Module 3 Cheat Sheet - Introduction to Shell Scripting

Bash shebang

```
#!/bin/bash
```

Copied!

Get the path to a command

```
which bash
```

Copied!

Pipes, filters, and chaining

Chain filter commands together using the pipe operator:

```
ls | sort -r
```

Copied!

Pipe the output of manual page for `ls` to `head` to display the first 20 lines:

```
man ls | head -20
```

Copied!

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

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

Copied!

Working with shell and environment variables:

List all shell variables:

```
set
```

Copied!

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

```
my_planet=Earth
```

Copied!

Display value of a shell variable:

```
echo $my_planet
```

Copied!

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

```
read first_name
```

Copied!

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

List all environment variables:

```
env
```

Copied!

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

```
export my_planet
export my_galaxy='Milky Way'
```

Copied!

Metacharacters

Comments #:

```
# The shell will not respond to this message
```

Copied!

Command separator ;:

```
echo 'here are some files and folders'; ls
```

Copied!

File name expansion wildcard *:

```
ls *.json
```

Copied!

Single character wildcard ?:

```
ls file_2021-06-??.json
```

Copied!

Quoting

Single quotes '' - interpret literally:

```
echo 'My home directory can be accessed by entering: echo $HOME'
```

Copied!

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

```
echo "My home directory is $HOME"
```

Copied!

Backslash \ - escape metacharacter interpretation:

```
echo "This dollar sign should render: \$"
```

Copied!

I/O Redirection

Redirect output to file and overwrite any existing content:

```
echo 'Write this text to file x' > x
```

Copied!

Append output to file:

```
echo 'Add this line to file x' >> x
```

Copied!

Redirect standard error to file:

```
bad_command_1 2> error.log
```

Copied!

Append standard error to file:

```
bad_command_2 2>> error.log
```

Copied!

Redirect file contents to standard input:

```
$ tr “[a-z]” “[A-Z]” < a_text_file.txt
```

Copied!

The input redirection above is equivalent to:

```
$cat a_text_file.txt | tr “[a-z]” “[A-Z]”
```

Copied!

Command Substitution

Capture output of a command and echo its value:

```
THE_PRESENT=$(date)
echo "There is no time like $THE_PRESENT"
```

Copied!

Capture output of a command and echo its value:

```
echo "There is no time like $(date)"
```

Copied!

Command line arguments

```
./My_Bash_Script.sh arg1 arg2 arg3
```

Copied!

Batch vs. concurrent modes

Run commands sequentially:

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

Copied!

Run commands in parallel:

```
./ETL_chunk_one_on_these_nodes.sh  & ./ETL_chunk_two_on_those_nodes.sh
```

Copied!

Scheduling jobs with cron

Open crontab editor:

```
crontab -e
```

Copied!

Job scheduling syntax:

```
m  h  dom  mon  dow   command
```

Copied!

(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:

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

Copied!

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

```
1  0 1 * * ./My_Shell_Script.sh
```

Copied!

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

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

Copied!

Deploy your cron job:

Close the crontab editor and save the file.

List all cron jobs:

```
crontab -l
```

Copied!

Conditionals

`if-then-else` syntax:

```
if [[ $# == 2 ]]
then
  echo "number of arguments is equal to 2"
else
  echo "number of arguments is not equal to 2"
fi
```

Copied!

'and' operator `&&`:

```
if [ condition1 ] && [ condition2 ]
```

Copied!

'or' operator `||`:

```
if [ condition1 ] || [ condition2 ]
```

Copied!

Logical operators

Operator	Definition
`==`	is equal to
`!=`	is not equal to
`<`	is less than
`>`	is greater than
`<=`	is less than or equal to
`>=`	is greater than or equal to

Arithmetic calculations

Integer arithmetic notation:

```
$(())
```

Copied!

Basic arithmetic operators:

Symbol	Operation
`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division

Display the result of adding 3 and 2:

```
echo $((3+2))
```

Copied!

Negate a number:

```
echo $((-1*-2))
```

Copied!

Arrays

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

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

Copied!

Add an item to your array:

```
my_array+="six"
my_array+=7
```

Copied!

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

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

Copied!

`for` loops

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

```
for i in {0..5}; do
    echo "this is iteration number $i"
done
```

Copied!

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

```
for item in ${my_array[@]}; do
  echo $item
done
```

Copied!

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

```
for i in {0..6}; do
    echo ${my_array[$i]}
done
```

Copied!

Authors

Jeff Grossman
Sam Propupchuk

Other Contributors

Rav Ahuja