Unix Guide

Meher Krishna Patel

Created on : Octorber, 2017

Last updated : October, 2018

Table of contents

Chapter 1

Unix commands

Various useful unix-commands are listed in this chapter. Also, some of the examples of these commands are shown in the last section. Further, in next chapter, shell-scripting is discussed.

1.1 Unix commands cheatsheet

Following are the list of various useful unix-commands,

1.1.1 Basic file commands

Commands	Details
Basic opeations	
ls	directory listing
ls -a	show hidden file
ls -l	show details for file
ln file1 file2	create file2 (linked with file1). Changes will appear in both file
ln -s file1 file2	create shortcut of file1 as file2 (soft link)
mkdir dirname	create directory dirname
cd dirname	go to dirname
cd	go to home directory
pwd	present working directory
rm filename	remove (delete) filename
rm -r dirname	remove directory dirname
rm -f filename	Force remove file filename
rm -rf dirname	Force remove directory dirname
cp file1 file2	copy file1 to file2
cp -r dir1 dir2	copy dir1 to dir2
mv file1 file2	rename file1 to file2
touch filename	create filename (if not exist)
cat > filename	add content to file (ctrl-c to exit)
cat >> filename	add content at the end of the filename
cat filename	show content of filename (all at once)
more filename	show content of filename (fit to screen with more option)
head filename	show first 10 lines of filename
tail filename	show last 10 lines of filename
tail -f filename	last 10 lines of the filename (useful for log files)
command > filename	write output of command in the filename (overwrite the fie
command >> filname	append the output at the end of file name
wc filename (wc -l, -w, -c)	returns number of lines, word, character and filename.
chmod 777 filename	read(4), write(2) and execute(1) permission to all [4+2+1=7] chmod [owner, group, others] filename

1.1.2 Sort

Commands	Details
Sort	
sort filename	sort the lines of the filename
sort -u filename	sort and eliminate duplicates
sort -r filename	reverse order sort
sort file1 -o file2 sort file1 > file2	sort data of file1 and save to file2
sort -n filename	sort data based on numeric value i.e. 12 > 4
sort -M fielname	sort data with month name i.e. Jan, Feb, Mar etc.

1.1.3 Cut

Commands	Details
Cut	
cut d"-" 1,4 filename	cut the 1st and 4th column of filename, where - is delimiter
cut -c3- filename	remove first 2 character from each line and printed the rest
cut -c-3-8 filename	print character 3-8 from each line
cut -c1-3,6-8,10- filename	print character 1-3, skip 4, print 6-8, skip 9, print 10 to end from each line
cut -d',' -f3 filename	extact field-3 from each line with ',' as delimiter
cut -d',' -f3,6 filename	extact field-3-to-6 from each line with ',' as delimiter

1.1.4 Paste

Commands	Details
Paste	
paste file1 file2 file3	paste the lines of file2, then file3 beside the lines of file1
paste -d',' file1 file2	put comma at the end of each file content
paste -s filename	merge all lines of filename

1.1.5 Grep

Commands	Details
grep	
grep pattern filenames	print the lines with pattern in filenames
grep '[0-9]' filenames	print the lines from filenames which have numbers 0-9 (regular expression)
grep -v pattern filenames	print all lines which do not contain pattern
grep -l pattern filenames	print the filenames which have pattern
grep -n pattern filenames	print the line numbers as well

1.1.6 Sed

Commands	Details
sed	
sed 's/word1/word2/' filename	replace word1 with word2 in filename
sed 's/word1/word2/' file1 > file2 mv file2 > file1	use two steps to make change permanent
sed -n '2,4p' filename	print line 2 to 4 of filename
sed -n /word1/p' filename	print lines which contain word1 in filename

1.2 Multiple commands

1.2.1 Pipes

Pipes are used for sending output of one command to next command.

```
# total number of files in the directory
$ ls | wc -l
1
```

(continues on next page)

(continued from previous page)

```
# word counts for file testfile.txt
$ cat testfile.txt | wc
      8      37     188
```

1.2.2 Run individual commands (;)

Commands can be run separately from one line by separating them using semicolon,

```
$ date; ls
Thu Feb 23 15:24:09 NZDT 2017
testfile.txt
```

1.3 Examples

1.3.1 File examples

```
$ ls
$ touch testfile.txt # create testfile.txt
$ ls
testfile.txt

$ cat testfile.txt

$ cal > testfile.txt  # write the calender to testfile.txt
$ cat testfile.txt
   February 2017
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28

$ date >> testfile.txt  # append date at the end of testfile.txt
$ cat testfile.txt
   February 2017
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28

Thu Feb 23 14:47:51 NZDT 2017

$ date > testfile.txt  # replace all the content of testfile with date
$ cat testfile.txt
Thu Feb 23 14:48:09 NZDT 2017

$ cal > testfile.txt  # write calender to file
$ wc testfile.txt     # count lines, word and character in testfile.txt
   8  37 188 testfile.txt

# other options for wc
$ wc -l testfile.txt  # lines
```

(continues on next page)

(continued from previous page)

```
8 testfile.txt
$ wc -w testfile.txt  # words
37 testfile.txt
$ wc -c testfile.txt  # characters
188 testfile.txt
```

1.3.2 Sort example

```
# create a file
$ cat > test
Tiger
Lion
Cat
Eagle
^C

# see the content
$ cat test
Tiger
Lion
Cat
Eagle

# sort the content
$ sort test
Cat
Eagle
Lion
Tiger
dhriti@meher ~/Desktop $
```

1.3.3 Cut command

```
$ cat > test
first last age id
Meher Patel 20 101
Krishna patel 30 102
Tom Jerry 43 103
^C

$ cut -d" " -f 1,3,4 test
first age id
Meher 20 101
Krishna 30 102
Tom 43 103
```

Chapter 2

Shell scripting

In this section, Unix-shell-scripting is discussed.

2.1 Hello World

Files containing the shell scripts are usually saved as filename.sh.

```
# helloworld.sh

echo "Hello World"
echo Hello World Again
```

The 'bash' or 'sh' command can be used to execute the above file. Further, we can change the permission of the file using 'chmod' command and execute it directly as shown below,

```
$ bash helloworld.sh
Hello World
Hello World Again

$ sh helloworld.sh
Hello World
Hello World Again

$ chmod 777 helloworld.sh
$ ./helloworld.sh
Hello World
Hello World Again
```

2.2 Execute commands from script

We can store various commands in a file and then run all those commands by executing the file. This is quite useful for repetitive tasks,

```
# run_commands.sh

echo 'list of files'
ls

echo 'date'
date
```

```
$ sh run_commands.sh
list of files
helloworld.sh  run_commands.sh      testfile.txt
date
Mon Feb 27 07:10:33 NZDT 2017
```

2.3 Shell Variables and input from user

2.3.1 Shell variable

In below code, 'n' is the variable; and $ sign is used to display the value of the variable,

```
# test.sh

# no spaces between variable and input
n='Tiger'

echo $n
```

Output is shown below,

```
$ echo test.sh
Tiger
```

2.3.2 Arithmetic

For mathematical operation, **$((expression))** is used as shown in following example,

```
# test.sh

i=1

j=$((i+1))

echo $j
```

Output of above code is shown below,

```
$ bash test.sh
2
```

2.3.3 Single quote vs double quote

Note the differences between the outputs with single and double quote,

```
# test.sh

i=1

j=$((i+1))

echo '$j'
echo "$j"
```

Below are the outputs,

```
$ bash test.sh
$j
2
```

2.3.4 User input

'read' is used to get input from user,

```
# run_commands.sh

echo enter your name
read name

echo Hello $name, How are you?
```

```
$ bash run_commands.sh
enter your name
Meher
Hello Meher, How are you?
```

2.4 Positional arguments

We can read some arguments while executing the code as well. $1 and $2 are the positional arguments provided by the user,

```
# test.sh

# $1 and $2 are the arguments provided while executing the file
echo $1 $2
```

```
$ bash test.sh 12
12
$ bash test.sh 12 13
12 13
$ bash test.sh 12 13 14
12 13
```

- 'set' command can used to convert an string to positional parameter as shown below,

```
$ set Meher Krishna Patel
$ echo $1
Meher
$ echo $2
Krishna
$ echo $3
Patel
$ echo $4
```

- (' ') can be used to save the outputs of the command as positional parameter,

```
$ cat > test
How are you?
^C
$ cat test
How are you?
$ set `cat test`
```

(continues on next page)

(continued from previous page)

```
$ echo $1
How
$ echo $2
are
```

2.5 Mathematical operations

'expr', $, $((expression)) or ' | bc ' commands can be used for mathematical operation as shown below.

Note: There is no spaces between x=3.

```
# test.sh

x=4
y=2


echo $x + $y
echo `expr $x + $y`    # ` ` is required with expr for maths operations

i=$(expr $x + 1)
echo $i

z=`echo $x \* $y | bc`    # * has special meaning, hence \*
echo $z

d=$(($x / $y))  # using double bracket $((expression))
echo $d
```

2.6 if-else

Note that [ ] are used for condition-check operations (not required for other cases). Also, look for the spaces after [and before].

```
# test.sh

echo enter a number
read x

if [ $x -gt 4 ]
then
    echo 'greater than 4'
elif [ $x -eq 4 ]
then
    echo 'equal to 4'
else
    echo 'less then 4'
fi
```

Following is the output of above code,

```
$ sh test.sh
enter a number
```

(continues on next page)

(continued from previous page)

```
6
greater than 4
```

- Below code check whether the file exists in the directory or not. Similarly '-d', '-r' and '-w' can be used for checking the directory, read and write permission respectively

```
# test.sh

echo enter a filename
read x

if [ -f $x ]
then
    echo 'yes, there is file'
else
    echo 'no, there is no such file'
fi
```

Following is the output,

```
$ sh test.sh
enter a filename
test.sh
yes, there is file
```

2.7 Case statement

Use double quote with case statement,

```
# test.sh

# case '$1'   # do not use single quote
case "$1"
in
    0) echo zero;;
    1) echo one;;
    *) echo 'neither 0 nor 1';;
esac
```

Following are the outputs,

```
$ bash test.sh 1
one
$ bash test.sh 0
zero
$ bash test.sh 3
neither 0 nor 1
```

2.8 For loop

- Below code loop over all the files of the directory and print the result,

```
# test.sh

for files in *
do
```

(continues on next page)

(continued from previous page)

```
    echo $files
done
```

Following is the output of above code, which listed all the files in the current directory,

```
$ bash test.sh
helloworld.sh
meher.txt
month.txt
m.txt
run_commands.sh
surname.txt
temp.txt
test.sh
```

- Following is the another example,

```
# test.sh

for files in 1 2 3
do
    echo $files
done

# outputs
# $ bash test.sh
# 1
# 2
# 3
```

2.9 While loop

Following are the example of while loop,

```
# test.sh

i=1

while [ "$i" -lt 4 ]
do
    echo $i
    i=$(($i+1))
done


# outputs
# $ bash test.sh
# 1
# 2
# 3
```