Lab Activity 2

Friday: 27/8/2021

Activity

There are 3 questions which will be put up at the end of the lab.

Submission time - 5 pm

It is advised to submit minimum 10 mins before deadline.

Grep

- Grep is an essential Linux and Unix command.
- It is used to search text and strings in a given file. In other words, grep command searches the given file for lines containing a match to the given strings or words.
- It is one of the most useful commands on Linux and Unix-like system for developers and sysadmins. Let us see how to use grep on a Linux or Unix like system.

Setup

Text -

unix is great os. unix is open source. unix is free os.

learn operating system.

Unix linux which one you choose.

Unix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

Save in geekfile.txt

Setup

Text -

wrong

love

bear

Save in pattern.txt

Usage: grep [options] pattern [files]

-c : This prints only a count of the lines that match a pattern

-h : Display the matched lines, but do not display the filenames.

-i : Ignores, case for matching

-l : Displays list of a filenames only.

-n : Display the matched lines and their line numbers.

-v : This prints out all the lines that do not matches the pattern

-e exp : Specifies expression with this option. Can use multiple times.

-f file : Takes patterns from file, one per line.

-E : Treats pattern as an extended regular expression (ERE)

-w : Match whole word

-o : Print only the matched parts of a matching line,

with each such part on a separate output line.

-A n : Prints searched line and n lines after the result.

-B n : Prints searched line and n line before the result.

-C n : Prints searched line and n lines after before the result.

Examples

```
$grep -i "UnIx" geekfile.txt
$grep -c "unix" geekfile.txt
$grep -l "unix" * OR $grep -l "unix" f1.txt f2.txt f3.xt f4.txt
$grep -w "unix" geekfile.txt
$ grep -o "unix" geekfile.txt
$grep -o "unix" geekfile.txt | wc -l
$ grep -n "unix" geekfile.txt
```

Examples

```
$ grep -v "unix" geeks.txt

$ grep "^unix" geeks.txt

$ grep "os$" geeks.txt

$grep "..ich" geeks.txt

$grep "ea[dt]" hamlet.txt

$grep "ea[^dr ]" hamlet.txt

$grep –e "bear" –e "wrong" –e "love" hamlet.txt

$grep "^[A-Z]" hamlet.txt

$grep –f pattern.txt  geeks.txt
```

Awk

- Awk is a basic programming language.
- Awk sees input as an array.
- When awk scans over a text file, it treats each line as a record.
- Each record is broken into fields.
- Tracks using NR (number of records) and NF (number of fields) built-in variables.
- Format : pattern or keyword { actions }

Examples

For example, this gives you the line count of a file:

```
$ awk '{print}' hamlet.txt
```

```
$ awk '{print $NF}' geeks.txt hamlet.txt
```

```
$awk '{print NR,$0}' hamlet.txt
```

Similar - NF,

```
$awk '{ if($5 == "of") print $0;}' hamlet.txt
```

```
$awk 'BEGIN { for(i=1;i<=6;i++) print "square of", i, "is",i*i; }'
```

Setup

Text -

1

2

3

In inp.txt

Functions in awk

Functions are important because they allow you to write code once and reuse it throughout your work. When constructing one-liners, custom functions are a little less useful than they are in scripts, but awk defines many functions for you already. They work basically the same as any function in any other language or spreadsheet: You learn the order that the function needs information from you, and you can feed it whatever you want to get the results.

```
name(parameters) { actions }
```

Examples of builtin functions

```
$ awk 'BEGIN{print substr("Graphic Era University", 9, 8)}'
```

```
$ awk 'BEGIN{print index("Graphic", "ph"); print index("University", "abc")}'
```

```
$ awk 'BEGIN{print length("Graphic Era University")}'
```

```
$ awk 'BEGIN{string="My Nationality Is Indian"; fieldsep=" "; n=split(string, array,
fieldsep); for(i=1; i<=n; i++){printf("%s\n", array[i]);}}'
```

Examples of user defined functions

```
$ cat func.awk

func MAX(val1, val2) {

       if (val1 > val2)

               return val1

       else

               return val2

                      }

BEGIN {largest = 0}

{largest = MAX(largest, $5)}

END {print largest}
```

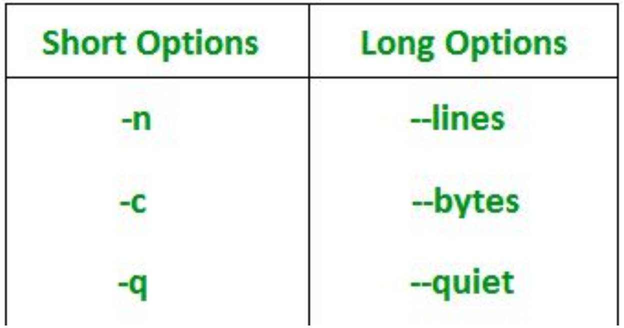
To run:awk -f func.awk data.file

Head

The head command, as the name implies, print the top N number of data of the given input. By default, it prints the first 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name.

Syntax: head [OPTION]... [FILE]...

Options :



Short Options	Long Options
-n	--lines
-c	--bytes
-q	--quiet

Examples

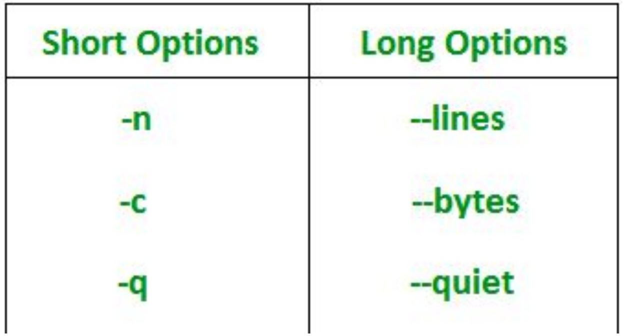
```
$ head -n 5 hamlet.txt

$ head -c 6 hamlet.txt

$ head -q  hamlet.txt
```

Tail

The tail command, as the name implies, print the bottom N number of data of the given input. By default, it prints the last 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name.

Syntax: head [OPTION]... [FILE]...

Options :



Short Options	Long Options
-n	--lines
-c	--bytes
-q	--quiet

Examples

```
$ tail -n 5 hamlet.txt

$ tail -c 6 hamlet.txt

$ tail -q hamlet.txt
```

How to use head and tail

- Print line between M and N lines(M>N):

```
$ head -n 3 state.txt | tail -1
```

- Use with piping

```
$ ls -t | head -n 3
```

- Use to get sorted mth element

```
$ ls -t | sort | head -n 3
```

Command line args like $1/$2/$3/$#

Arguments passed to a script are processed in the same order in which they're sent.

Example 1 -

```
echo "Username: $1";

echo "Age: $2";
```

Example 2 -

```
args=("$@")

echo "First->"  ${args[0]}

echo "Second->" ${args[1]}
```

Other useful args

$0	The name of script itself
$$	Process id of current shell
$*	Values of all the arguments
$#	Total number of arguments passed to script
$@	Values of all the arguments

Xargs

xargs is a Unix command which can be used to build and execute commands from standard input.

https://www.geeksforgeeks.org/xargs-command-unix/