**UNIX SHELL SCRIPTING**

NAME: Prasanna Pawar



|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Command | Argument | Purpose |
| 1 | *vi* | *vi file\_name* | 1. To create file 2. To editing existing file (Press *I*, make changes, press *ESC*, *:wq!* (Save and Exit), *:q!*(Exit without saving) |
| 2 | *help* | *[command] --help* | Provides a quick summary of a command’s usage. Displays basic options and their descriptions. |
| 3 | *man* | *man [command]* | Provides a detailed manual for a command. Includes sections like description, options, examples, exit status, and author. |
| 4 | *cat* | *cat file\_name* | Viewing Content of File (all the data) |
| *cat -n file\_name* | number all output lines |
| *cat -b file\_name* | number nonempty output lines |
| *cat -s file\_name* | suppress repeated empty output lines |
| 5 | *more* | *more file\_name* | It also show the content in file but only limited (fit in to 1 window that much only) |
| 6 | *pwd* | *pwd file\_name* | It shows the current path or currently where we are (current directory) |
| 7 | *ls* |  | It list down what we have in our directory (current) |
| *ls -l* | It list down what we have in our directory (current) but in more details (like date of file creation..etc) |
| *ls -lt* | It list down all files according to Date of Creation (ASC) |
| *ls -ltr* | It list down all files according to Date of Creation (DESC) |
| *ls -a* | It shows the hidden files |
| 8 | *wc* | *wc file\_name* | It show the count of file (line count, word count, charcater count) |
| *wc -l file\_name* | It will show the count of Lines |
| *wc -m file\_name* | It will show the count of Charcaters |
| *wc -c file\_name* | It will show the Bytes count |
| *wc -w file\_name* | It will show the count words |
| 9 | *mv*  *mv* | *mv file\_name /path/to/move/newfile\_name* | Move with New Name |
| *mv file1 file2 /path/to/move/* | Multiple files to move |
| *mv file\_name destination\_directory* | To move file |
| *mv myfolder /path/to/move/* | moving directory |
| *mv file\_name new\_file\_name* | To rename the file |
| 10 | *cp* | *cp file\_name file\_name1* | Copy the file |
| 11 | *rm* | *rm file\_name* | To remove the file |
| *rmdir directory\_name* | remove directory but directory must empty |
| *rm -r directory\_name* | remove directory even if it is not empty |
| 12 | *mkdir* |  | Creating Directories (means Folder) |
| *mkdir Directory\_Name* | For creating Directory |
| *mkdir Directory1 Directory2… DirectoryN* | It create multiple directories at a time |
| *mkdir -p ./ Directory1/Directory2* | '-p' helps to Create parent directories as needed (If any Directory we give in path not exist then it create) './' It means start where you are |
| 13 | *command -r* |  | For doing operations on Directory |
| *cp -r Source Destination* | Copy the file |
| *cp -r Source Source1 Source2 Destination* | Copy the Multiple File |
| *cp -r File1 Directory1 File2 Directory2 Destination* | Copy Mix files and Directories |
| *rm -r Directory  rmdir Directory1 Directory2* | Removing Directory or Directories  For *rmdir* Directory must be empty otherwise it's not work |
| *mv Directory Directory\_New\_Name* | Rename the directory Name |
| 14 | *cd* | *cd* | To go to the home directory, just use *cd* with no parameters |
| *cd /user/home/tsrdj* | To go to the */user/home/tsrdj* Directory (It is called Full Path) |
| *cd ./Directory\_Name* | To go 1 Directory Forward (It is called Relative Path) |
| *cd-* | To go last used Directory |
| *cd ..* | To go to the 1 Directory back |
| *cd ../..* | To go to the 2 Directory back |
| *cd ../../..* | To go to the 3 Directory back (We can *../../..* As per our need how many directory we have to go back) |
| 15 | *head*  *head* | *head files\_name* | It will show the FIRST 10 lines of its standard output |
| *head -n file\_name* | If we want FIRST 50 lines replace *n* with 50 (Ex. *head -n emp\_data*) |
| 16 | *tail* | *tail file\_name* | It will show the LAST 10 lines of its standard output |
| *tail -n file\_name* | If we want LAST 50 lines replace *n* with 50 (Ex. *tail -n emp\_data*) |
| 17 | *grep* | *grep <search\_word> file\_name* | It helps to search in file (It is LIKE in SQL) |
| *grep -n <word> file\_name* | It search and also give the LINE number of our searched result |
| *grep -i <word> file\_name* | It search but Case Insensitive |
| *grep -v <word> file\_name* | It search except the given Search (It is like NOT EQUAL TO) |
| For Advance Search *1. grep "^search" file\_name 2. grep "search$" file\_name* | 1. when we only know the starting 2. when we only know the ending |
| *grep -l <word> \** | It shows the list of all the files contain that word |
| 18 | *tr (Replace)* | *tr 'set1' 'set2' tr -s 'set'* | It changes all the characters in one set into characters in a second set. Sometimes it deletes sets of charcters |
| For example we have to change all charcters into lowercase then $cat EMP\_FILE | tr 'a-z' 'A-Z' |
| 19 | *sort* | *sort file\_name* | **Rules:** 1) Lines starting with a number will appear before lines starting with a letter, 2) Lines starting with a letter that appears earlier in the alphabet will appear before lines starting with a letter that appears later in the alphabet, 3) Lines starting witha lowercase letter will appear before lines starting with the same letter in UPPERCASE |
| *sort -r file\_name* | It sort but in revrse (*DESC*) (by default it is ASC) |
| *sort -u file\_name* | It give sorted but unique |
| *sort -f file\_name* | It sort data but ignore the case sensitive |
| *sort -n file\_name* | It take String first then numeric |
| 20 | *uniq*  *uniq* | *uniq file\_name* | It give DISTINCT output |
| *uniq -c file\_name* | It give number of occurrences (*It is like COUNT(\*) in SQL*) |
| *uniq -d file\_name* | Only print duplicate lines, one for each group |
| *uniq -u file\_name* | Only print UNIQUE lines |
| *uniq -i file\_name* | ignore differences in case when comparing (It become CASE INSENSTIVE) |
| 21 | *sed* |  | Sed is a stream editor that you can use as a filter. It reads each line of input and then performs a set of requested actions |
| *sed -n 'xp' file\_name* | It is for print the purticular line from file. We have give number of line at *x* |
| *sed '/^a/d' file\_name* | It will delete all lines which start with *a* |
| *sed -i '/^a/d' file\_name* | by adding *-i* it will delete permantly. If don.t use *-i* then changes are not made in file they just for view |
| *sed 's/P/p/' file\_name* | It will replace all '*P*' with '*p*'. It change only 1 in line not all |
| *sed -i 's/P/p/' file\_name* | It will replace all '*P*' with '*p*'. It change only 1 in line not all. HERE we add *-i* that's why changes are permanant |
| *sed -i 's/P/p/g' file\_name* | It will replace all '*P*' with '*p*'. It change not only 1 but all in line (because we add '*g*'). HERE we add *-i* that's why changes are permanant |
| *sed -e 'command\_1' -e 'command\_2'* | by using *-e* we can run multiple *-sed commands* |
| *sed '1d;$d' file\_name* | It will delete 1st and Last line. |
| 22 | *du* | *du -sh file\_name* | It will show space (storage) of our file (*-sh* is for human readable format) |
| *du -sh file1 file2* | by adding multiple file name we can see multiple files space |
| 23 | *df* | *df -h* | It display info related to file system about total space and available space |
| 24 | *ps* | *ps* | we get all info about background running task (it is like task manager in windows) |
| 25 | *sleep* | *sleep 5m  $ sleep 5m &* | It will sleep of *5m* because we use 5 before m if want for 3 minutes then we can use sleep *3m*. For that time we can'n do anything  and & because it runs on background |
| 26 | *kill*  *kill* | *kill <PID>* | We can terminate any process for that we have know *PID* of that process by using *$ ps* we get *PID* of that process (it is like END TASK in windows) |
| *kill -9 <PID>* | It kill forcefully |
| 27 | *cut* |  | It give perticular part of line |
| *cut -d '|' -f2,3 file\_name* | It give second and third part of line which seperated by deliminator '|' (deliminator may changes sometime is ',' , ':' ) |
| *cut -c 5-10 file\_name* | It give 5 to 10 character of each line |
| 28 | *find* | *find /path/to/search* | Searches for files and directories within the specified path. |
| *find .* | Searches in the current directory and its subdirectories. |
| *find / -name "file.txt"* | Searches for a file named "file.txt" in the entire filesystem. |
| *find . -iname "file.txt"* | Case-insensitive search for "file.txt" in the current directory. |
| *find . -type f* | Finds all files in the current directory and subdirectories. |
| *find . -type d* | Finds all directories in the current directory and subdirectories. |
| *find . -size +10M* | Finds files larger than 10MB. |
| *find . -size -500k* | Finds files smaller than 500KB. |
| *find . -mtime -7* | Finds files modified within the last 7 days. |
| *find . -atime +30* | Finds files last accessed more than 30 days ago. |
| *find . -user username* | Finds files owned by the specified user. |
| *find . -group groupname* | Finds files belonging to the specified group. |
| *find . -perm 777* | Find files which have all permissions |
| *find . -perm 644* | Finds files with specific permissions (e.g., 644). |
| *find . -empty* | Finds empty files and directories. |
| *find . -exec rm {} \;* | Finds and deletes files (use with caution). |
| *find . -exec ls -l {} \;* | Finds and lists files with detailed information. |
| *find . -print* | Prints the paths of found files and directories. |
| *find . -maxdepth 2* | Searches only up to 2 levels deep in directories. |
| *find . -mindepth 3* | Skips searching the first 2 levels of directories. |
| *find . -name "\*.log" -o -name "\*.txt"* | Finds files with .log or .txt extensions. |
| *find . -type f -name "\*.sh" -exec chmod +x {} \;* | Finds and makes .sh files executable. |
| 29 | *awk*  *awk* | *awk '{print $0}' file.txt* | Prints all lines of the file (default behavior). |
| *awk '{print $1}' file.txt* | Prints the first column of each line. |
| *awk '{print $1, $3}' file.txt* | Prints the first and third columns of each line. |
| *awk -F '|' 'NR==3{print $5}' file\_name* | Prints the 5th column of 3rd Line. (Column seperated by '|' that's why we use '|') |
| *awk 'NR==3' file.txt* | Prints only the third line of the file. |
| *awk 'NR>2 && NR<6' file.txt* | Prints lines from the 3rd to the 5th. |
| *awk 'END{print NR}' file.txt* | Prints the total number of lines in the file. |
| *awk '/pattern/' file.txt* | Prints lines containing "pattern". |
| *awk '$2 > 50' file.txt* | Prints lines where the second column value is greater than 50. |
| *awk 'BEGIN{print "Header"} {print} END{print "Footer"}' file.txt* | Adds a header and footer while printing file content. |
| *awk -F ':' '{print $1}' /etc/passwd* | Uses : as a field separator and prints the first field. |
| *awk '{sum+=$2} END {print sum}' file.txt* | Sums up the values in the second column. |
| *awk '{if ($3 > 60) print $1, "Pass"; else print $1, "Fail"}' file.txt* | Adds "Pass" or "Fail" based on the third column's value. |
| *awk '{count[$2]++} END {for (i in count) print i, count[i]}' file.txt* | Counts occurrences of unique values in the second column. |
| *awk 'length($0) > 50' file.txt* | Prints lines that have more than 50 characters. |
| *awk '{print toupper($1)}' file.txt* | Converts the first column to uppercase. |
| *awk '{print tolower($1)}' file.txt* | Converts the first column to lowercase. |
| *awk '{sub("old", "new"); print}' file.txt* | Replaces the first occurrence of "old" with "new" in each line. |
| *awk '{gsub("old", "new"); print}' file.txt* | Replaces all occurrences of "old" with "new" in each line. |

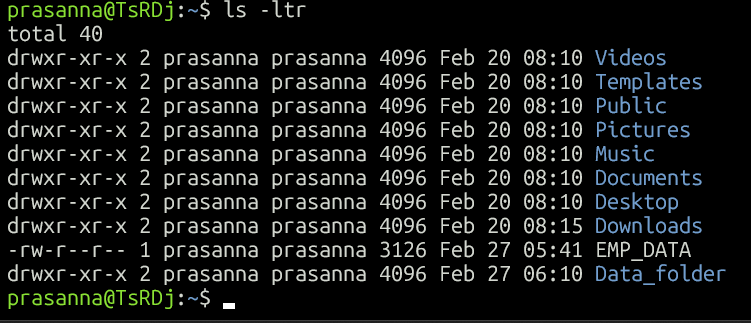
# Manipulating File Attributes

* UNIX support different types of files, to determine a file’s type, specify the *-l* option to the *ls*.

Example

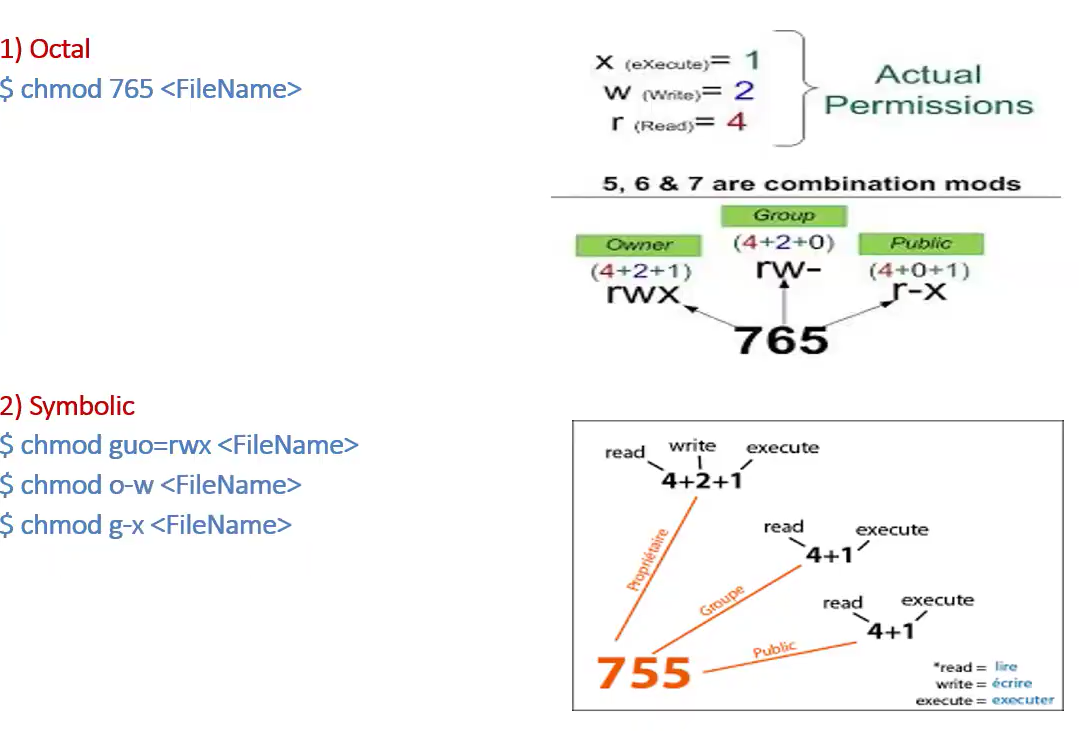
$ls -l <File\_Name>

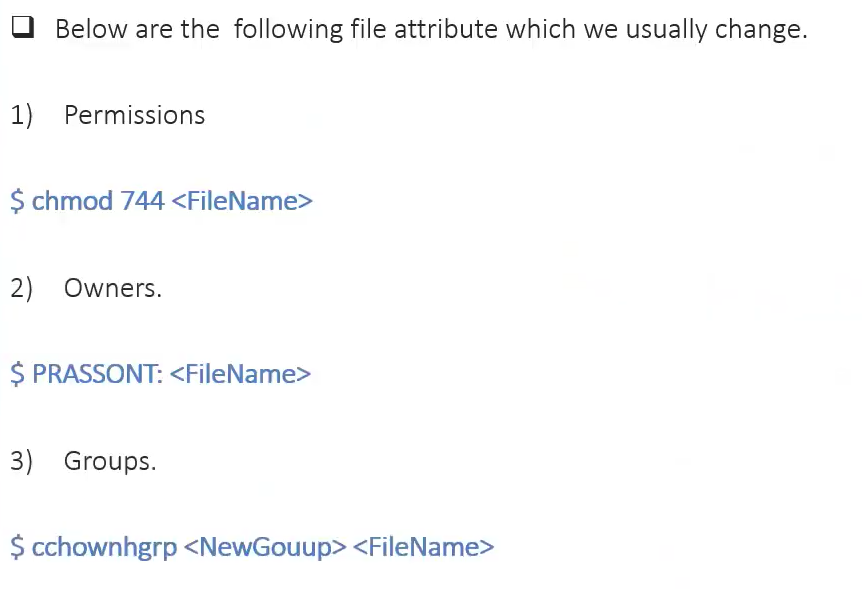
🢥First character represents the type of a file. Below are the variable options

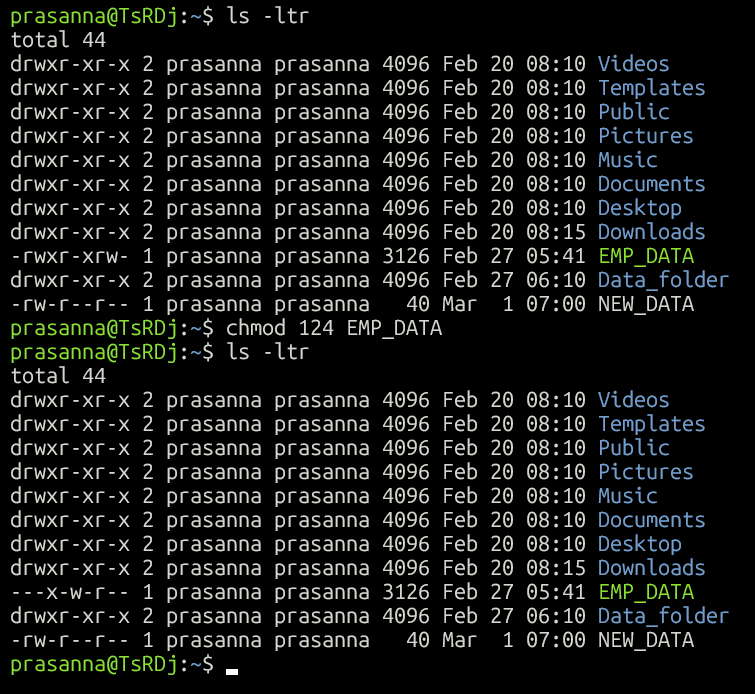


## Permission:

* You can change the permission of the file using *chmod* command, using symbolic and octal.







We can see here the permission of file before using *chmod* command

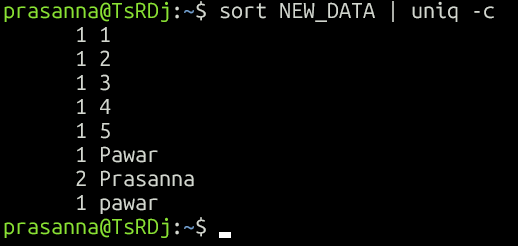
We can see here the permission of file after using *chmod* command

# USE of ‘|’

We can use COMMAND within a COMMAND like SUBQUERY by using |

Example: Here we use sort and uniq command. Not only these we can use any command like that using |

In following example count of each line



If we want only files not directories because when we do *$ls -ltr* it will show all files and directories then we can use,

*$ ls -ltr | grep ^-*

And if we want only directories

*$ls -ltr | grep ^d*

If we want last 1 file only. Then,

*$ ls -ltr | grep ^- | tail -1*

These are some examples. You can try many more like this!

# Sed (Stream editor)

* Sed is a stream editor that you can use as a filter. It reads each line of input and then performs a set of requested actions

*$ sed ‘script’ files*

|  |  |
| --- | --- |
| *sed -n 'xp' file\_name* | It is for print the particular line from file. We have given number of line at *x* |
| *sed '/^a/d' file\_name* | It will delete all lines which start with *a* |
| *sed -i '/^a/d' file\_name* | by adding *-i* it will delete permantly. If don.t use *-i* then changes are not made in file they just for view |
| *sed 's/P/p/' file\_name* | It will replace all '*P*' with '*p*'. It changes only 1 in line not all |
| *sed -i 's/P/p/' file\_name* | It will replace all '*P*' with '*p*'. It change only 1 in line not all. HERE we add *-i* that's why changes are permanent |
| *sed -i 's/P/p/g' file\_name* | It will replace all '*P*' with '*p*'. It change not only 1 but all in line (because we add '*g*'). HERE we add *-i* that's why changes are permanant |
| *sed -e 'command\_1' -e 'command\_2'* | by using *-e* we can run multiple *-sed commands* |
| *sed '1d;$d' file\_name* | It will delete 1st and Last line. |

# Awk

|  |  |
| --- | --- |
| *awk '{print $0}' file.txt* | Prints all lines of the file (default behavior). |
| *awk '{print $1}' file.txt* | Prints the first column of each line. |
| *awk '{print $1, $3}' file.txt* | Prints the first and third columns of each line. |
| *awk -F '|' 'NR==3{print $5}' file\_name* | Prints the 5th column of 3rd Line. (Column seperated by '|' that's why we use '|') |
| *awk 'NR==3' file.txt* | Prints only the third line of the file. |
| *awk 'NR>2 && NR<6' file.txt* | Prints lines from the 3rd to the 5th. |
| *awk 'END{print NR}' file.txt* | Prints the total number of lines in the file. |
| *awk '/pattern/' file.txt* | Prints lines containing "pattern". |
| *awk '$2 > 50' file.txt* | Prints lines where the second column value is greater than 50. |
| *awk 'BEGIN{print "Header"} {print} END{print "Footer"}' file.txt* | Adds a header and footer while printing file content. |
| *awk -F ':' '{print $1}' /etc/passwd* | Uses : as a field separator and prints the first field. |
| *awk '{sum+=$2} END {print sum}' file.txt* | Sums up the values in the second column. |
| *awk '{if ($3 > 60) print $1, "Pass"; else print $1, "Fail"}' file.txt* | Adds "Pass" or "Fail" based on the third column's value. |
| *awk '{count[$2]++} END {for (i in count) print i, count[i]}' file.txt* | Counts occurrences of unique values in the second column. |
| *awk 'length($0) > 50' file.txt* | Prints lines that have more than 50 characters. |
| *awk '{print toupper($1)}' file.txt* | Converts the first column to uppercase. |
| *awk '{print tolower($1)}' file.txt* | Converts the first column to lowercase. |
| *awk '{sub("old", "new"); print}' file.txt* | Replaces the first occurrence of "old" with "new" in each line. |
| *awk '{gsub("old", "new"); print}' file.txt* | Replaces all occurrences of "old" with "new" in each line. |

$ awk -F '|' 'NR==5{print $5}' file\_name

THIS COMMAND GIVE Prints the 5th column of 5th Line. (Column separated by '|' that's why we use '|')

But we need same output without using awk,

$ sed -n '5p' file\_name | cut -d ‘|’ -f5

Without sed

$ head -5 file\_name | tail -1 | cut -d ‘|’ -f

# SHELL SCRIPT

* Scripts are collections of commands that are stored in a file. The shell can read this file and act on the commands as if they were typed at the keyboard.
* One of the most important tasks in writing shell scripts is making the shell script executable and making sure that the correct shell is invoked on the script.
* If you wanted to run the script by typing its name, you need to do two things:

1. Make it executable.
2. Make sure that the right shell is used when the script is run.

* To ensure that the correct shell is used to run the script, you must add the following "magic" line to the beginning of the script: #!/bin/sh
* Without a magic line, users might not be able to get a Bourne shell (*sh*) script to run correctly.
* The #!/bin/sh must be the first line of a shell script in order for *sh* to be used to run the script. If this appears on any other line, it is treated as a comment and ignored by all shells.

# TASK\_1.ksh

## Objective:

Make a script in which State that Number of actual records should match Number record given in trailer(footer) of data file.

If get match then state “Validation Successful” if does not match state “Validating Unsuccessful”

Demo *data\_file.txt*

ID|Name|City|Salary

1|User\_1|Mumbai|53789.92

2|User\_2|Delhi|19834.47

3|User\_3|Bangalore|75022.25

4|User\_4|Hyderabad|41506.68

5|User\_5|Chennai|62971.14

6|User\_6|Pune|31287.79

7|User\_7|Kolkata|89015.55

8|User\_8|Ahmedabad|24653.32

9|User\_9|Jaipur|75984.44

10|User\_10|Surat|38476.61

data\_file\_.txt|10|2025-03-06 14:30:15

#!/bin/sh

File=$1

var1=$(tail -1 "$File" | cut -d '|' -f2)

var2=$(sed '1d;$d' "$File" | wc -l)

if [ "$var1" -eq "$var2" ];

then

echo "Validation Successful"

else

echo "Validation Failed"

fi

* **#!/bin/sh**

This is the **shebang** line, specifying that the script should be executed using the Bourne shell (sh).

* **File=$1**

This means that when you run script the variable File will hold "Parameter\_1"

* **var1=$(tail -1 "$File" | cut -d '|' -f2)**

here we define Variable\_1 for our script in which we write command ***tail -1 "$File"*** Extracts the last line of the file ***cut -d '|' -f2***

Splits the last line using | as the delimiter and extracts the **2nd field**. And the extracted value is stored in the variable ***var1.***

* **var2=$(sed '1d;$d' "$File" | wc -l)**

***sed '1d;$d' "$File"*** *1d* → Deletes the first line of the file. *$d* → Deletes the last line of the file. This removes both the header and footer, leaving only the actual data rows. ***wc -l*** Counts the number of remaining lines. The result is stored in ***var2.***

* **if [ "$var1" -eq "$var2" ];**

Checks whether ***var1*** **equals** ***var2*** (count of actual data rows). -***eq*** is used for numeric comparison.

* **then echo "Validation Successful"**

If the values match, it means the last line correctly reflects the number of data rows, so the script prints: **Validation Successful**

* **Else echo "Validation Failed" fi**

If the values do not match, it prints: **Validation Failed.** The script ends with **fi**, which is the closing statement for an if block in shell scripting.

# TASK\_2.ksh

## Objective

We have to connect UNIX to DATABASE and bring our required output in our SCRIPT output

Here we have need count of records in particular table

#!/bin/bash

DB\_USER="hr"

DB\_PASSWORD="hr"

DB\_SID="xe"

TABLE\_NAME=$1

COUNT=$(sqlplus -s "$DB\_USER/$DB\_PASSWORD@$DB\_SID" <<EOF

SET HEAD OFF;

SET FEEDBACK OFF;

SELECT COUNT(\*) FROM $TABLE\_NAME;

EXIT;

EOF

)

echo "Record Count in Table $TABLE\_NAME: $COUNT"

* **Oracle Database Credentials:** here we define DB credentials

DB\_USER="hr"

DB\_PASSWORD="hr"

DB\_SID="xe"

* **Table Name:** here we state our table name as Parameter 1

TABLE\_NAME=$1

* **Execute SQL Query to Get Count**

COUNT=$(sqlplus -s "$DB\_USER/$DB\_PASSWORD@$DB\_SID" <<EOF

SET HEAD OFF;

SET FEEDBACK OFF;

SELECT COUNT(\*) FROM $TABLE\_NAME;

EXIT;

EOF

)

***COUNT=$(sqlplus -s "$DB\_USER/$DB\_PASSWORD@$DB\_SID" <<EOF***

* The $( ... ) syntax captures the **output of the SQL query** and stores it in the variable COUNT.
* sqlplus -s connects to an **Oracle Database** using sqlplus in **silent mode (-s)** to suppress login messages.
* "${DB\_USER}/${DB\_PASSWORD}@${DB\_SID}":
  + $DB\_USER → Oracle database username (e.g., hr).
  + $DB\_PASSWORD → Password for the user.
  + $DB\_SID → Oracle Database **System Identifier** (e.g., xe).
* The <<EOF ... EOF syntax (called **Here Document**) is used to pass multiple SQL commands to sqlplus.

***SET HEAD OFF;***

* Disables column headers in the output.

***SET FEEDBACK OFF;***

* Suppresses extra messages like 10 rows selected.

***SELECT COUNT(\*) FROM $TABLE\_NAME;***

* This is our SQL query which will hit the database

***EXIT;***

* Ensures that sqlplus exits after running the query.

***echo "Record Count in Table $TABLE\_NAME: $COUNT"***

* This will print output from DB

# TASK\_3.ksh

## Objective:

**Part 1:** In which State that Number of actual records should match Number record given in trailer(footer) of data file. (like TASK\_1.ksh)

**Part 2:** In which we have to check every record or line must have same number of field (columns)

**Part 3:** In which we have to check whether the duplicate records are present or not

#!/bin/sh

File=$1

var1=$(tail -1 "$File" | cut -d '|' -f3)

var2=$(sed '1d;$d' "$File" | uniq | wc -l)

if [ "$var1" -eq "$var2" ];

then

echo "Validation Successful"

else

echo "Validation Failed"

fi

var3=$(awk -F'|' '{print NF}' "$File" | sort -u | wc -l)

if [ "$var3" -eq 1 ];

then

echo "Validation Successful : Records have the same number of fields."

else

echo "Validation Failed : Number of fields are not same"

  exit 1

fi

var4=$(sort "$File" | uniq -id)

if [ -z "$var4" ];

then

echo "Validation Successful : File doesn't have duplicate records"

else

echo "Validation Failed: Duplicate Record Found"

fi

We have to make all parts combine script

Part 3

Part 2

Part 1

Part 1 Explanation in TASK\_1.ksh

Part 2 Explanation:

var3=$(awk -F'|' '{print NF}' "$File" | sort -u | wc -l)

if [ "$var3" -eq 1 ];

then

echo "Validation Successful : Records have the same number of fields."

else

echo "Validation Failed : Number of fields are not same"

  exit 1

fi

* **var3=$(awk -F'|' '{print NF}' "$File" | sort -u | wc -l) : Here we assign Variable 3**
* ***awk -F'|' '{print NF}' "$File"***

awk is used for text processing. ***-F'|'*** sets the field separator to |, meaning awk will treat | as the column delimiter. ***{print NF}*** prints the number of fields (columns) in each line of the file. ***"$File"*** refers to a shell variable containing the filename.

* ***sort -u***

***sort*** sorts the output. ***-u*** (unique) removes duplicate values, so we get only distinct field counts.

* ***wc -l***

***wc -l*** counts the number of unique field counts. If all lines have the same number of fields, there will be only one unique count.

If different lines have different field counts, there will be multiple values. The result is stored in var3.

* **if [ "$var3" -eq 1 ];**
* Checks if the number of unique field counts is exactly 1, meaning all lines have the same number of fields.
* **then echo "Validation Successful : Records have the same number of fields."**
* If all records have the same number of fields (var3 == 1), it prints a success message.

**else**

**echo "Validation Failed : Number of fields are not same"**

**exit 1**

**fi**

* If var3 is greater than 1 (meaning different lines have different numbers of fields): Prints a failure message. ***exit 1*** stops the script with a non-zero exit code (indicating an error).
* **var4=$(sort "$File" | uniq -id)**
* sort "$File" Sorts the file ($File) in ascending order, which helps uniq identify duplicate lines that are adjacent.
* **uniq -id**
* ***uniq*** filters out unique lines and displays only duplicate lines. ***-i*** makes the comparison case-insensitive. ***-d*** prints only duplicate lines (not unique ones). The output (if there are duplicate lines) is stored in ***var4***.
* **if [ -z "$var4" ];**
* -***z "$var4"*** checks if var4 is empty. If var4 is empty, it means no duplicate records were found.
* **Then echo "Validation Successful : File doesn't have duplicate records"**
* If var4 is empty, the script prints a success message indicating that there are no duplicate records.
* **else**

**echo "Validation Failed: Duplicate Record Found"**

**fi**

* If var4 is **not empty** (i.e., duplicates exist), the script prints a failure message.

# TASK\_4.KSH

## Objective:

In this script we have to **validate and retrieve metadata information** for a specific file from an Oracle database table named **FILE\_LOAD\_METADATA**. It ensures that:

1. A valid FILE\_ID is provided as an parameter (If only and only 1 parameter allowed not less than 1 or greater than 1.
2. The FILE\_ID exists in the database (optional).
3. Metadata (delimiter, number of columns, and target column/connection) is extracted and stored in a CSV file.

#!/bin/sh

if [ $# -ne 1 ]; then

echo "Validation failed: Script requires exactly one parameter."

exit 1

fi

echo "Validation successful"

DB\_USER="hr"

DB\_PASSWORD="hr"

DB\_SID="xe"

OUTPUT\_FILE="$1.csv"

FILE\_EXISTS=$(sqlplus -s "$DB\_USER/$DB\_PASSWORD@$DB\_SID" <<EOF

SET HEAD OFF;

SET FEEDBACK OFF;

SET PAGESIZE 0;

SET LINESIZE 500;

SET TRIMSPOOL ON;

SELECT COUNT(\*) FROM FILE\_LOAD\_METADATA WHERE FILE\_ID = '$1';

EXIT;

EOF

)

if [ "$(echo "$FILE\_EXISTS")" -eq 0 ]; then

echo "Error: Invalid parameter. FILE\_ID '$1' not found in FILE\_LOAD\_METADATA."

exit 1

fi

var1=$(sqlplus -s "$DB\_USER/$DB\_PASSWORD@$DB\_SID" <<EOF

SET HEAD OFF;

SET FEEDBACK OFF;

SET PAGESIZE 0;

SET LINESIZE 500;

SET TRIMSPOOL ON;

SELECT DELIMINATOR, NO\_OF\_COLUMN, TARGET\_COLUMN

FROM FILE\_LOAD\_METADATA WHERE FILE\_ID = '$1';

EXIT;

EOF

)

DELIMITER=$(echo "$var1" | awk '{print $1}')

COLUMN\_COUNT=$(echo "$var1" | awk '{print $2}')

FILE\_CONNECTION=$(echo "$var1" | awk '{print $3}')

echo "Deliminator = $DELIMITER" > "$OUTPUT\_FILE"

echo "No\_of\_Column = $COLUMN\_COUNT" >> "$OUTPUT\_FILE"

echo "Target\_Connection = $FILE\_CONNECTION" >> "$OUTPUT\_FILE"

echo "Output stored in $OUTPUT\_FILE"

PART\_3

Here we save our file

Here we retrieve required field from Database

PART\_2

**Here we assign values of DB credential and also output\_file**

**PART\_1**

* **if [ $# -ne 1 ]; then**

**echo "Validation failed: Script requires exactly one parameter."**

**exit 1**

**fi**

* ***$#*** represents the number of arguments passed to the script. ***if [ $# -ne 1 ];*** Checks if the number of arguments is **not equal to 1** (***-ne*** means "not equal"). If true, it prints an error message and **exits with status 1** (indicating failure).
* **echo "Validation successful"**
* If exactly **one argument** is passed, the script prints "Validation successful" and continues.
* **DB\_USER="hr"**

**DB\_PASSWORD="hr"**

**DB\_SID="xe"**

**OUTPUT\_FILE="$1.csv"**

* Defines **Oracle database credentials**:
  + DB\_USER="hr" → Username is hr.
  + DB\_PASSWORD="hr" → Password is hr.
  + DB\_SID="xe" → The Oracle Database SID (System Identifier) is xe.
  + Defines OUTPUT\_FILE, where $1 (the input argument) represents the FILE\_ID. This means if the script is run with 1234, the output file will be **1234.csv**.
* **FILE\_EXISTS=$(sqlplus -s "$DB\_USER/$DB\_PASSWORD@$DB\_SID" <<EOF**

**SET HEAD OFF;**

**SET FEEDBACK OFF;**

**SET PAGESIZE 0;**

**SET LINESIZE 500;**

**SET TRIMSPOOL ON;**

**SELECT COUNT(\*) FROM FILE\_LOAD\_METADATA WHERE FILE\_ID = '$1';**

**EXIT;**

**EOF**

**)**

* Runs an **SQL query** using sqlplus to check if FILE\_ID exists in the FILE\_LOAD\_METADATA table.
* sqlplus -s → The -s option runs SQL\*Plus in **silent mode** (no extra output).
* <<EOF ... EOF → A **here document** used to pass multiple SQL commands to sqlplus.
* **SQL Commands Explanation**
* **SET HEAD OFF;** → Hides column headers in the output.
* **SET FEEDBACK OFF;** → Suppresses SQL execution messages.
* **SET PAGESIZE 0;** → Removes extra blank lines from output.
* **SET LINESIZE 500;** → Ensures long output lines don’t wrap.
* **SET TRIMSPOOL ON;** → Removes trailing spaces.
* **SELECT COUNT(\*) FROM FILE\_LOAD\_METADATA WHERE FILE\_ID = '$1';**
  + Checks how many times FILE\_ID appears in the database.
* The result is stored in **$FILE\_EXISTS**.
* **if [ "$(echo "$FILE\_EXISTS")" -eq 0 ]; then**

**echo "Error: Invalid parameter. FILE\_ID '$1' not found in FILE\_LOAD\_METADATA."**

**exit 1**

**fi**

* if [ "$(echo "$FILE\_EXISTS")" -eq 0 ];
  + If FILE\_EXISTS is 0, it means FILE\_ID **does not exist**.
  + The script prints an error message and **exits with 1**.
* **var1=$(sqlplus -s "$DB\_USER/$DB\_PASSWORD@$DB\_SID" <<EOF**

**SET HEAD OFF;**

**SET FEEDBACK OFF;**

**SET PAGESIZE 0;**

**SET LINESIZE 500;**

**SET TRIMSPOOL ON;**

**SELECT DELIMINATOR, NO\_OF\_COLUMN, TARGET\_COLUMN**

**FROM FILE\_LOAD\_METADATA WHERE FILE\_ID = '$1';**

**EXIT;**

**EOF**

**)**

* Runs another SQL query to fetch three metadata values for the given FILE\_ID:
  1. DELIMINATOR → The **column separator** (e.g., , or |).
  2. NO\_OF\_COLUMN → The **number of expected columns** in the file.
  3. TARGET\_COLUMN → The **target table or connection information**.
* Stores the output in **var1**.
* **DELIMITER=$(echo "$var1" | awk '{print $1}')**

**COLUMN\_COUNT=$(echo "$var1" | awk '{print $2}')**

**FILE\_CONNECTION=$(echo "$var1" | awk '{print $3}')**

* Uses ***awk*** to extract values from var1:
  + $1 → First value (DELIMINATOR).
  + $2 → Second value (NO\_OF\_COLUMN).
  + $3 → Third value (TARGET\_COLUMN).
* **echo "Deliminator = $DELIMITER" > "$OUTPUT\_FILE"**

**echo "No\_of\_Column = $COLUMN\_COUNT" >> "$OUTPUT\_FILE"**

**echo "Target\_Connection = $FILE\_CONNECTION" >> "$OUTPUT\_FILE"**

* **Creates and writes metadata** into the output file ($1.csv):
  + > **Creates a new file** (overwrites if it exists).
  + >> **Appends** the next lines to the same file.
* **echo "Output stored in $OUTPUT\_FILE"**
* Displays a message indicating where the metadata has been saved.

## **TASK\_4.ksh but file save in particular location by giving path**

* ***OUTPUT\_DIR="H:\10.4\INFORMATICA\server\infa\_shared\BWParam"***
* Defines the **output directory path** where the CSV file will be stored.
* ***OUTPUT\_FILE="${OUTPUT\_DIR}/${1}.csv"***
* The output file is named using the **first script argument ($1)**.
* ***if [ ! -d "$OUTPUT\_DIR" ];***
* ***-d*** → Checks if ***$OUTPUT\_DIR*** is a **directory**.
* ***!*** → Negates the condition, meaning **"if the directory does not exist"**.
* If ***$OUTPUT\_DIR*** **doesn't exist**, the script executes the then block.
* ***echo "Error: Output directory does not exist. Creating it now..."***
* **Prints an error message** indicating that the directory does not exist.
* ***mkdir -p "$OUTPUT\_DIR"***
* ***mkdir*** → Creates a new directory.
* ***-p*** → Ensures **all parent directories** are created if they don’t exist.

#!/bin/sh

if [ $# -ne 1 ]; then

echo "Validation failed: Script requires exactly one parameter."

exit 1

fi

echo "Validation successful"

DB\_USER="hr"

DB\_PASSWORD="hr"

DB\_SID="xe"

OUTPUT\_DIR="H:\10.4\INFORMATICA\server\infa\_shared\BWParam"

OUTPUT\_FILE="${OUTPUT\_DIR}/${1}.csv”

if [ ! -d "$OUTPUT\_DIR" ]; then

echo "Error: Output directory does not exist. Creating it now..."

mkdir -p "$OUTPUT\_DIR"

fi

var1=$(sqlplus -s "$DB\_USER/$DB\_PASSWORD@$DB\_SID" <<EOF

SET HEAD OFF;

SET FEEDBACK OFF;

SET PAGESIZE 0;

SET LINESIZE 500;

SET TRIMSPOOL ON;

SELECT DELIMINATOR, NO\_OF\_COLUMN, TARGET\_COLUMN

FROM FILE\_LOAD\_METADATA WHERE FILE\_ID = '$1';

EXIT;

EOF

)

DELIMITER=$(echo "$var1" | awk '{print $1}')

COLUMN\_COUNT=$(echo "$var1" | awk '{print $2}')

FILE\_CONNECTION=$(echo "$var1" | awk '{print $3}')

echo "Deliminator,$DELIMITER" > "$OUTPUT\_FILE"

echo "No\_of\_Column,$COLUMN\_COUNT" >> "$OUTPUT\_FILE"

echo "Target\_Connection,$FILE\_CONNECTION" >> "$OUTPUT\_FILE"

# Print confirmation

echo "Output stored in: $OUTPUT\_FILE"

Rest Script same as above