```

- a. echo "this is iteration number \$i"
- 2. done

Use a for loop to print all items in an array:

```
1. for item in ${my_array[@]}; do
```

- a. echo \$item
- 2. done

Use array indexing within a **for** loop, assuming the array has seven elements:

```
1. for i in {0..6}; do
```

- a. echo \${my_array[\$i]}
- 2. done

Summary & Highlights

Congratulations! You have completed this module. At this point, you know that:

 A shell script is a program that begins with a 'shebang' directive and is used to run commands and programs. Scripting languages are interpreted rather than compiled.

^{2.} my_array+=7

- Filters are shell commands. The pipe operator `|` allows you to chain filter commands.
- Shell variables can be assigned values with `=` and listed using `set.`
 Environment variables are shell variables with extended scope, and you can list them with `env.`
- Metacharacters are special characters that have meaning to the shell.
- Quoting specifies whether the shell should interpret special characters as metacharacters or 'escape' them.
- Input/Output, or I/O redirection, refers to a set of features used for redirecting.
- You can use command substitution to replace a command with its output.
- Command line arguments provide a way to pass arguments to a shell script.
- In concurrent mode, multiple commands can run simultaneously.
- You can schedule cron jobs to run periodically at selected times. `m h dom mon dow command` is the cron job syntax.
- You can edit cron jobs by running `crontab -e,` and `crontab -l` lists all cron jobs in the cron table.