Cheatography

Bash scripting Cheat Sheet

by eugvol via cheatography.com/198523/cs/42009/

Globs

*	Any number of characters inc. none	*.txt	test1.txt
?	Matches single character	?.txt	a.txt, not ab.txt
[abc]	Matches any one of enclosed characters	[ab].txt	a.txt, b.txt, not c.txt
[a-c]	Matches any character in the range	[a-c].txt	a.txt, b.txt, c.txt, not d.txt
?(pattern-list)	Extended: matches zero or one occurrence of the given patterns	?(a|b).txt	a.txt, b.txt, .txt
*(pattern-list)	Extended: matches zero or more occurrences of the given patterns	*(a|b|c).txt	a.txt, aa.txt, abac.txt, .txt
+(pattern-list)	Extended: matches one or more occurrences of the given patterns	+(a|b|c).txt	a.txt, ab.txt, ba.txt, aaabbbccc.txt, etc
@(pattern-list)	Extended: matches one of the given patterns	@(a|b|c).txt	a.txt, b.txt, or c.txt
!(pattern-list)	Extended: matches anything except one of the given patterns	!(a|b|c).txt	any .txt files except a.txt, b.txt, c.txt

shopt -s extglob turns on extended globs.
shopt -s nullglob return no output if no matches, otherwise returns the glob pattern.
shopt -s nocaseglob makes globing case-insensitive.
shopt -s dotglob includes filenames starting with a dot (hidden files) in glob patterns.

For loop

Basic

```
for i in 1 2 3 4 5
do
     echo " Number $i"
done
```

C-style syntax

```
for ((i = 1; i <= 5; i++))
do
     echo " Number $i"
done
```

Over command output

```
for user in $(cat /etc/p asswd | cut -d ':'
do
     echo " User: $user"
done
```

Over files

```
for file in /path/to/directory/*
do
     echo " Pro cessing $file"
done
```

Over range with break

```
for i in {1..10}
do
     if [ " $i" -eq 5 ]; the
n
          break
     fi
     echo " Number $i"
done
```

Over range with step

```
for i in {0..10..2}
do
     echo " Number $i"
done
```

break and **continue** can be used

Special characters and variables

/	Root directory	$HOME	The current user's home directory
.	Current directory	$PWD	The current working directory
~	Current user directory	$PATH	A list of directories separated by colons (:) where the system looks for executable files

Cheatography

Bash scripting Cheat Sheet

by eugvol via cheatography.com/198523/cs/42009/

Special characters and variables (cont)

~username	Directory of a user with username	$USER	The username of the user running the script
..	Parent directory	$HOSTNAME	The hostname of the machine the script is running on
$0	The name of the Bash script	$RANDOM	Returns a different random number each time is it referred to
$1-$9	The first 9 arguments to the Bash script	$SECONDS	Number of seconds since the shell was started. Can be used to measure elapsed time of a script
$#	Number of arguments passed to Bash script	$OLDPWD	Previous directory
$@	All the arguments supplied to the Bash script	$LINENO	Current line number in a script or shell. Used for debugging
$?	The exit status of the most recently run process	$SHELL	Path to the user's default shell
$$	The process ID of the current script	$UID	User unique ID

Arithmetics

let writes the result to a variable but doesn't print it. Used *only for integers*.

```
let b=2+5
let b=2*5
let c=$a/$b
let c=$a%$b
let c=$a**$b
let "a = 5 + 2"
let "a=5+2"
let "c = $a * $b"
let "c = (2 + 3) * 4"
let a++
let b--
let c+=1
```

expr returns the result and prints it. *Spaces* are important. Used *only for integers*.

```
expr 10 + 5
expr 3 * 2
expr 10 / 2
expr $a - $b
expr 11 % 5
a=$(expr 11 % 5)
```

$(()) can also be used for arithmetics *with integers only*.

```
a=$((3+4))
b=$(($ a-$b))
c=$((a*b))
((b=++a))
((b=--a))
c=$((b +=3))
```

bc is used for more complex calculations. It can deal *with decimals as well*.

```
a=$(echo "5 + 3" | bc)
b=$(echo " 10.5 -2.3" | bc -l)
c=$(echo " sqr t(2 5)" | bc -l)
d=$(echo "2 ^ 3" | bc -l)
rounde d_v alu e=$ (echo " sca le=2; 10/3" | bc)
```

Shebang

```
#!/bin/bash
#!/usr/bin/env bash
```

Variables

`MY_STR 1=Hello`	Hello
`MY_STR 2=$ MY_STR1`	Hello
`MY_NUM=45`	45
`MY_STR 3="Num value is $MY_NUM "`	Num value is 45
`MY_STR 4='Num value is $MY_NUM '`	Num value is $MY_NUM
`MY_PAT H=/etc`	/etc
`MY_COM MAN D=$(ls $MY_PATH)`	result of **ls /etc**

Integers test

Within [ ]	Within [[ ]]
-eq	==
-ne	!=
-gt	>
-lt	<
-ge	>=
-le	<=

Decimals comparison

File tests

-d	returns 0 if directory
-f	returns 0 if file
-e	returns 0 if exists
-s	returns 0 if file isn't empty
-r	returns 0 if file exists and permissions are granted (-w, -x are also possible)

Use either **test -flag argument** or **[-flag argument]** format.
Use **echo $?** to get the result in bash cmd

String tests

`[ $a = " Hel lo" ]`	returns 0 if strings are equal
`[ $a != $b ]`	returns 0 if strings are not equal
`[ -z $a ]`	returns 0 if $a length is zero
`[ -n $a ]`	returns 0 if $a length is non-zero
`[[ $a == $b ]]`	returns 0 if strings are equal
`[[ $a != $b ]]`	returns 0 if strings are not equal
`[[ $a > $b ]]`	returns 0 if $a is alphabeti-cally greater than $b
`[[ $a < $b ]]`	returns 0 if $a is alphabeti-cally less than $b
`[[ " $FI LEN AME " == *.txt ]]`	returns 0 if $FILENAME matches glob pattern
`[[ " Hello !!" =~ ^Hello [[: spa ce:]]. * ]]`	returns 0 if string matches regex pattern

White spaces are important.

```
decimal1=3.14
decima l2=2.71
# Use bc to compare decimal numbers
result =$(echo " $de cimal1 > $decim al2 " | bc -
l)
if [ " $re sul t" -eq 1 ]; then
       echo " $de cimal1 is greater than $decim -
al2 "
elif [ " $re sul t" -eq 0 ]; then
       echo " $de cimal1 is equal to $decim al2 "
else
       echo " $de cimal1 is less than $decim -
al2 "
fi
```

Conditional structures

cmd1 || cmd2	run cmd1, if fails run cmd2
cmd1 && cmd2	run cmd1, if ok run cmd2

```
if [ $a -gt 5 ] && [ $b -eq 20 ]; then
     echo "a > 5 AND b=20"
elif [ $b -lt 15 ] || [ $c -ge 30 ]; then
     echo " b<15 OR c>=30"
elif ! [ $a -eq 10 ]; then
     echo "a!=10"
else
     echo "None of the conditions met"
fi
```

```
case $input in
     start|START)
         echo " Sta rting the process..."
         ;;
     stop|STOP)
         echo " Sto pping the process..."
         ;;
     *)
         echo " Invalid option: $input"
         ;;
 esac
```

```
option s=( " STA RT" "STOP")
select opt in "$options[@]"
do
    case $opt in
     START)
         echo " Sta rting the process..."
         ;;
     STOP)
         echo " Sto pping the process..."
         ;;
     *)
         echo " Invalid option: $input"
         ;;
 esac
done
```

Associative arrays	
Declare an associative array	`declare -A fruits`
Adding/appending an item	`fruits[apple]="red"` `fruits[banana]="yellow"`
Reading an item	`echo "The apple is ${frui ts[ app le ] }"`

Cheatography

Bash scripting Cheat Sheet

by eugvol via cheatography.com/198523/cs/42009/

Associative arrays (cont)

Changing an item value	`fruits [ap ple ]="g ree n"`
Removing an item	`unset fruits [ba nana]`

Looping through keys	`for key in " ${! fru its [@] }"; do`
	`  echo "$key"`
	`done`

Looping through values	`for value in " ${f rui ts[ @]} "; do`
	`  echo "$value"`
	`done`

Looping through keys and values	`for key in " ${! fru its [@] }"; do`
	`  echo " $key: ${fruits[$key]} "`
	`done`

Length of an associative array	`echo ${#fru its[@]}`

While/Until loops

While	Until

`counter=1`
`while [ $counter -le 10 ]`
`do`
`  echo $counter`
`  ((counter++))`
`done`
	`counter=1`
	`until [ $counter -gt 10 ]`
	`do`
	`  echo $counter`
	`  ((counter++))`
	`done`

While with reading from a file

```
while IFS= read -r line; do
    echo "$line"
done < filena me.txt
```

Exporting env variables

printenv	Print list of all env variables
export VAR="Hello World"	Create env variable (for current session only)
echo 'export NEW_VAR="Hello World"' >> ~/.bashrc	Create env variable for future sessions
source ~/.bashrc	Reload the shell startup file (required after changing the file with prev. command)
export PATH=$PATH:newvalue	Appending a value (for current session only)
unset VAR	Remove env variable

Arrays

Explicit declaration	`declare -a my_array`
Implicit creation	`my_arr ay= (el ement1 element2 element3)`
Read a single element	`echo ${my_a rra y[0]}`
Read all elements	`echo ${my_a rra y[@]}`
Changing an element	`my_arr ay[ 1]= new _el ement`
Appending an element	`my_arr ay+ =(e lem ent4)`
Removing an element	`unset my_arr ay[1]`
Length of an array	`echo ${#my_ arr ay[@]}`

Looping through an array	`for element in " ${m y_a rra y[@ ]}"; do`
	`  echo $element`
	`done`

Sparce array	`declare -a sparse_array`
	`sparse_array[3]="Third Element"`
	`sparse_array[7]="Seventh Element"`
	`for index in " ${! spa rse _ar ray [@] }";`
	`  echo " Index $index: ${sparse_array[$i`
	`"`
	`done`

ECHO and READ

`echo " Hello, World! "`	Hello, World!
`echo -n "This is a "` `echo " single line."`	This is a single line.
`echo -e "This is a\ttab \ts epa rat ed \tt e xt."`	This is a tab separated text.
`echo -E "This is a\ttab \ts epa rat ed \tt e xt."`	This is a\ttab\ts-eparated-\ttext.
`read var`	reading into a variable
`read var1 var2 var3`	reading into several variables
`read -a fruits`	reading into array (white-space is the default delimiter)
`read -p " Enter your name: " name`	reading with prompt
`read -sp "Password: " password`	reading with silent input and prompt
`read -n 3 val`	reading with a limited number of characters
`read -t 5 val`	reading with 5s timeout