

basic bash commands

- ☐ pwd : print working directory
- ☐ cd /path/to/dir : change directory
- ☐ ls /dir/to/list : list directory content (default is .)
 - l : display the content on one column
 - l : display the content with long listing format
 - a : display the content of the directory (including hidden files)
 - R : Display the content of the directory and the content of subdirectories
- ☐ mv /path/to/file /path/where/to/move : move or rename a file or a directory
- ☐ cp /path/to/file /path/where/to/copy : copy a file
 - r : copy recursively (used to copy directory)
- ☐ rm /path/to/file : remove a file
 - r : remove recursively (used to remove directories)
 - f : force remove
- ☐ mkdir /path/dirName : create an empty directory
- ☐ rmdir /path/to/dir : remove a directory (works only if the directory is empty)

bash redirections

- ☐ command > file : redirect stdout to file. (creates the file if it doesn't exist and overwrite it if it does exist)
- ☐ command >> file : redirect stdout to file. (creates the file if it doesn't exist and append to the end if it does exist)
- ☐ command 2> file : redirect stderr to file (creates the file if it doesn't exist and overwrite it if it does exist)
- ☐ command 2>> file : redirect stdout to file. (creates the file if it doesn't exist and append to the end if it does exist)

bash redirections (cont)

- ☐ command &> file : redirect stdout and stderr to file (creates the file if it doesn't exist and overwrite it if it does exist)
- ☐ command &>> file : redirect stdout and stderr to file. (creates the file if it doesn't exist and append to the end if it does exist)
- ☐ command < file : redirect stdin to file.
- ☐ command1 | command 2 : uses the output of command1 as the input of command2

file globbing regex

- ☐ \ : escape character. It deletes the significance of a special character
- ☐ ? : Any character, once.
- ☐ * : Any character, 0, 1 or many time.
- ☐ [...] : Any character that is in the class. ex: [abc], [a-z], [0-9]
- ☐ [^...]: Any character that is not in the class. ex: [^abc], [^a-z], [^0-9]
- ☐ {s1, s2, sN} : match s1 or s2 or sN

control structure (if)

```
if <expression>; then
    [st ate ments]
elif <expression>; then
    [st ate ments]
else
    [st ate ments]
fi
```

control structure (while)

```
while <expression>; do
    [st ate ments]
done
```

control structure (for)

```
for var in <expression>; do
    echo $var
    [st ate ments]
done
```

control structure (case)

```
# patterns are file globbing
regex
case <expression> in
    pattern1)
        [st ate ments]
        ;;
    pattern2)
        [st ate ments]
        ;;
    *)
        [st ate ments]
        ;;
esac
```

function definition

```
function functionName {
    [st ate ments]
    [return X]
}
```

conditional expressions

- ☐ && : logical and operator
- ☐ || : logical or operator
- ☐ [[string]] : return 0 if string is not empty
- ☐ [[-z string]] : return 0 if the string is empty
- ☐ [[string1 == string2]] : return 0 if the string are equivalent
- ☐ [[string1 != string2]] : return 0 if the string are not equivalent
- ☐ [[string =~ pattern]] : return 0 if the string matches the pattern (extended regex)
- ☐ [[-e file]] : return 0 if the file exists



By gregcheater

Published 14th March, 2016.
Last updated 11th May, 2016.
Page 1 of 3.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

conditional expressions (cont)

- ☐ `[[ -d file ]]` : return 0 if file is a directory
- ☐ `[[ -f file ]]` : return 0 if file is a file
- ☐ `[[ -x file ]]` : return 0 if file is executable
- ☐ `[[ $n1 -eq $n2 ]]` : return 0 if \$n1 == \$n2
- ☐ `[[ $n1 -lt $n2 ]]` : return 0 if \$n1 < \$n2
- ☐ `[[ $n1 -gt $n2 ]]` : return 0 if \$n1 > \$n2
- ☐ `[[ $n1 -ge $n2 ]]` : return 0 if \$n1 >= \$n2
- ☐ `[[ $n1 -le $n2 ]]` : return 0 if \$n1 <= \$n2
- ☐ `[[ $n1 -ne $n2 ]]` : return 0 if \$n1 != \$n2

more basic bash commands

- ☐ `passwd` : change your password
- ☐ `history` : consult the history of your command
- ☐ `jobs` : list of your pending processes
- ☐ `cat file1 file2 ...` : concatenate files and print to stdout
- ☐ `more / less file1 file2 ..` : display a file page by page on stdout
- ☐ `tail / head number` : display the "number" first or last line of a file on stdout
- ☐ `touch file1 file2 ...` : change the modification date of the files
- ☐ `chmod` : change the privileges of a file / directory
- ☐ `echo "text"` : display a line of text to stdout
- ☐ `sort file1 file2 ...` : sort the file (combine files if many are specified) and print the result to stdout (files aren't impacted)
 - r : sort in reverse order
 - n : numerical sort
 - u : delete duplicated lines
- ☐ `wc file1 file2 ...` : print to stdout the number of characters, words and lines of files
 - l : number of lines only
 - w : number of words only
 - c : number of characters only

more basic bash commands (cont)

- ☐ `diff file1 file2` : compare file1 and file 2 for differences
 - i : ignore the character case
 - B : ignore empty lines
 - w : ignore whitespaces
 - c : add context to the output (good for readability)
- ☐ `which commandName` : print the path of a command
- ☐ `pushd / popd /path/to/dir` : change directory using the directory stack
- ☐ `dirs` : print the directory stack
- ☐ `find /path/to/dir -name pattern` : find every files and directory that have a name that matches "pattern" in the directory specified and its subdirectories
- ☐ `man commandName` : Display the manual for command commandName
- ☐ `sudo command` : run the command as superuser
- ☐ `command1 | xargs -i command2` : uses the output of the command1 as the input of the command2. output will be accessible via {} in command2

grep (simple regex)

- ☐ `grep "pattern" file1 file2 ...` : print the lines that matched the pattern
 - v : print lines that didn't match the pattern
 - i : ignore the character case
 - l : print the name of the files that have at least one match
 - o : print only the piece of line that matched the pattern
 - E : uses the extended regex
 - q : quiet. returns 0 in \$? if at least one line has been matched. 1 if no line matched

variables

- ☐ `VAR=VARVALUE` : create a variable VAR. the variable can be accessed like so: \$VAR or \${VAR}
- ☐ `VAR="$VAR2"` : \$VAR will contains the value of \$VAR2
- ☐ `VAR='VAR2'` : \$VAR will contains \$VAR2
- ☐ `VAR=$(command)` : \$VAR will contains the output of the command
- ☐ `(( VAR = $VAR + 1 ))` : the double parentheses must be used when doing arithmetics
- ☐ `${VAR#pattern}` : return a substring of VAR where the smallest string (starting from the beginning) matching "pattern" will be cut
- ☐ `${VAR##pattern}` : return a substring of VAR where the longest string (starting from the beginning) matching "pattern" will be cut
- ☐ `${VAR%pattern}` : return a substring of VAR where the smallest string (starting from the end) matching "pattern" will be cut
- ☐ `${VAR%%pattern}` : return a substring of VAR where the longest string (starting from the end) matching "pattern" will be cut
- ☐ `?` : the exit status of the last command / function executed. usually 0 when everything went right.
- ☐ `$#` : the number of args passed to the script / function
- ☐ `$0` : the name of the script
- ☐ `$n` : the nth argument passed to the script / function
- ☐ `$@` : the list of all the argument passed to the script / function
- ☐ Arrays

`myArray=(value1 value2 value3)`: declare an array

`declare -a myArray=(value1 value2 value3)`: declare an array



By gregcheater

cheatography.com/gregcheater/

Published 14th March, 2016.
Last updated 11th May, 2016.
Page 2 of 3.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

variables (cont)

`${myArray[index]}` : access an element (index starts at 0)

`myArray[index]=` : add or modify the element at index

`${#myArray[*]}` : return the length of the array

`${myArray[*]}` : all the elements of the array

simple regex

`\` : escape character. It deletes the significance of a special character

`.` : joker. It represents any characters

`*` : 0, 1 or many repetition of the last character / sequence of character

`^` : The beginning of the line

`$` : The end of the line

`[...]` : Any character that is in the class. ex: [abc], [a-z], [0-9]

`[^...]` : Any character that is not in the class. ex: [^abc], [^a-z], [^0-9]

`\(...\)` : Capture the pattern. The pattern can then be accessed with `\1`, `\2` ... `\n` depending on the number of capture in the regex

`\{n\}` : n repetitions of the last character / sequence of character

`\{n,\}` : At least n repetitions of the last character / sequence of character

`\{n, m\}` : Between n and m repetitions of the last character / sequence of character

extended regex

`\` : escape character. It deletes the significance of a special character

`.` : joker. It represents any characters

`*` : 0, 1 or many repetition of the last character / sequence of character

`+` : 1 or more repetition of the last character / sequence of character

extended regex (cont)

`?` : The last character / sequence of character can appear or not

`^` : The beginning of the line

`$` : The end of the line

`[...]` : Any character that is in the class. ex: [abc], [a-z], [0-9]

`[^...]` : Any character that is not in the class. ex: [^abc], [^a-z], [^0-9]

`s1|s2` : Either s1 or s2 but not both

`(...)` : change the priority

`{n}` : n repetitions of the last character / sequence of character

`{n,}` : At least n repetitions of the last character / sequence of character

`{n, m}` : Between n and m repetitions of the last character / sequence of character

sed (simple regex)

`sed 'sed script' file` : execute the script on every line of "file"

`s/pattern/newString/gI` : Substitute the piece of the line that matches "pattern" by "newString". g (optional): global, I (optional): ignore case

`/pattern/d` : delete the line if "pattern" is matched

`/pattern/p` : print the line if "pattern" is matched

`/pattern1/,/pattern2/` : print every lines between the first line that matches "pattern1" to the first line that matches "pattern2"

`-i.ext` : Modifications done "in-place". A backup file will be created with .ext extension (it is optional)

`-n` : print only the lines that matched the pattern

awk (extended regex)

`awk -Fc 'awk script' file1 file2 ...` (where "c" is the delimiter)

typical awk script: 'BEGIN {statements} /pattern/ {script statements} END {statements}'

BEGIN {} : Will be executed once at the start

END {} : Will be executed once at the end

/pattern/ : only lines that matched the pattern will be processed

/pattern1/,/pattern2/ : every line from the first line that matches pattern1 to the first line that matches pattern2 will be processed

{script statements} : core of the script

printf: C-style formatter (man printf)

\$n : the nth field of the line

\$0 : the entire line

NR : the record number

NF : the number of fields in the record

FS: The field separator (the delimiter)