



Module 3 Cheat Sheet - Introduction to Shell Scripting

Bash shebang

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

Get the path to a command

```
which bash
```

Pipes, filters, and chaining

Chain filter commands together using the pipe operator:

```
ls | sort -r
```

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

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

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

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

Working with shell and environment variables:

List all shell variables:

```
set
```

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

```
my_planet=Earth
```

Display value of a shell variable:

```
echo $my_planet
```

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

```
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:

```
env
```

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

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

Metacharacters

Comments `#`:

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

Command separator `;`:

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

File name expansion wildcard `*` :

```
ls *.json
```

Single character wildcard `?` :

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

Quoting

Single quotes `' '` - interpret literally:

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

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

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

Backslash `\` - escape metacharacter interpretation:

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

I/O Redirection

Redirect output to file and overwrite any existing content:

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

Append output to file:

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

Redirect standard error to file:

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

Append standard error to file:

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

Redirect file contents to standard input:

```
$ tr -d 'a-z' -d 'A-Z' < a_text_file.txt
```

The input redirection above is equivalent to:

```
$cat a_text_file.txt | tr -d 'a-z' -d 'A-Z'
```

Command Substitution

Capture output of a command and echo its value:

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

Capture output of a command and echo its value:

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

Command line arguments

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

Batch vs. concurrent modes

Run commands sequentially:

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

Run commands in parallel:

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

Scheduling jobs with cron

Open crontab editor:

```
crontab -e
```

Job scheduling syntax:

```
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:

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

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

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

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

```
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:

```
crontab -l
```

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
```

'and' operator `&&` :

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

'or' operator `||` :

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

Logical operators

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

Arithmetic calculations

Integer arithmetic notation:

```
$(())
```

Basic arithmetic operators:

Symbol	Operation
<code>+</code>	addition
<code>-</code>	subtraction
<code>*</code>	multiplication
<code>/</code>	division

Display the result of adding 3 and 2:

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

Negate a number:

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

Arrays

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

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

Add an item to your array:

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

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

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

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
```

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

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

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

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

Authors

Jeff Grossman
Sam Propupchuk

Other Contributors

Rav Ahuja

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-06-07	2.0	Jeff Grossman	Added advanced scripting examples
2023-05-17	1.3	Nick Yi	Added content
2023-05-09	1.2	Nick Yi	Add code blocks, update title
2023-04-26	1.1	Nick Yi	ID Review
2023-02-14	1.0	Jeff Grossman	Update to reflect module content