UNIX COMMANDS

Compiled By

Mohammed Abdul Sami

sami.ma@gmail.com

ORACLE Certified Associate
Oracle Solaris 11
System Administrator

ITIL

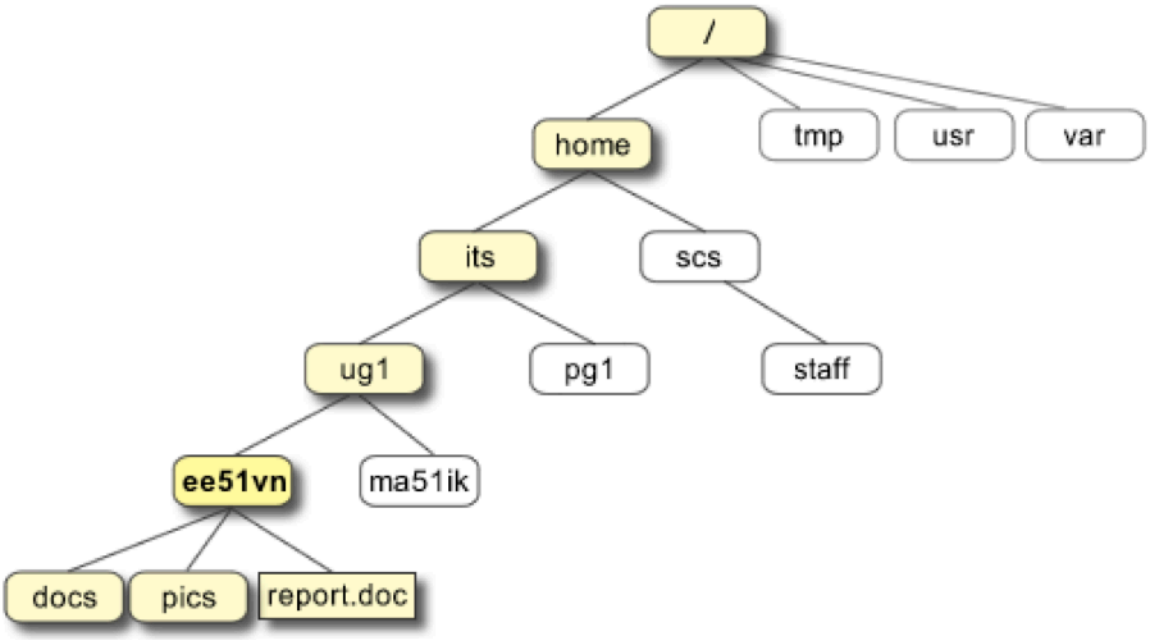
redhat. CERTIFIED ENGINEER

bmcsoftware CERTIFIED ADMINISTRATOR
BMC CONTROL-M FOR DISTRIBUTED SYSTEMS

Basic Unix Commands

Directory Commands:

All the files are grouped together in the directory structure. The file-system is arranged in a hierarchical structure, like an inverted tree. The top of the hierarchy is traditionally called root (written as a slash **/**)



man hier command gives more information about the directories created by system

pwd
Command to displays the current directory.
cd
You can change your current directory with the cd command
cd ~
The cd is also a shortcut to get back into your home directory.
cd ..
To go to the parent directory (the one just above your current directory)
cd -
Another useful shortcut with cd is to just type cd - to go to the previous directory.
ls
You can list the contents of a directory with **ls** command
ls -l
Long list
ls -lh
Long list with file sizes mentioned in human readable form
ls -li
Long list with file inode number
ls -lu
Long list and show let access time instead of last modification time
ls -lr
long list with reverse sorting

ls -lS
long list and sort by file size

mkdir <dir name>
command to create a new directory
mkdir -p <dirname>/<dirname>
creates directory along with required parent directories

rmdir <dir name>
Deletes given directory
rmdir -p <dirname>/dirname>
deletes directory and its parent directory as well

Wildcard Characters in filename (file globbing):

***** can be used to represent 0 to many characters in the filenames
? can be used represent one single character in the filename
[] square brackets
The square bracket [is interpreted by the shell as a sign to generate filenames, matching any of the characters between **[]**. Each pair of brackets is replaced by exactly one character.
Ex: rm ran[iy].txt
the above command delete the any file whose name is either **rani.txt** or **rany.txt**
Ranges can be used within **[]** like **[0-9]** for any of the digit, **[a-z]** for any of the lower case letters

File Commands:

touch <filename>
Cretes a new empty file
touch -m <filename>
Changes the last modified time of the file to current time
touch -a <filename>
changes the last access time of the file the current time
touch -m -t yyyymmddhhmi.ss <file name>
changes the last modified time of the file to given time (yyyymmddhhmi.ss)
touch -a -t yyyymmddhhmi.ss <filename>
changes the last access time of the file to given time (yyyymmddhhmi.ss)

The **cat** command can be used to print the content of the given file on the terminal

cat <filename>
cat command can also be used to create a new file
cat <file1> <file2> <file3>
prints the content of all files as if one single file
tac <filename>
prints contents of **<filename>** from last line to first line
cat > <filename>
above will allow you to type text and save the text by pressing **CTRL+D**
strings command
prints the readable characters from binary file

The **rm** Command

Deletes the given file or files for ever
rm -i <file name>
Deletes the given file but before ask for user confirmation
rm -f <file name>
Deletes the given file force fully and does not ask for confirmation
rm -rf <dir name>
Deletes the entire directory recursively including all files and sub directories

The **cp** - copy command can be used to create a copy of the in the same directory or create a copy of the file in another directory.

cp <filename1> <filename2>
a copy of file **<filename1>** will be created in the same directory as **<filename2>**
cp <filename> </directory/>

<filename> will be copied to the given directory
cp -r
copy recursively, copies directory recursively
cp -p
copies files and retains the ownership, permissions etc

mv - move command can be used to move the file or directories to another location
mv <filename1> </path/directory>
moves the **<filename1>** to given directory **</path/directory/>**
mv command renames the file if it is moved within same directory
mv <filename1> <filename2>
<filename1> will be renamed as **<filename2>**
file command tells what type of file is it
file <filename>

stat command

stat command gives more information about the file
head <filename>
prints first 10 lines of the files
head -n <filename>
prints first **n** lines from the file
tail <filename>
prints last 10 lines of the file
tail -n <filename>
print last **n** lines of the file

more <filename>
prints large file one full screen at a time then scrolls down as user presses ENTER key
less <filename>
Same as **more** command but allows user to scroll up and scroll down using arrow keys

Combining Commands:

; semicolon
You can put two or more commands on the same line separated by a semicolon ; . Both the commands will be executed sequentially with the shell waiting for each command to finish before starting the next one.

Ex: **$ ls ; pwd**

&& double ampersand
The shell will interpret **&&** as a logical AND. When using **&&** the second command is executed only if the first one succeeds

ex: **$ ls && pwd**
ex: **$ lls && pwd**

|| double vertical bar
The || represents a logical OR. The second command is executed only when the first command fails

ex: **$ ls || pwd**
ex: **$ lls || pwd**

Output Redirection

Greater than symbol **(>)** can be used to diver the output of any command from default output destination (eg: monitor) to any other device (like File or Printer)
If the **>** is used to redirect the output to a file then the target file will be overwritten. Instead if single **>** we can used **>>** to append tp the target file.

The bash shell has three basic streams; it takes input from **stdin** (stream 0), it sends output to **stdout** (stream 1) and it sends error messages to **stderr** (stream 2) .

a not **>** is actually **1>** mean redirect only expected output from commands. (Not Error Messages).

Error messages from the commands can be redirected using **2>** (Eg. **lls 2> err.txt**)

2>&1 redirects error messages to same target as output

&> redirect both output and error messages to same file (Eg. **find / abc.txt &> my output.txt**)

Unix Pipes

Pipe (|) can be used to send the output of one command to another command
Ex: **ls | tail -2**
I above example the output of the command **ls** will be sent as input to **tail** command

Unix Filters Commands

tee command is used to store and view (both at the same time) the output of any other command. **tee** command writes to the **STDOUT** and also to given file.

ls | tee <filename>
The output of the command ls will be printed on terminal (**STDOUT**) and also written to the file **<filename>**

The **grep** filter is to filter lines of text containing (or not containing) a certain string.

grep -i which filters in a case insensitive way.
grep -v which outputs lines not matching the string.
grep -A1 one line after the matching line is also displayed.
grep -B1 one line before the matching line also displayed
grep -C1 one line before and after the matching line also displayed

The **cut** filter can select columns from files, depending on a delimiter. it assumes default delimiter as **tab**.

Ex: **$ ls | cut -c1-5**
Prints first 5 letters of each filename
Ex: **$ ls |cut -c1,3,5**
prints the first, third and fifth letter of each filename

If the file is having tab separated columnar data then **cut** command can be used to print selected columns (fields) using **-f** option

Ex: **cat /etc/protocols |cut -f1**

To avoid lines which are not arranged in columns use **-s** options with cut command

Ex: **cat /etc/protocols|cut -s -f1**

Printing selected columns by specifying column numbers separated by comma **(,)**

ex: **cat /etc/protocols|cut -s -f1,2**

Printing a range of columns by using hyper **(-)**

ex: **cat /etc/protocols | cut -s -f1-2**

If the columns are separated by any other character like comma (,) use **-d** option to specify column separator (delimiter)

ex: **cat /etc/passwd | cut -d':' -f1**

/etc/passwd contains linux user information such as user information in column separated by a **:**

tr can translate characters

ex: **ls | tr 'a' 'A'**

any character **'a'** in the filenames will be replaced with **'A'** in the output

ex: **ls | tr [0-9] 'N'**

any number in the filenames will be replaced with 'N' in the output

ex: **cat <filename>|tr [a-z] [A-Z]**

content of files will be printed in upper case letters

Consecutive occurrence of given character will be squeezed to single occurrent

ex: **cat <filename>|tr -s ' '**

it will squeeze more than one space to just one.

tr -d to delete characters from output

ex: **ls |tr -d 'a'**

wc Counting words, lines and characters

wc <filename>

prints number of lines, words and characters in the file **<filename>**
wc -l counts number of lines
wc -w counts number of words
wc -c counts number of characters

ex: **ls|wc -l**

sort filter will sort the input in alphabetical order

ex: **ls | sort**

-r portion can be used to sort in reverse order

ex: **ls| sort -r**

in case if the input data arranged in columns then sort has option **-kn** (n is for columns number) to sort using any of the given column number
ex: **cat /etc/protocols|cut -s -f1,2 |sort -k1**
it sorts on the second column
-t option can be used along with **-k** to specify column separator
ex: **cat /etc/passwd|sort -k1 -t:**
-n option can be used if numerical data is sorted. for example, third column of **/etc/passwd** file represent user ID which is a number

ex: **cat /etc/passwd|sort -k3 -t: -n**
ex: **cat /etc/protocols|cut -s -f1,2 |sort -k2 -n**

uniq filter can remove duplicates from a sorted list

```
$ cat music.txt
Queen
Brel
Queen
Abba
$ sort music.txt
Abba
Brel
Queen
Queen
```

```
$ sort music.txt |uniq
Abba
Brel
Queen
```

uniq can also count occurrences with the **-c** option

```
$ sort music.txt |uniq -c
1 Abba
1 Brel
2 Queen
```

comm

Comparing 2 files can be done with the **comm**. By default **comm** will output three columns. First column displays the text which is present only in the first file, second column displays the text which is present only in second file and third column displays the text which is present in both the files.

Consider 2 files **list1.txt** and **list2.txt**

```
cat list1.txt
Abba
Bowie
Cure
Queen
Sweet
cat list2.txt
Abba
Cure
Queen
Turner
comm list1.txt list2.txt
                Abba
Bowie
                 Cure
                 Queen
Sweet
        Turner
```

comm command allows you to decide which column not to display by specifying that column number

comm -12 list1.txt list2.txt
third column will be displayed
comm -13 list1.txt list2.txt
second column will be displayed
comm -23 list1.txt list2.txt
first column will be displayed

find command can be used just to find the files or find the files and perform some action of each of the file found. by default it list out either files or directories but can be tweaked to find only files or directories.

Options:

-name "file / directory" file / directory to be searched
-iname "fine / directory" file / directory to be searched ignoring case
-type f or **d** for file or directory
-mtime N modified **N** days ago
-mtime +N modified earlier than **N** days
-mtime -N modified later than **N** days
-atime N accessed **N** days ago
-atime +N accessed earlier than **N** days
-atime -N accessed later than **N** days
-newer <filename> find all files which are modified after the given file **<filename>**
-size N Matching file size
Actions:
-exec unix command {} \; executed the given command for each file found. Flower braces **{}** are replaced with current filename
-delete deletes the file
-print prints the file name

Syntax:
find </path/to/search> [OPTIONS] [ACTIONS]
ex: Search for all .txt files on the server
find / -iname "*.txt"

ex2: Search for all .txt files

ex3: Search for all files modified a week ago and delete them from **/backup** directory
find /backup -mtime 7 -delete
or
find /backup -mtime 7 -exec rm -f {} \;

locate command searches for the given file. Tt relies on an index created for locate command. We can create or update that index by using **updatdb** command.

```
$ updatedb
$ locate abc.txt
```

The date command can display the date, time, time zone and more.

```
$ date
$ date +' format'
```

Formats: **%Y, %d, %m , %T** (time), **%H, %M %S** and **%p** (AM/PM)

Set Date:

```
date +format -s "date string"
```

The **hwclock -r** command shows **BIOS** time

The **cal** command prints the calendar

```
$ cal
```

Archiving

The **tar** command archives a complete directory into a single file called as tar archive or tar ball

```
$ tar  -cvf  my archive.tar   /directory/to/be/tarred
```

Options:

c tells **tar** command to create a new archive from **/directory/to/be/tarred**
v tells **tar** command to be verbose and print progress messages on terminal
f tells **tar** command to create given **.tar** file

A **tar** ball can be extracted using same **tar** command

```
$ tar -xvf my archive.tar
```
Options:

x tells **tar** command to extract the archive
v tells **tar** command to be verbose and print progress messages on terminal
f tells **tar** command to create given **.tar** file

The **gzip** command compresses the given file and appends **.gz** at the end of compressed filename

```
$ gzip <filename>
```

The **gunzip** command uncompresses the given gzipped **(.gz)** file and removes the **.gz** from filename

```
$ gunzip  <filename>.gz
```

The **zcat** command prints the contents of gzipped file which normal **cat** command cannot do

```
$ zcat  <filename>.gz
```

The **zmore** command prints the contents of gzipped files one screen at a time

```
$ zmore  <filename>.gz
```

The **bzip2** command compresses the given file and appends **.bz2** at the end of compressed filename

```
$ bzip2 <filename>
```

The **bunzip2** command uncompresses the given bzipped **(.bz2)** file and removes the **.bz2** from filename

```
$ bunzip2  <filename>.bz2
```

The **bzcat** command prints the contents of bzipped file which a normal **cat** command cannot do

```
$ bzcat  <filename>.bz2
```

The **bzmore** command prints the contents of bzipped files one screen at a time

```
$ bzmore  <filename>.bz2
```

Unix Exercise for Directory, File and Filters

Working with Directories:

1. Display your current directory.

2. Change to the /etc directory.

3. Now change to your home directory using only three key presses.

4. Change to the /boot/grub directory using only eleven key presses.

5. Go to the parent directory of the current directory.

6. Go to the root (/) directory.

7. List the contents of the root directory.

8. List a long listing of the root directory.

9. Stay where you are, and list the contents of /etc.

10. Stay where you are, and list the contents of /bin and /sbin.

11. Stay where you are, and list the contents of ~.

12. List all the files (including hidden files) in your home directory.

13. List the files in /boot in a human readable format.

14. Create a directory testdir in your home directory.

15. Change to the /etc directory, stay here and create a directory newdir in your home directory.

16. Create in one command the directories ~/dir1/dir2/dir3 (dir3 is a subdirectory from dir2, and dir2 is a subdirectory from dir1).

17. Remove the directory testdir.

Working With Files

1. Display the type of file of linux_commands.pdf present in ~/Desktop directory
2. Display the type of file /bin/passwd
3. list the file in /etc directory with their last access date
4. Create a file called today.txt using cat command
5. print the contents of file resolv.conf present in /etc directory
6. Create a directory ~/mydir and enter it.
7. Create the files myfile1.txt and myfile2.txt in mydir.
8. Change the last update date on myfile1.txt to match yesterday's date.
9. Copy myfile2.txt to copy.myfile2.txt
10. Rename copy.myfile2.txt to kim.txt
11. Create a directory called ~/testbackup and copy all files from ~/mydir into it.
12. Use one command to remove the directory ~/testbackup and all files into it.
13. Create a directory ~/etcbackup and copy all *.conf files from /etc into it.
14. Display the first 12 lines of /etc/services.
15. Display the last line of /etc/passwd.
16. Use cat to create a file named count.txt that looks like this:

    One
    Two
    Three
    Four
    Five
17. Use cp to make a backup of count.txt file to cnt.txt.
18. Display cnt.txt, but with all lines in reverse order (the last line first).
19. Use more to display /etc/services.
20. Display the readable character strings from the /usr/bin/passwd file.
21. Use ls to find the biggest file in /etc.
22. Execute ls after cd /etc, but only if cd /etc succeeds.
23. Execute cd /etc after cd etc, but only if cd /etc fails.

24. Create a test directory and enter it.

25. Create the following files :

    file1
    file10
    file11
    file2
    File2
    File3
    file33
    fileAB
    filea
    fileA
    fileAAA
    file(
    file 2

(the last one has 6 characters including a space)

26. List (with ls) all files starting with file

27. List (with ls) all files starting with File

28. List (with ls) all files starting with file and ending in a number.

29. List (with ls) all files starting with file and ending with a letter

30. List (with ls) all files starting with File and having a digit as fifth character.

31. List (with ls) all files starting with File and having a digit as fifth character and nothing else.

32. List (with ls) all files starting with a letter and ending in a number.

33. List (with ls) all files that have exactly five characters.

34. List (with ls) all files that start with f or F and end with 3 or A.

35. List (with ls) all files that start with f have i or R as second character and end in a number.

36. List all files that do not start with the letter F.

37. Write a find command that finds all files created after January 30th 2018.

38. Write a find command that finds all *.odf files created on 1st August 2018.

39. Count the number of *.conf files in /etc and all its subdirs.

40. Create a file called loctest.txt. Can you find this file with locate ? Why not ? How do you make locate find this file ?

41. Use find and -exec to move all .html files to ~/mywww directory

42. Issue the date command. Now display the date in YYYY/MM/DD format.

43. Issue the cal command. Display a calendar of 1582 and 1752. Notice anything special ?

44. create a tar ball of /root directory

45. gzip the tar ball created in question 44

Unix Filters

Pattern Matching Using Regular Expressions (Regex)

Regex is a technique used to extract information from text files by searching the files with specific search patterns. Regex can be used with many unix tools such as grep, sed, awk, vi etc

Regular Expression Metacharacters

Metacharacter	Function
\	Escapes the special meaning of an RE character
^	Matches the beginning of the line
$	Matches the end of the line
\<	Matches the beginning of the word anchor
\>	Matches the end of the word anchor
[ ]	Matches any one character from the specified set
[-]	Matches any one character in the specified range
*	Matches zero or more of the preceding character
.	Matches any single character
\{\}	Specifies the minimum and maximum number of matches for a regular expression

Escaping a Regular Expression

- A \ (backslash) character escapes the regular expression characters.
- The \ interprets the next character literally, not as a metacharacter.
- Thus, a **\$** matches a dollar sign and a **\.** matches a period.

```
$ grep '$' /etc/enscript.cfg
```

The above command prints all lines

now run the same thing by escaping the $

```
$ grep '\$' /etc/enscript.cfg
```

Now it prints only lines which are having $ symbol

Line Anchors

Line anchors force a regular expression to match only at the start or end of a line.

- `^` for the beginning of the line.
- `$` for the end of the line.

```
$ cat /etc/group|grep -i 'me'
```

This above line prints all line which have 'me' in it either as part of other word or as a separate word.

Same command used with line anchor

```
$ cat /etc/group|grep -i '^me'
```

Now it prints only those line which starts with 'me'.

Consider the following command which prints all lines from **`/etc/passwd`** file which has 'sh' in them.

```
$  cat /etc/passwd|grep -i 'sh'
```

Now use end of line anchor $ to print only those lines which ends with 'sh'

```
$  cat /etc/passwd|grep -i 'sh$'
```

Exercise 1

1. Print only lines from from /etc/passwd file which ends with ‘bash’
2. Print only lines from /etc/passwd file which ends with ‘csh’
3. Print only lines from /etc/group file which starts with ‘r’
4. Print only lines from /etc/yum.conf which contains ‘.’
5. Print blank lines from /etc/yum.conf file
6. Print commented lines from /etc/protocols file

Word Anchors

Word anchors are used to refer to the beginning and end of a word in regular expressions.

- Use `\<` for the beginning of the word.
- Use `\>` for the end of the word.

Like line anchors we can use word anchors to match the search patter at the beginning of the word or end of the word.

```
$ cat /etc/protocols | grep -i 'udp'
```

prints all lines which contains string 'udp' whether it is a complete word or part of another word

Below code prints the lines where 'udp' is a complete word

```
$ cat /etc/protocols | grep -i  '\<udp\>'
```

Below code prints the lines containing the words which starts with 'udp'

```
$ cat /etc/protocols | grep -i  '\<udp'
```

Below code prints the lines containing the words which ends with 'udp'

```
$ cat /etc/protocols | grep -i 'udp\>'
```

Character Classes

A character class makes one small sequence of characters match a larger set of characters, such as the following:

- `[abc]` matches a single character in the class.
- `[a-c]` matches a single character in the range.
- `[^a-c]` matches a single character not in the range.

```
$ cat /etc/protocols | grep  '^[udp]'
```

The above code prints all line which starts with letters 'u', 'd' or 'p'

```
 $ cat /etc/protocols | grep  '^[a-c]'
```

The above code prints all lines which starts with either 'a', 'b' or 'c'

```
$ cat /etc/protocols | grep  '^[^udp]'
```

The above code prints all lines which does not start with letters 'u', 'd' or 'p'

Named Classes

certain named classes of characters are predefined within bracket expressions, as follows.

`'[:alnum:]'` Alphanumeric characters: '`[:alpha:]`' and '`[:digit:]`'; in the 'C' locale and ASCII character encoding, this is the same as '[0-9A-Za-z]'.

`'[:alpha:]'` Alphabetic characters: '`[:lower:]`' and '`[:upper:]`'; in the 'C' locale and ASCII character encoding, this is the same as '[A-Za-z]'.

`'[:blank:]'` Blank characters: space and tab.

`'[:cntrl:]'` Control characters. In ASCII, these characters have octal codes 000 through 037, and 177 (DEL). In other character sets, these are the equivalent characters, if any.

`'[:digit:]'` Digits: 0 1 2 3 4 5 6 7 8 9.

`'[:graph:]'` Graphical characters: '`[:alnum:]`' and '`[:punct:]`'.

`'[:lower:]'` Lower-case letters; in the 'C' locale and ASCII character encoding, this is a b c d e f g h i j k l m n o p q r s t u v w x y z.

`'[:print:]'` Printable characters: '`[:alnum:]`', '`[:punct:]`', and space.

`'[:punct:]'` Punctuation characters; in the 'C' locale and ASCII character encoding, this is ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~.

`'[:space:]'` Space characters: in the 'C' locale, this is tab, newline, vertical tab, form feed, carriage return, and space. See Usage, for more discussion of matching newlines.

`'[:upper:]'` Upper-case letters: in the 'C' locale and ASCII character encoding, this is A B C D E F G H I J K L M N O P Q R S T U V W X Y Z.

`'[:xdigit:]'` Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f.

```
$ cat protocols |grep '\<[[:upper:]]\>'
```

The above code prints all lines which contains atleast one word in upper case letters

Single Character Match

The . (dot) regular expression matches any one character.

```
$ cat /usr/share/dict/words | grep '............'
```

The above code prints all words from dictionary which are 13 characters long

```
$ cat /usr/share/dict/words | grep '^c....'
```

The above code prints all words from dictionary which are minimum 5 characters long and starts with 'c'

Character Match by Specifying a Range

The `\{` and `\}` expressions allow you to specify the minimum and maximum number of matches for a regular expression.

```
$ cat /usr/share/dict/words | grep 't\{3\}'
```

The above code prints all words from dictionary which contains 'ttt' ('t' 3 times)

```
$ cat /usr/share/dict/words | grep 't\{1,3\}'
```

The above code prints all words from dictionary which contains 't', 'tt' to 'ttt'

Closure Character (*)

The * symbol, when used in a regular expression, is termed a closure. * matches the preceding character zero or more times.

```
$ cat /usr/share/dict/words | grep '\<t.*m\>'
```

The above code prints all words from dictionary which starts with 't' and ends with 'm'

Exercise 2

1. From /usr/share/dict/words print all words with 2 or 3 consecutive ‘o’
2. From /usr.share/dict/words prints all words 5 letters long starting with ‘ka’
3. From /usr/share/dict/words print all 4 letter words containing letters ‘l’ , ‘f’ , ‘e’ and ‘a’
4. From /etc/protocol file print all lines which contains only alphabets or numbers but not special characters.
5. From /usr/share/dict/words print all 3 letter words which does not contain any vowels

sed – Stream Editor

A “non-interactive” text editor that is called from the unix command line. Input text flows through the program, is modified, and is directed to standard output.

- A tool usually designed for a short line substitution, deletion and print
- Allows for automation of editing process similar to those you might find in vi or ex (or ed)
- Non-destructive
- Reads in a stream of data from a file, performs a set of actions, & outputs the results

How it Works

- SED reads the data one line at a time, make a copy of the input line & places it in a buffer called the pattern space
- Modifies that copy in the pattern space
- Outputs the copy to standard output (terminal window)
- The pattern space holds the line of text currently being processed
- You don’t make changes to the original file
- Changes can be captured by redirecting it to another files

sed Commands

a	append
c	change lines
d	delete lines
i	insert
p	print lines
s	substitute

syntax

```
$ sed [option] 'instruction'   <Input file>
```

- Options
  - n only prints matches
  - f <scriptfile> run commands in scriptfile
  - e allows multiple instructions on a single line
  - i in-line, source file will be modified

- Instructions

sed instruction consists of line addresses and sed command.

- Line Addresses
  a) Line Numbers
  b) Search Pattern

sed's Print Command (p)

USAGE - without any expression

```
$ sed -n 'p' <filename>
```

Prints the contents of file. Same like **cat** command

USAGE - With line numbers

```
$ sed -n '<lineno1>,<lineno2>p' <filename>
```

Examples:

```
$ sed -n '5p' <filename> ## prints fifth line
$ sed -n '3,6p' <filename > ## prints lines from 3 to 5
$ sed -n '$p' <filename> ## prints last line of the file
$ sed -n '5,$p' <filename> ## prints all lines from line 5th till end of file
$ sed -n '4,+2p' <filename> ## prints 4th line plus next 2 lines
$ sed -n '1~2p' <filename> prints every second line
```

USAGE - With search pattern

Always search pattern must be enclosed in / (forwarded slashes)

```
$ sed -n '/<pattern1>/, /<pattern2>/p' <filename>
```

Ex:

```
$ sed -n '/^w/p' <filename> ## prints all lines starting with 'w'
$ sed -n '/Y$/p' <filename> ## prints all lines ending with 'Y'
$ sed -n '/^$/p' <filename> ## prints all lines which are blank
$ sed -n '/abc/, /xyz/p' <filename> ## prints all lines between pattern 'abc' and 'xyz'
$sed -n '/^[arf]/p' <filename> ## prints all lines which starts with either a, r or f
$ sed -n /^[^arf]/p <finlename> ## prints all lines except which starts with a,r or f
```

sed's Delete Command (d)

USAGE - With line numbers

```
$ sed -n '<lineno1>,<lineno2>d' <filename>
```

USAGE - With search pattern

Always search pattern must be enclosed in / (forwarded slashes)

```
$ sed -n '/<pattern1>/, /<pattern2>/d' <filename>
```

Substitute Command (s)

Replaces search pattern with given pattern

USAGE

```
$ sed   's/pattern1/pattern2/flag'   <filename>
```

Flags:
n Replace nth instance of /pattern/ on each addressed line. n is any number in the range 1 to 512, and the default is 1.
g Replace all instances of /pattern/ on each addressed line, not just the first instance.

Regex can be used to build patterns

Append Command (a)

Append command is used to add a line after the matching address

USAGE with search Pattern

```
$ sed   '/<pattern>/ a\ <text for new line>'    <filename>
```

USAGE with search Pattern

```
$ sed '<line#> a\ <text for new line'    <filename>
```

Insert Command (i)

Insert command is used to add a line before the matching address

USAGE with search Pattern

```
$ sed   '/<pattern>/ i\ <text for new line>'    <filename>
```

USAGE with search Pattern

```
$ sed '<line#> i\ <text for new line'    <filename>
```

Change Command (c)

Change command deletes the matching lines and in their place writes a line with given text

USAGE with search Pattern

```
$ sed   '/<pattern>/ c\ <text for new line>'    <filename>
```

USAGE with search Pattern

```
$ sed '<line#> c\ <text for new line'    <filename>
```

Exercise for sed

1. Add 2 new line at the beginning of ~/.bashrc file have the following text
   JAVA_HOME="/usr/bin/java"
   export JAVA_HOME
2. Add DNS server 1.1.1.1 to /etc/resolve.conf file (need to run sed as root user)
3. From /etc/yum.conf file print 13 lines starting from section heading [main]
4. From .bash_history file delete all entries of clear command
5. Change the DNS server IP added in Question 2 to 8.8.8.8

Awk Programming

AWK is an interpreted programming language. It is very powerful and specially designed for the text processing. AWK name is derived from the family names of its authors – Alfred Aho, Peter Weinberger, and Brian Kernighan.

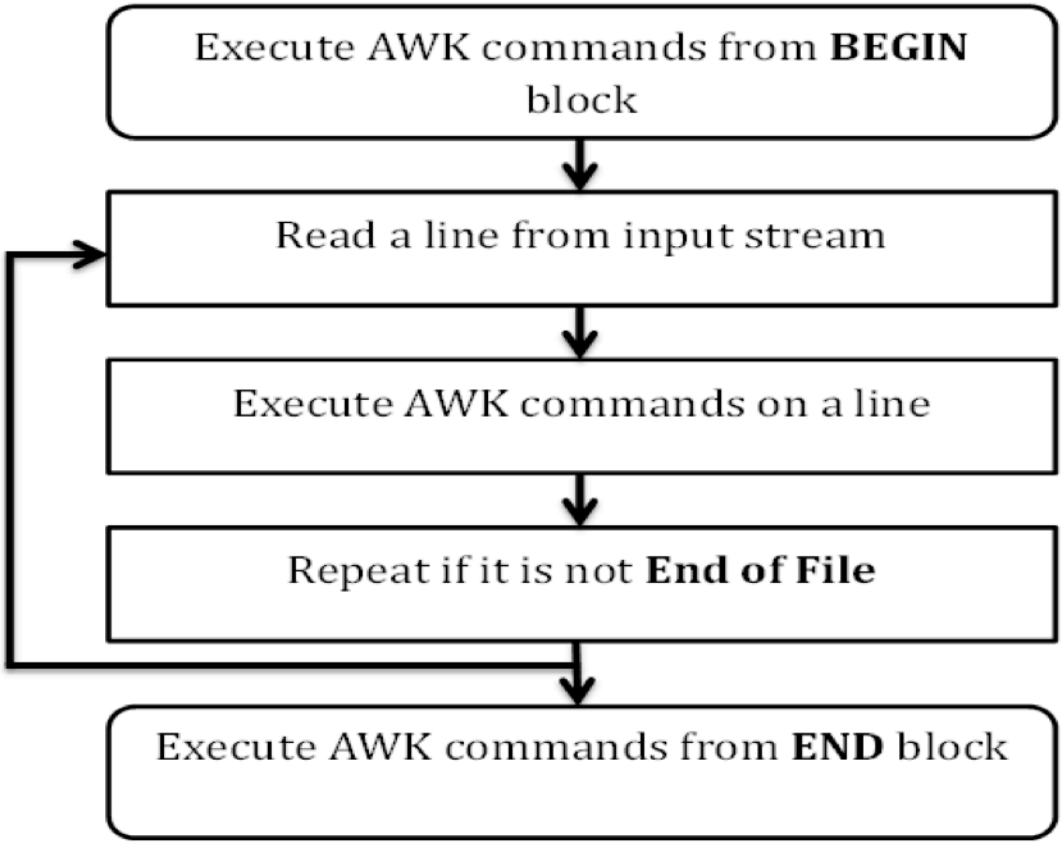
Versions

AWK - this is original AWK from AT & T Laboratory.
NAWK - this is newer and improved version of AWK from AT & T Laboratory.
GAWK - this is GNU AWK. All GNU/Linux distributions ship GAWK and is fully compatible with AWK and NAWK.

AWK assumes the file given is arranged in columns with each line having different fields.

How it Works

AWK follows a simple workflow: Read from input stream (file.. Pipe etc), Execute for each line from input, and Repeat till end of input stream



awk: Actions

awk is a pattern-action language. Action can be one or more from the following list separated by semicolons ';'

- **print** [list of expressions separated by ,]
- **printf** format [, list of expressions]
- **if** (conditional) statement [else statement]
- **Arithmetic Operations**

Syntax

Command line Usage:

```
$ awk   '/<pattern>/    {action}  '   <inputfile>
$ awk   '<condition>   {action} '    <inputfile>
$ awk   '{action} '   <inputfile>
$ awk   '/patterns/ {action1 ; action2}'  <inputfile>
$ awk   'BEGIN { commands}  {commands}  END {commands}'
```

Pattern matching can be done using Regex and condition can also be specified.

Variables

Variables can be used anytime no declaration is needed. 'C' like Assignment operators can be used to for assigning values to variables also increment and decrement operators can be used.

Eg:

```
X   =   10  -- spaces around '=' are for readability purpose
Y   +=  X
X++
X–
```

Assignment Operators

Operator	Meaning
=	Assignment
+=	Add result to variable
-=	Subtract result from variable
*=	Multiply variable by result
/=	Divide variable by result

%=	Apply modulo to variable

Arithmetic Operators

Operator	Type	Meaning
+	Arithmetic	Addition
-	Arithmetic	Subtraction
*	Arithmetic	Multiplication
/	Arithmetic	Division
%	Arithmetic	Modulo
++	Arithmetic	Increment
- -	Arithmetic	Decrement

Relational Operators

Operator	Meaning
==	Is equal
!=	Is not equal to
>	Is greater than
>=	Is greater than or equal to
<	Is less than
<=	Is less than or equal to
~	Regex Matching
!~	Not Matching the Regex

Logical Operators

Operator	Meaning
&&	Logical And

||	Logical OR operator
!	Logical Not operator

Built-in variables

FS	Field separator
NR	Number of current record
NF	Number of fields in current record
RS	Record separator
$0	Entire input record
$n	n-th field in current record
FILENAME	Current filename

Decision Making

If statement is used to check the conditions, if the condition returns true, it performs its corresponding action otherwise perform a different set of actions from else block.

Syntactically awk's if conditions are similar to C languages if conditions.

Syntax

```
if  (  condition )
{
    statement(s) to be execute if the given condition is true
}
else
{
   statement(s) to be executed if the condition is false
}
```

Note: in case of a single statement curly '{}" braces can be avoided.

if statement with else if syntax

```
if  (  condition )
{
    statement(s) to be execute if the given condition is true
}
else if ( condition )
{
    statement(s) to be execute if the given condition is true
}
else
{
   statement(s) to be executed if the condition is false
}
```

Exercise for awk

Create a text file containing employees data given below

100	scott	accounts	accountant	10-21-2014	4000
200	richard	sales	salesman	03-11-2012	3000
300	rishi	account	executive	01-18-2016	3500
400	cyrus	production	engineer	12-03-2014	4500
500	anita	sales	caller	06-09-2016	3000
600	dave	administration	manager	03-12-2015	6000

The given columns are

EMP ID
EMP NAME
Department
Designation
Date of Join
Salary

Solve the following exercise using above mention data

1. Print details of all employees
2. Print department names from emp.txt (3rd column)
3. Print names of all employees in sales department
4. Print details of employee whose ID is 200
5. Print employee name whose ID is 200
6. Print details of employees whose salary is 4000 or more
7. Print name and salary separated by 5 spaces of employee whose salary is 4000 or more
8. Print name and salary of employee whose salary is 4000 or more
9. Print name and salary of employee whose name is scott
10. Print details of all employees separated by tab "\t"
11. Print all employee names and date of join who joined in year 2014

12. Print employee table with first line as columns names and rest with data
13. Show total number of employees
14. Sum up salaries of all employees
15. List out the files names in the directory
16. How many employees joined in the month of march of any year
17. List out all the details of files whose size is 40KB or more
18. List out all the details of files whose size is less than 100 bytes
19. List out the filenames for all files whose size is less than 100 bytes
20. Lets give a hike of 500 buks for each employee of sales department