BASH cheat sheet - Level 2

Miscellaneous

\ Escape character. It preserves the literal value of the next character that follows, with the exception of newline.

\` *command*\` The backtick (\`) is a command substitution.

echo The current working directory is: \`pwd\`
>The current working directory is: */home/user/path*

The text between a pair of backtick is executed by the shell before the main command and is then replaced by the output of that execution. The syntax **$(*command*)** is generally preferable.

$ It introduces parameter expansion, command substitution, or arithmetic expansion. The parameter name or symbol to be expanded may be enclosed in braces.

Using variables

variable=value
Assign a value *value* to the variable *variable*. The variable scope is restricted to the shell.

local *variable=value*
Assign a value *value* to the local variable *variable*. It doesn't come out a curly bracket area.

export *variable=value*
Make the variable *name* available to the shell and sub-processes.

***variable*=$(*command*)**
Assign the output of *command* to *variable*.

${#*variable*}
Length of the value contained by the variable.

${*variable*:*N*}
Keep the character of the value contained by variable after the **N**th.

${*variable*:*N*:*length*}
Substring the value contained by *variable* from thr Nth character to up to *length* specidied.

${*variable*/*pattern*/*string*}
The longest match of *pattern* against the *variable* value is replaced with *string*.

Print commands

echo *My home is: $HOME*
>My home is: /home/*user*
Write arguments to the standard output.

echo –e Enable interpretation of backslash-escaped characters.

printf Format and print the arguments.

printf %q "$IFS"
>' \t\n'
Print the arguments shell-quoted.

printf "%.1f" 2.558
>2.6
Specify the decimal precision.

```
printf "%s\t%s\n" "1" "2" "3" "4"
>1          2
 3          4
```
%s interprets the associated argument literally as string.

Using quotes

<u>**Weak quoting -** double quote (") :</u>

***string*="My home is: $HOME"**
echo $*string*
>My home is: /home/*user*
Use when you want to enclose variables or use shell expansion inside a string.

<u>**Strong quoting -** single quote (') :</u>

echo 'My HOME is: $HOME'
>My HOME is: $HOME
Preserves the literal value of each character within the quotes.

Wildcards operators

<u>Regular expressions :</u> Used to match text.

^	Matches the beginning of the line.
$	Matches the end of the line.
^$	Matches blank lines.
.	Any character.
[]	Any of the character inside the brackets.
[^a-f]	Matches any character except those in the range a to f.
\a	A letter (similar to [a-zA-Z]).
\t	A tabulation.
\n	A new line.
\w	An alphanumeric ([a-zA-Z0-9_])
\W	Non alphanumeric (The opposite of \w)
?	The preceding item matches 0 or 1 time.
*****	The preceding item matches 0 or more times.
+	The preceding item matches 1 or more times.
{N}	The preceding item matches exactly N times.
{N,}	The preceding item matches N times or more.
{N,M}	The preceding item matches at least N times and not more than M times.
[:*class*:]	POSIX Character Classes ([:alnum:], [:alpha:], [:blank:], [:digit:], etc, respectively equivalent to A-Za-z0-9, A-Za-z, space or a tab, 0-9, etc).

<u>Globbing (Pathname expansion) :</u>
Used to match filename(s).

?	Any single character
*****	Zero or more characters
[]	Specify a range. Any character of the range or none of them by using **!** inside the bracket.
{*term1*,*term2*}	Specify a list of terms separated by commas and each term must be a name or a wildcard.
{term1..term2}	Called brace expansion, this syntax expands all the terms between *term1* and *term2* (Letters or Integers).

<u>With the **extglob** shell option enabled (check it with **shopt**) :</u>
In the following description, a *pattern-list* is a list of one or more patterns separated by a |.

man *command* : display the *command*'s manual page

Jacques Dainat - 2015

?(*pattern-list*)	Matches zero or one occurrence of the given patterns.
***(*pattern-list*)**	Matches zero or more occurrences of the given patterns.
+(*pattern-list*)	Matches one or more occurrences of the given patterns.
@(*pattern-list*)	Matches one of the given patterns.
!(*pattern-list*)	Matches anything except one of the given patterns.

/!\ Regular expressions and globbing wildcards should not be mixed up. They have different meaning.

File modification commands

tr string1 string2 < file
Replace *string1* characters occurrences within *file* by *string2* characters (where the first character in string1 is translated into the first character in string2 and so on).

sed is a non-interactive text file editor :

sed 's/*pattern1*/*pattern2*/g' *ficOrigine*
Replace **pattern1** occurrence within *file* by ***pattern2***. The **s** means « substitute » and the **g** means « global replacment » (Not only the first occurence).
-e : allows combining multiple commands (use a -e before each command).
-i : Edit files in-place. (Be carefull using that option)

sed -n *5,10p file*
Print lines 5 to 10.

The awk command

awk is a field-oriented pattern processing language.

```
awk    'BEGIN { Initial command(s) }
              { by line command(s) }
        END { final command(s) }'    file
```

$0 is an entire line.
$1 is the first field, **$2** the second, etc.

By default, fields are separated by white space. Use the **–F** option to define the input field separator (can be a regular expression).

NF	Number of fields in the current record.
NR	Ordinal number of the current record.
FNR	Ordinal number of the current record in the current file.

-v *name*=$*var* It allows to pass the shell variable $*var* to awk command. The variable is known as *name* within the awk command.

awk '{ if (*$2* ~ *pattern*) *arr*[*$0*]++} END { for (*i* in *arr*){print *$i*} }' *file*
For each line where the second field match the *pattern*, save the line as key in the associative array *arr* and increment its value. At the end print each key of the associative array. This will remove the duplicate lines that have matched.

awk 'FNR==NR{*arr*[*$4*]++;next}{ if(*$4* in *arr*)print $0 }' *file1 file2*
Print all lines of *file2* where the fourth field matches one of the third field of *file1*.

String commands together

command < file
Redirect *file* into a *command*. *File* is read as standard input instead of the terminal command.

command1* | *command2
Connect the standard output of the left command to the standard input of the right command.

command1* ; *command2
Separate two commands. Permit putting several commands on the same line.

Math calculation

+	Plus
+=	Plus-equal (increment variable by a constant)
-	Minus.
-=	Minus-equal (decrement variable by a constant).
*****	Multiplication.
***=**	Times-equal (multiply variable by a constant).
/	Division.
/=	Slash-equal (divide variable by a constant).
%	Modulo (returns the remainder of an integer division operation).
%=	Modulo-equal (remainder of dividing variable by a constant).
******	Exponentiation.
++	Increment a variable by 1.
--	Decrement a variable by 1.

((*var* = *operation*)) or ***var*=$((*operation*))**
Assign the result of an arithmetic evaluation to the variable *var*.

/ !\ Natively Bash can only handle integer arithmetic.

Floating-point arithmetic
You must delegate such kind of calcul to specific command line tool as **bc**.

echo "*operation*" | bc –l
Display the result of a floating-point arithmetic.

***var*=$(echo "*operation* " | bc -l)**
Assign the floating-point arithmetic result to the variable *var*.

man *command* : display the *command*'s manual page

Jacques Dainat - 2015