Shell Scripting

A file descriptor is an integer number to identify ***STDIN, STDOUT and STDERR.***

|  |  |
| --- | --- |
| 0 | STDIN |
| 1 | STDOUT |
| 2 | STDERR |

& - to store success output and error output into a same file.

ls & > file.txt

Read a file content by opening it – Using vi and vim editor

Read a file content without opening it – cat,less and more

Read a file content with conditions – tail, grep, awk

more -2 file.txt: top 2 lines

more +4 file.txt: from fourth line it starts displaying.

head filename.txt: Display top 10 lines by default.

Display a range of lines

head -12 lines xyz.txt | tail -7 : Display 6 to 12 lines

awk ‘NR>=6 && NR<=12 {print}’ xyz.txt

sed -n ‘6,12p’ xyz.txt

Grep

grep command: used to search for a given string.

***Basic Options***: -i -w -v -o -n -c -A -B -C -r -l -h

***Advanced Options***: -e -f and -E

grep [options] “string/pattern” file/files

-i: To ignore case for matching/searching.

-w: To match a whole word.

-v: To display lines that are not matching with a given string

-o: To display only matched part from matched lines

-n: To display matched line numbers

-c: To display matched number of lines

-r: to search under its current directory and sub-directory

-l : to display only filenames

grep -f “search string or pattern” filename

-f : takes search string from a file

grep -e “string1” -e “string2” filename

grep -E “line|above|bash|shell|python” test.txt

Pattern is a string, and it represents more than one string.

grep -E [options] [pattern] file.txt

Pattern is a string, and it represents more than one string.

xy|pq : matches xy or pq in a file

^xyz – matches lines starting with xyz

xyz$ - matches line ending with xyz

^$- to get empty lines (very useful in sed command

\ - To remove the special purpose of any symbol. grep -E “\^” test.txt

. – Match any one character: grep -E “t..s” file.txt it matches any characters with two characters in btw t and s.

\b – Matches empty string at the edge of a word.

grep -E “line\b” test.txt

? – preceding character is optional and matches utmost once.

\* - matches any number of times

+ - Should match atleast one or more times

\b – Matches empty string at the edge of a word.

grep -E “line\b” test.txt

Text

Description automatically generated

Cut command is used to extract parts of each line from a file

-d – delimiter

-c- character

-b – bytes

-s – suppress

--complement- apart from that everything

--output-delimiter – to define a custom delimiter

By default, delimiter is tab

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Complement gives everything apart from the one mentioned in the command.

***awk -F [options] ‘[selection\_criteria] {action}’ <file\_name>***

tr command for shell scripting

tr is used for translate or delete given set of characters from the input

tr ‘[: upper:]’ ‘[: lower:]’ <xyz.txt>

tr [: lower:]’ ‘[: upper:]’ <xyz.txt>

tr ‘i’ ‘I’ <xyz.txt>

tr ‘i’ ‘\_’ <xyz.txt>

Example: [: lower:], [a-z],[A-Z], [:digit:], [0-9],[:space:]

-d: deletes given set of characters

**tee** command for shell scripting

**ls -ltr | tee xyz.txt**

Simultaneously to display output and store the output

**ls -ltr | tee -a xyz.txt**

append an output to a file

***Variables in Shell:***

* Variables should be only 20 characters, they can contain [a-z], [A-Z], [0-9] and \_
* While defining variables there shouldn’t be any spaces between variable and its value
* To store the output of a shell command, use $(command) or `command`
* The store the value of one variable into other variables we use $
* System variables will always be CAPITAL letters and custom variables will be small letters

***Script for docker status:***

#!/bin/bash

docker\_status=$(systemctl status docker | awk '/Active/ {print$3}' | tr -d ["()"])

docker\_version=$(docker -v | awk '{print$3}' | tr -d [,])

echo "The docker status is ${docker\_status} and docker version is ${docker\_version}"

**Advanced usage of echo command**

echo -e “Message” where -e is the escape character.

\n – New line

\t – Tab line

-n – The cursor stays at the same line

\b – backspace it comes one word back

\r – Carriage return

\v – vertical tab

echo -e "This is first line\nThis is second line"

***Heredoc***

This is used to store multi-lines or multi line block.

command <<Delimiter

line1

line2

line3

Delimiter

***Heredoc for multi lines command***

**#!/bin/bash**

**cat<<EOF**

**The name of user is: $USER**

**The directory is: $HOME**

**EOF**

***To redirect the output to separate file***

**#!/bin/bash**

**cat<<EOF > names.txt**

**The name of user is: $USER**

**The directory is: $HOME**

**EOF**

**HERE command:** This is for string

**tr [a-z] [A-Z] <<<$(docker -v)**

***Commenting in shell scripts***

**#!/bin/bash**

**#Author: Mahesh Mamidibathula**

**<<MYCOMM**

**This script is for mutli comments**

**Multi comments**

**MYCOMM**

**: '**

**aLL WELL**

**ASKDAS**

**ASDK**

**'**

**echo " The docker version is : $(docker -v)"**

***Debugging shell scripts***

Set command can be used to debug shell scripts.

-n: This is purely for syntax checks

-x: This displays line by line of shell scripting

-e: This stops the execution of shell script the moment it encounters an error

-v: This is verbose which display the variable values

***Exit status***

Non-zero means and usually exit status is stored in $?

***Basic operations on Strings***

***Defining a string***

Graphical user interface, text

Description automatically generated

***Length of string: $#***

Graphical user interface, text, application

Description automatically generated

***String replacement:***

Graphical user interface, text, chat or text message

Description automatically generated

***String Slicing:***

Graphical user interface, text

Description automatically generated

***String operations on file paths:***

***realpath: Coverts each filename argument to an absolute pathname but it doesn’t validate the path.***

***basename:***

* Strips directory information
* Strips suffixes from file names

***dirname:*** It will delete any suffix beginning with last slash character and return the result.

***Read:*** for obtaining input variables

Graphical user interface, text, application

Description automatically generated

***REPLY*** is the default variable in which the input is stored.

Text

Description automatically generated

**Command line arguments**

$# - Total no of arguments or count of arguments

$0 – Name of shell script, $1,$2… are first second and .. arguments respectively

$@ - All command line arguments

$\* - all command line arguments

***Case statements***

**#!/usr/bin/env bash**

**set -x**

**read -p "Enter the first number" a**

**read -p "Enter the second number" b**

**read -p "Enter the choices"**

**echo "+++++++++++++++++++++"**

**read "1. For Addition,2.For substraction,3.For Multiplication, 4. For Division"**

**echo "====================="**

**case ${REPLY} in**

**1)**

**echo "The value of addition is : $((a+b))"**

**;;**

**2)**

**echo "The value of substraction is : $((a-b))"**

**;;**

**3)**

**echo "The value of mutliplication is: $((a\*b))"**

**;;**

**4)**

**echo "The value of division is: $((a/b))"**

**;;**

**\*)**

**echo "Invalid choice of input"**

**;;**

**esac**

***Questions***

1. How to display from 6 to 12 lines in a file?
2. What are file identifiers for STDIN, STDOUT and STDERR?
3. How to search for a string in multiple files?
4. Display the usage of i,w,v,o,c,n,

Test command

* test condition or [condition] or [[ condition ]].
* [[]] works with bash/ksh/zsh shells.

***Numbers***

[[ int1 -eq int2 ]] – True if they are equal

[[ int1 -ne int2 ]] - False if they are unequal

[[ int1 -lt int2 ]] – True if int1 is less than int2 or it returns false

[[ int1 -gt int2 ]]- True if int1 is greater than int2 or it returns false

[[ int1 -ge int2 ]] – True if int1 is greater than or equal to int2 or it returns false

[[ !int1 -eq int2 ]] – It reverse the result

***Strings***

[[ -z str ]] – It returns true if length of str is zero else false

[[ -n str ]] – It returns true if length of str is non-zero else false

[[ str1 == str2 ]] – It returns true if both the strings are equal else false

[[ str1 != str2]] – It returns true if both the strings are equal else false

**File test operators**

[[ -d file ]] – It returns true if file/path is directory else false

[[ -f file ]] – It returns true if file/path is a file else false

[[ -e file ]] – It returns true if file/path exists else false

[[ -r file ]] – It returns true if file/path is readable else false

[[ -w file ]] – It returns true if file/path is writable else false

[[ -x file ]] – It returns true if file/path is executable else false

***Command chaining operators***

* ; -
* &&
* ||
* && ||

Behavior of operators

* cmd1;cmd2 – Run command 1 then command 2 regardless of whether command1 is success or not
* cmd1&&cmd2 – Run cmd2 if cmd1 succeeds
* cmd1||cmd2- Run cmd2 if cmd1 fails
* cmd1&&cmd2 || cmd3 – Run cmd3 if cmd1&&cmd2 fails

A screenshot of a computer

Description automatically generated

***This is to nullify an output of first command: 2>&1 1>/dev/null***

which docker && echo "Docker exists" || echo "Docker uninstalled"

***Executing block of code using {}***

sudo -v to check if the user has sudo previliges

***if conditional statements***

**#!/usr/bin/env bash**

**echo "This script is to check for docker status and start if docker service is stopped"**

**if [[ $(id -u) -eq 0 ]]**

**then**

**echo "The user is a root user"**

**if systemctl status docker 2>/dev/null 1>/dev/null**

**then**

**echo "Docker already in start phase"**

**else**

**systemctl start docker**

**fi**

**else**

**if sudo -v 2>/dev/null 1>/dev/null**

**then**

**if systemctl status docker 2>/dev/null 1>/dev/null**

**then**

**echo "Docker already in start Phase"**

**else**

**systemctl start docker**

**fi**

**else**

**echo "You don't have sudo previliges"**

**fi**

**fi**

* **Logical AND Operator && or -a**
* **Logical OR operator || or -o**
* **Logical NOT Operator!**

**What is the difference between [] and [[]]?**

[[ ]] is always beneficial.

#!/usr/bin/env bash

user=` id -un `

echo "${user}"

if [[ $user = vagrant ]]

then

echo "The user is $user"

fi

if [ "$user" = "vagrant" ]

then

echo "The user is $user"

fi

***at and crontab (Scheduling):***

***atq***: to check the list of queued jobs

***atrm***: to remove a queued job

* at 10.00 A.M
* at 10.00 A.M
* at 10.00 A.M July 25
* at 10.00 A.M 06/22/2015
* at 10.00 A.M 6.22.2015
* at 10.00 A.M next month
* at 10.00 A.M tomorrow

“***at***” is used to schedule jobs at a specified time to execute only once. Once it executes done and it not periodic.

***Crontab***: This is to schedule jobs periodically

**minute(s) hour(s) day(s) month(s) weekday(s)**

**crontab -e**: To schedule a job.

**crontab -l**: To list the jobs.

**crontab -r**: To remove jobs.

0-6: (Mon-Sat)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Second(s)** | **Minute(s)** | **Hour(S)** | **Day(s)** | **Weekday(s)** |  | |  |
| \* | \* | \* | \* | 0 | {script.sh} | Run script on Sunday | |
| 0 | 0 | \*/2 | \* | \* | {script.sh} | Run script every 2 hours | |
| \* | \* | 9,21 | \* | \* | {script.sh} | Everyday at 9AM and 9 PM | |
|  |  |  |  |  | {script.sh} |  | |
|  |  |  |  |  | {script.sh} |  | |
| @Reboot | | | | | {script.sh} | During the reboot | |
| @Yearly | | | | | {script.sh} | Runs every year | |

free -b to check free size in bytes

free -mt : to check free size in bytes

To send automatic mail alert

To check ram size: free -m (to show in MB), free -g ( to show in gb), free -mt( To check total size).

Associative arrays are arrays with indexed values as strings.

**# An array is an indexed based array**

**a=(78 52 12 64 32 14 71)**

**echo "To display the contents of an array is : ${a[@]}"**

**echo "To give the count of array is: ${#a[@]}"**

**echo "To display the indexes of array: ${!a[@]}"**

**echo "To display from the second element to last element: ${a[@]:1:3}"**

**# To display indexed array**

**declare -A myindexedarry**

**myindexedarray=([name]="Shell Scripting" [version]=4.4.0)**

**echo "${myindexedarray[@]}"**

***for loops:***

***#!/usr/bin/env bash***

***for ((cnt=1;cnt<10;cnt++))***

***do***

***echo "Number is : ${cnt}"***

***done***

***Continue and break statements***

***Continue:*** This statement will help to continue further processing of loop.

***Break***: This statement breaks out of the loop.

#!/usr/bin/env bash

for each in ` seq 1 10 `

do

echo $each

done

for each in ` seq 1 10 `

do

if [[ $each -eq 5 ]];then

continue

fi

echo $each

done

***for loop with arrays***

***awk scripting***

Syntax: awk options ‘pattern {logic}’

* $0, $1, $2…………
* NR 🡪 No of records
* NF 🡪 No of fields
* FILENAME

awk scripting

awk options ‘pattern/condition {action}’ filename

awk ‘BEGIN {Begin\_Action} pattern/condition{logic\_action} END{end\_action}’

awk command with options, action, and basic variables

**sed command is a string editor**

* Viewing file content
* Searching
* Find and replace
* Insertion or deletion

**sed also performs regular expression which allows it to perform complex pattern matching**

**advantage of sed over vi/vim editors**

* **Edit without opening it**

**sed [options] commands [file-to-work-with-sed]**

How sed works?

* Read a line from input stream
* Execute sed command on a line
* Display result on an output stream

**Viewing file content and deleting file content**

* sed ‘ ’ file.txt = cat command
* sed -n ‘p’ file.txt (-n suppress output)
* sed -n ‘3p’ file.txt (3rd line)
* sed -n ‘$p’ file.txt – to print last line
* sed -n ‘3,10p’ file.txt – to print from third to 10th line
* sed -n ’30, $p’ file.txt – to print from 30 to last line
* sed -n ’12, +7p’ file.txt – from 12 to next 7 lines
* sed -n ‘1-3p’ file.txt – lines with 3 interval positions
* sed -n ‘1-2p’ file.txt – Alternative lines
* sed ‘4, $d’ file.txt – Deletes 4th to last line
* sed ’10,45d’ file.txt – Deletes 10 to 45 lines
* sed ’10, +12d’ file.txt – Deletes from 10 to 12 more lines

if we want to change original file we have to include ‘-i’ option.

* sed -i ‘4,10d’ file.txt – delete from 4 to 10 lines
* sed -i.back ‘3,$d’ file.txt – take a backup before performing deletion of 3 to last lines