DAY 6

The **awk** command is a powerful tool for text processing in Unix-like operating systems. It's a scripting language that's designed for processing structured text like CSV, TSV, or log files.

Text Processing: awk is often used to process text files. For example, awk '{print $1}' filename will print the first field (column) of each line in the file.

Pattern Scanning: awk can scan a file line by line for a pattern. When a line matches one of the patterns, awk performs specified actions on that line. For example, awk '/pattern/ {print $0}' filename will print the lines that match the pattern.

Text Transformation: awk can transform the data in a text file. For example, awk '{$1=""; print $0}' filename will remove the first field from each line in the file.

Calculations: awk can perform mathematical calculations. For example, awk '{sum+=$1} END {print sum}' filename will sum up the values of the first field in the file.

Generating Reports: awk can generate reports based on the data in a text file. For example, awk '{count[$1]++} END {for (word in count) print word, count[word]}' filename will print each unique word in the first field and its count.

The **sed** command, short for Stream EDitor, is a powerful utility in Unix-like operating systems. It's primarily used for text manipulation and transformation.

Find and Replace Text: The most common use of sed is to find and replace text in a file. For example, sed 's/foo/bar/g' filename will replace all occurrences of 'foo' with 'bar' in the specified file.

Delete Lines: sed can be used to delete lines in a file that match a specific pattern. For example, sed '/pattern_to_match/d' filename will delete all lines containing 'pattern_to_match' from the file.

Insert and Append Text: sed can insert text before a line or append text after a line in a file. For example, sed '3i\This is the inserted text.' filename will insert 'This is the inserted text.' before the third line of the file.

Modify Lines: sed can change any text in a file based on a regular expression. For example, sed 's/^/prefix_/' filename will add 'prefix_' at the beginning of every line in the file.

Print Lines: sed can print only those lines that match a specific pattern. For example, sed -n '/pattern_to_match/p' filename will print only the lines that contain 'pattern_to_match'.

SHELL VARIABLES

A variable is a string of characters to which a value can be assigned. In Linux, once a value is assigned to a variable, the variable can be accessed by prefixing the variable name with a “$”.

Linux follows the below naming convention for shell variables.

1. Variable names must be all in upper case.

2. Variable names cannot start with a number (0-9).

3. Variable name can contain alphabets(a-z, A-Z), numbers(0-9) and the special character, "_".

Some examples of valid variable names are, APPNAME, USER1, Admin_1, _day, etc.,

Some examples of invalid variable names are 1APP, 1-user, !ADMIN, etc.,

CONT.,

```
[root@machine1 /]$ track=Wintel
[root@machine1 /]$ echo $track
Wintel
[root@machine1 /]$ echo track
track
[root@machine1 /]$ track=Linux
[root@machine1 /]$ echo $track
Linux
[root@machine1 /]$ unset track
[root@machine1 /]$ echo $track
[root@machine1 /]$ readonly stream=linux
[root@machine1 /]$ echo $stream
linux
[root@machine1 /]$ stream=windows
/bin/bash: stream: This variable is read only.
```

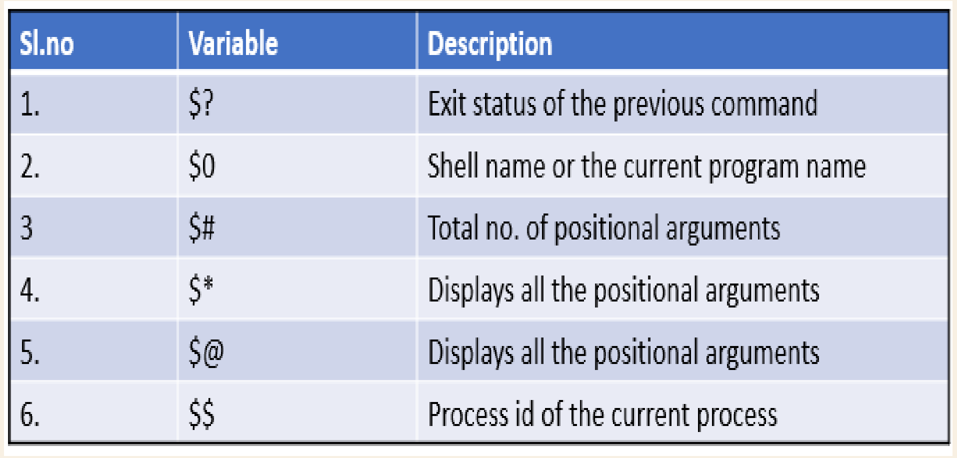
“Wintel” is the value that is assigned to the variable “track”. Only when accessed with the $ symbol before the name of the variable, the variable got expanded into its value.

Otherwise, "track" is interpreted like any other normal string. This is an example for a user-defined variable. These are local to the environment where they are set.

To delete this variable, you can use the "**unset**" command. Otherwise, if you want to update the value of a variable without deleting it, we can define the variable again with the new value.

In some cases, we might not want any user having access to the variable to change its value. In this case, we can declare a variable as "read only" while creating it.

SPECIAL VARIABLES



Sl.no	Variable	Description
1.	$?	Exit status of the previous command
2.	$0	Shell name or the current program name
3	$#	Total no. of positional arguments
4.	$*	Displays all the positional arguments
5.	$@	Displays all the positional arguments
6.	$$	Process id of the current process

In Linux, there are special variables which are reserved for specific functions.

Each of them hold a special value in Linux.

These special variables cannot be assigned with any user-defined values.

These special variables can be used with commands as well as scripts.

The following table shows the special variables in Unix and their functions.

Command line arguments:

One of the simple yet useful way of supplying parameters to a Linux shell script is by passing them as command line arguments.

These parameters can be text, number, filename or any data. Any number of parameters can be supplied to a Linux shell script.

$0 is a special positional parameter which holds the name of the script/program that is being executed.

```
[root@machine1 /]# cat script.sh
#!/bin/bash
echo "The name of the script is $0"
echo "The first argument is $1"
echo "The second argument is $2"
echo "The third argument is $3"
echo "The total number of arguments are $#"
echo "The arguments are $@"
echo "The arguments are $*"
```

Positional parameters can also be just set on the terminal using "set" command.

```
[root@machine1 /]# set file1 file2
[root@machine1 /]# echo $1
file1
[root@machine1 /]# echo $2
file2
[root@machine1 /]# echo $@
file1 file2
[root@machine1 /]# echo $*
file1 file2
```

The difference between $@ and $* be executing the below commands.

```
[root@machine1 /]# cat "$@"
file1 exists
file2 exists
[root@machine1 /]# cat "$*"
cat: file1 file2: No such file or directory
```

$@ stores the arguments as separate strings.

RESOURCE UTILIZATION

```
#!/bin/bash

# CPU utilization
echo "CPU Utilization:"
top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk '{print 100 - $1"%"}'

# Memory utilization
echo "Memory Utilization:"
free -m | awk 'NR==2{printf "%.2f%%\n", $3*100/$2 }'

# Disk utilization
echo "Disk Utilization:"
df -h | awk '$NF=="/"{printf "%s\n", $5}'
```

#!/bin/bash is known as a shebang or hashbang. It is used in scripts to indicate an interpreter for execution under UNIX and UNIX-like operating systems.

#! is a two-byte magic number, a special marker that designates a file type, or in this case an executable shell script (starting with #! is actually interpreted as a directive to the program loader to run the rest of the text file as a program).

/bin/bash is the path to the interpreter that should be used to execute the script. In this case, it's the Bash shell.

So, #!/bin/bash at the start of a script tells the system that this script should be executed with the Bash shell, even if the shell is not the current shell for the user.

For CPU utilization, it uses the top command with -bn1 option to run top once in batch mode and then grep to find the line containing "Cpu(s)". The sed command is used to extract the idle CPU percentage and awk is used to subtract it from 100 to get the CPU utilization.

For memory utilization, it uses the free command with -m option to display the amount of free and used memory in the system in MB. awk is used to calculate the percentage of used memory.

For disk utilization, it uses the df command with -h option to report file system disk space usage in human-readable format. awk is used to print the percentage of disk usage for the root directory ("/").

DAY 6 - END