BASH SCRIPTING

Strings

CONTENTS

Cut command

The `cut` command in Linux is used to extract specific sections from each line of input—whether from a file or piped data. It's especially handy for pulling out columns from structured text like CSVs or logs.

To cut by using the hyphen (-) as the delimiter, execute the below command:

cut -d- -f(columnNumber) **<fileName>**

If we want to use space as a delimiter, then we have to quote the space (' ') with the cut command. To cut the output by using space as delimiter, execute the command as follows:

cut -d ' ' -f(columnNumber) **<fileName>**

The '-b' option is used to cut a section of line by byte. To cut a file by its byte position, execute the command as follows:

cut -b **<byte** number**>** **<file** name**>**

The '-c' option is used to cut a specific section by character. However, these character arguments can be a number or a range of numbers, a list of comma-separated numbers, or any other character.

To cut by specified character, execute the command as follows:

cut -c **< characters>** **<file** name**>**

Cut examples

Cut first 5 bytes	`cut -b 1-5 file.txt`
Cut characters 3 to 7	`cut -c 3-7 file.txt`
Cut first and third fields (comma-separated)	`cut -d ',' -f 1,3 file.csv`
Cut all but second field	`cut -d ':' -f 2 --complement file.txt`
Use with pipe	`cat

Awk

Awk is a scripting language used for manipulating data and generating reports. The awk command programming language requires no compiling and allows the user to use variables, numeric functions, string functions, and logical operators.

awk [options] 'selection_criteria {action}' input-file

The `awk` command's main purpose is to make information retrieval and text manipulation easy to perform in Linux. This Linux command works by scanning a set of input lines in order and searches for lines matching the patterns specified by the user.

Print all lines	`awk '{print}' file.txt`
Print first column	`awk '{print $1}' file.txt`
Print first and third columns	`awk '{print $1, $3}' file.txt`
Print lines containing "error"	`awk '/error/ {print}' log.txt`
Print line numbers	`awk '{print NR, $0}' file.txt`
Sum values in column 2	`awk '{sum += $2} END {print sum}' file.txt`
Use custom delimiter (e.g., comma)	`awk -F',' '{print $1}' file.csv`

`$0`	Entire line
`$1, $2, ...`	First, second, etc. fields
`NR`	Current line number
`NF`	Number of fields in current line
`FS`	Field separator (input)
`OFS`	Output field separator
`RS / ORS`	Record separator (input/output)

What awk can do?

(a) Scans a file line by line

(b) Splits each input line into fields

(c) Compares input line/fields to pattern

(d) Performs action(s) on matched lines

1. Default behavior of Awk: By default Awk prints every line of data from the specified file.

```
$ awk '{print}' datafile
```

2. Print the lines which match the given pattern.

```
$ awk '/manager/ {print}' employee.txt
```

What awk can do?

3. Splitting a Line Into Fields : For each record i.e line, the awk command splits the record delimited by whitespace character by default and stores it in the $n variables. If the line has 4 words, it will be stored in $1, $2, $3 and $4 respectively. Also, $0 represents the whole line.

```
$ awk '{print $1,$4}' datafile
```

Built in Commands for awk

Awk's built-in variables include the field variables—$1, $2, $3, and so on ($0 is the entire line) — that break a line of text into individual words or pieces called fields.

- **NR:** NR command keeps a current count of the number of input records. Remember that records are usually lines. Awk command performs the pattern/action statements once for each record in a file.
- **NF:** NF command keeps a count of the number of fields within the current input record.
- **FS:** FS command contains the field separator character which is used to divide fields on the input line. The default is "white space", meaning space and tab characters. FS can be reassigned to another character (typically in BEGIN) to change the field separator.
- **RS:** RS command stores the current record separator character. Since, by default, an input line is the input record, the default record separator character is a newline.
- **OFS:** OFS command stores the output field separator, which separates the fields when Awk prints them. The default is a blank space. Whenever print has several parameters separated with commas, it will print the value of OFS in between each parameter.
- **ORS:** ORS command stores the output record separator, which separates the output lines when Awk prints them. The default is a newline character. print automatically outputs the contents of ORS at the end of whatever it is given to print.

Find Duplicates in a Row

```
bosko@bosko-vm:~$ cat ~/answers.txt
a,1,1
b,3,4
c,5,2
d,6,1
e,3,3
f,3,7
bosko@bosko-vm:~$ awk -F ',' '{if($2==$3){print $1","$2","$3} else {print "No Duplica
tes"}}' answers.txt
a,1,1
No Duplicates
No Duplicates
No Duplicates
e,3,3
No Duplicates
```

Print the table attributes row-wise

```
bosko@bosko-vm:~$ awk '{i=0; while(i<=NF) { print i ":"$i; i++;}}' employees.txt
0:Adeline Bird clerk $654
1:Adeline
2:Bird
3:clerk
4:$654
0:Weston Bradley clerk $789
1:Weston
2:Bradley
3:clerk
4:$789
```

For loop and Awk

```
bosko@bosko-vm:~$ awk 'BEGIN{for(i=1; i<=10; i++) print "The square of", i, "is", i*i
;}'
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
The square of 6 is 36
The square of 7 is 49
The square of 8 is 64
The square of 9 is 81
The square of 10 is 100
```

Awk-Patterns

Inserting a pattern in front of an action in `awk` acts as a selector. The selector determines whether to perform an action or not. The following expressions can serve as patterns:

- Regular expressions.
- Arithmetic relational expressions.
- String-valued expressions.
- Arbitrary Boolean combinations of the expressions above.

Regex

Symbol	Meaning
`.`	Any single character except newline
`^`	Start of line
`$`	End of line
`*`	Zero or more of the preceding element
`+`	One or more of the preceding element
`?`	Zero or one of the preceding element
`{n}`	Exactly *n* repetitions
`{n,}`	*n* or more repetitions
`{n,m}`	Between *n* and *m* repetitions
`[]`	Match any one character in brackets
`[^]`	Match any character *not* in brackets
`	`
`()`	Group expressions
`\d`	Digit (0–9)
`\D`	Non-digit

Regex with Awk

Match lines starting with "INFO"	`awk '/^INFO/' syslog`	Anchors to the beginning of the line
Match lines ending with ".com"	`awk '/\.com$/' emails.txt`	Anchors to the end of the line
Match lines with a 3-digit number	`awk '/[0-9]{3}/' data.txt`	Finds any 3-digit sequence
Match lines with exactly 5 letters	`awk '/^[a-zA-Z]{5}$/' words.txt`	Matches lines with only 5 letters

Awk-Regex

Regular Expression Patterns

Regular expression patterns are the simplest form of expressions containing a string of characters enclosed in slashes. It can be a sequence of letters, numbers, or a combination of both.

In the following example, the program outputs all the lines starting with "A". If the specified string is a part of a larger word, it is also printed.

```
awk '$1 ~ /^A/ {print $0}' employees.txt
```

```
bosko@bosko-vm:~$ awk '$1 ~ /^A/ {print $0}' employees.txt
Adeline Bird clerk $654
Anaya Rice director $1334
Andre Wilkins clerk $446
Allen Thomas IT $7854
```

AWK-Relational Expression

Relational Expression Patterns

Another type of `awk` patterns are relational expression patterns. The relational expression patterns involve using any of the following relational operators: <, <=, ==, !=, >=, and >.

Following is an example of an `awk` relational expression:

```
awk 'BEGIN { a = 10; b = 10; if (a == b) print "a == b" }'
```

```
bosko@bosko-vm:~$ awk 'BEGIN { a = 10; b = 10; if (a == b) print "a = b" }'
a = b
bosko@bosko-vm:~$
```

Awk-Range

A range pattern is a pattern consisting of two patterns separated by a comma. Range patterns perform the specified action for each line between the occurrence of pattern one and pattern two.

For example:

```
awk '/clerk/, /manager/ {print $1, $2}' employees.txt
```

```
bosko@bosko-vm:~$ awk '/clerk/, /manager/ {print $1, $2}' employees.txt
Adeline Bird
Weston Bradley
Anaya Rice
Andre Wilkins
Ian Norris
Omar Landry
Harley Edwards
Marie Snow
```

Awk-Special Expressions

Special expression patterns include `BEGIN` and `END` which denote program initialization and end. The `BEGIN` pattern matches the beginning of the input, before the first record is processed. The `END` pattern matches the end of the input, after the last record has been processed.

For example, you can instruct `awk` to display a message at the beginning and at the end of the process:

```
awk 'BEGIN { print "List of debtors:" }; {print $1, $2}; END {print "End of the debtor list"}' debtors.txt
```

```
bosko@bosko-vm:~$ awk 'BEGIN { print "List of debtors:" }; {print $1, $2}; END {print
 "End of the debtor list"}' employee-debt.txt
List of debtors:
employees-debt
Maria 2334
John 5646
Gendry 8998
Bill 133
Edgar 758
Jake 0
Milford 0
End of the debtor list
```

Awk combining patterns

Combining Patterns

The `awk` command allows users to combine two or more patterns using logical operators. The combined patterns can be any Boolean combination of patterns. The logical operators for combining patterns are:

- `||` (or)
- `&&` (and)
- `!` (not)

For example:

```
awk '$3 > 10 && $4 < 20 {print $1, $2}' employees.txt
```

```
bosko@bosko-vm:~$ awk '$3 > 10 && $4 < 20 {print $1, $2}' employees.txt
Adeline Bird
Weston Bradley
Anaya Rice
Andre Wilkins
Ian Norris
Omar Landry
Harley Edwards
Marie Snow
Haylee Sweeney
Isis Ware
Jaelyn Khan
Julio Estrada
Allen Thomas
Isiah Hester
Jan Harper
Zachery Richard
```

Sed-stream editor

```
sed OPTIONS... [SCRIPT] [INPUTFILE...]
```

- SED is a powerful text stream editor. Can do insertion, deletion, search and replace(substitution).
- SED command in unix supports regular expression which allows it perform complex pattern matching.

It reads input line by line, applies operations like **search**, **replace**, **insert**, **delete**, and **print**, and outputs the result.

Sed Examples

Command	Purpose	Example
`s/old/new/`	Replace first occurrence of "old" with "new"	`sed 's/foo/bar/' file.txt`
`s/old/new/g`	Replace **all** occurrences	`sed 's/foo/bar/g' file.txt`
`s/old/new/2`	Replace **second** occurrence only	`sed 's/foo/bar/2' file.txt`
`3s/old/new/`	Replace on **line 3** only	`sed '3s/foo/bar/' file.txt`
`/pattern/d`	Delete lines matching pattern	`sed '/error/d' file.txt`
`5d`	Delete **line 5**	`sed '5d' file.txt`
`1,3d`	Delete lines 1 to 3	`sed '1,3d' file.txt`
`/pattern/p`	Print lines matching pattern	`sed -n '/error/p' file.txt`
`i\text`	Insert text **before** matched line	`sed '/pattern/i\New line' file.txt`
`a\text`	Append text **after** matched line	`sed '/pattern/a\New line' file.txt`

Sed-Replacing or substituting 1st occurrence of a pattern

```
sed 's/unix/linux/' textfile
```

Sed- Replace the nth occurrence of a pattern

```
sed 's/unix/linux/2' textfile
```

Sed - Replacing or substituting all occurrence of a pattern

```
sed 's/unix/linux/g' textfile
```

Sed- replacing from nth occurrence of a pattern to last

```
sed 's/unix/linux/3g' textfile
```

Sed- replacing string on the nth line

```
sed '3s/unix/linux/' textfile
```

Sed- replacing string on a range of lines

```
sed '1,3 s/unix/linux/' textfile
```

Sed-deleting

1. To Delete a particular line say n in this example

```
Syntax:
$ sed 'nd' filename.txt
Example:
  $ sed '5d' filename.txt
```

2. To Delete a last line

```
Syntax:
  $ sed '$d' filename.txt
```

3. To Delete line from range x to y

```
Syntax:
$ sed 'x,yd' filename.txt
Example:
  $ sed '3,6d' filename.txt
```

4. To Delete from nth to last line

```
Syntax:
$ sed 'nth,$d' filename.txt
Example:
  $ sed '12,$d' filename.txt
```

5. To Delete pattern matching line

```
Syntax:
$ sed '/pattern/d'
filename.txt
Example:
  $ sed '/abc/d'
  filename.txt
```

sed-Insert a line

1 – Insert one blank line after each line –

```
sed G a.txt
```

2 – To insert two blank lines –

```
sed 'G;G' a.txt
```

3 – Insert 5 spaces to the left of every lines –

```
sed 's/^/     /' a.txt
```

Sed-numbering lines

1 – Number each line of a file (left alignment). **=** is used to number the line. \t is used for tab between number and sentence –

```
sed =  a.txt | sed'N;s/\n/\t/'
```

3 – Number each line of file, only if line is not blank –

```
sed '/./=' a.txt | sed '/./N; s/\n/ /'
```

Head Command

The Linux `head` command prints the first lines of one or more files (or piped data) to standard output. By default, it shows the first 10 lines. However, `head` provides several arguments you can use to modify the output.

1. Display first 10 lines of a file

```
$ head example1.txt
```

2. show the first 4 lines

```
$ head -n 4 example1.txt
```

3. to see 20 bytes of output of the sample file, you would run the command:

```
$ head -c 20 example1.txt
```

4. first lines of files *example1.txt* and *example2.txt*, you would type:

```
$ head example1.txt example2.txt
```

5. to list files in the */etc* directory using the ls command and print 10 entries, you would run the command:

```
$ ls /etc | head
```

Tail Command

Linux tail command is used to display the last ten lines of one or more files. Its main purpose is to read the error message. By default, it displays the last ten lines of a file.

1. By default “tail” prints the last 10 lines of a file, then exits.

```
$ tail /path/to/file
```

2. the -n option comes handy, to choose specific number of lines instead of the default 10

```
$ tail -n 4 /etc/group
```

Grep

`grep` is a powerful command-line utility in Unix/Linux used to **search for text patterns within files**. Its name stands for “Global Regular Expression Print.” You can use it to find specific strings, match patterns with regular expressions, and filter output based on content.

Command	Description
`grep "error" logfile.txt`	Searches for lines containing "error" in `logfile.txt`
`grep -i "hello" file.txt`	Case-insensitive search for "hello"
`grep -r "main()" ./src`	Recursively searches for "main()" in the `src` folder
`grep -n "TODO" script.py`	Shows line numbers where "TODO" appears in `script.py`
`grep -v "DEBUG" app.log`	Shows all lines *except* those containing "DEBUG"
`grep -c "function" code.js`	Counts how many times "function" appears
`grep "^#" config.txt`	Finds lines that start with a # (comments)
`grep "[0-9]\{3\}-[0-9]\{3\}-[0-9]\{4\}" contacts.txt`	Matches US phone numbers like 123-456-7890

Assignment

Write a script to use grep,awk/sed to find the network interfaces and their associated ip addresses. The output should be in the form of a table

IPv4:

Mask:

Broadcast:

Assignment