**SHELL SCRIPTING**

# Shell variables vs Environment variables

**Shell variables are variables whose scope is in the current shell session, for example in an interactive shell session or a script.**

**var="hello"**

**BASE=/home/oracle**

**The use of shell variables is to keep track of data in the current session. Shell variables usually have names with lower-case letters.**

**Environment variables**

**An environment variable is a shell variable which has been exported. This means that it will be visible as a variable, not only in the shell session that created it, but also for any process (not just shells) that are started from that session. We us the key word (export)**

**VAR="hello" # shell variable created**

**export VAR # variable now part of the environment**

**or**

**export VAR="hello"**

**export BASE=/home/oracle**

**or**

**BASE=/home/oracle; export BASE**

**or**

**BASE=/home/oracle**

**export BASE**

**Once a shell variable has been exported, it stays exported until it is unset, or until its "export property" is removed (with export -n in bash), so there's usually no need to re-export it. Unsetting a variable with unset deletes it (no matter if it's an environment variable or not).**

# Variable Assignment

**=**

**the assignment operator (no space before and after)**

**# When is a variable "naked", i.e., lacking the '$' in front?**

**# Assignment**

**a=879**

**# Assignment using 'let'**

**The let assignment is used as Arithmetic variables and assignments, The original syntax is**

**(( intvar=expression )) , however the syntax isn't intuitive so we use a let built-in command instead**

**let intvar=expression**

**Note because of the special characters like \*+/- , it is a good practice to put the expression in quotes**

**let a=** **'16+5'**

**echo "the value of a is $a"**

**# In a 'for' loop (really, a type of disguised assignment):**

**for a in 7 8 9 11**

**do**

**echo "$a "**

**done**

**# In a 'read' statement (also a type of assignment):**

**echo "enter the value of a"**

**read a**

**echo "the value you supplied is $a"**

**# Variable Assignment, plain and fancy**

**#!/bin/bash**

**a=23 # Simple case**

**echo $a**

**b=$a**

**echo $b**

**# Variable Assignment (command substitution).**

**a=`date +" %F"`**

**echo "the current date is $a"**

**#(Note if you want to obtain the current date , `date` is the way since it will be executed and get the current date anything it is executed**

**Also note `some value` is obsolete, as it can cause errors, use $(some value) instead**

**a=$(date +" %F")**

**echo "the current date is $a"**

**Special shell variables**

**There are some variables which are set internally by the shell and which are available to the user:**

|  |  |
| --- | --- |
| Name | Description |
| $1 - $9 | **these variables are the positional parameters.** |
| $0 | **the name of the command currently being executed.** |
| $# | **the number of positional arguments given to this invocation of the shell** |
| $? | **the exit status of the last command executed**  **When a command completes successfully, it returns the exit status of 0(zero),**  **otherwise it returns a non-zero exit status.** |
| $! | **the process id of the last command run in the background** |
| $$ | **The process id of the current command being run** |
| $- | **the current options supplied to this invocation of the shell** |
| $\* | **a string containing all the arguments to the shell, starting at $1.** |
| $@ | **same as $\*** |

**See this example for better understanding**

**vi special\_vars.sh and copy and paste the following in red**

**#!/bin/sh**

**echo "File Name: $0"**

**echo "First Parameter : $1"**

**echo "Second Parameter : $2"**

**echo "Quoted Values: $@"**

**echo "Quoted Values: $\*"**

**echo "Total Number of Paramers : $#"**

**echo "Process ID of current command is : $$"**

**echo "The Exit Status is : $?"**

**chmod 755 special\_vars.sh**

**Here is a sample run for the above script:**

**./special\_vars.sh is cool**

*File Name: ./special\_vars.sh*

*First Parameter : is*

*second Parameter : cool*

*Quoted Values: is cool*

*Quoted Values: is cool*

*Total Number of Paramers : 2*

*Process ID of current command is: 12728*

*The Exit Status is : 0*

# File manipulation and pattern matching

**cut - remove sections from each line of files**

**-b --bytes (select only these bytes)**

**echo Teacher|cut -b 4**

**c**

**echo Teacher|cut -b 7**

**r**

**echo Teacher|cut -b 3,7**

**ar**

**echo Teacher|cut -b 3-7**

**acher**

**echo Teacher|cut -b5-**

**her**

**-c --characters (select only these characters)**

**echo Teacher|cut -c 4 ##Space is not an issue 'echo Teacher|cut -c4' will give you the same result**

**c**

**echo Teacher|cut -c 7**

**r**

**echo Teacher|cut -c 3,7**

**ar**

**echo Teacher|cut -c 3-7 OR echo Teacher|cut -c3-7**

**acher**

**echo Teacher|cut -c5-**

**her**

**Note: characaters(-c) and bytes(-c) may seems to be the same but**

**some characters are more than just like bytes, so that case your output will be wrong**

**To be safe when you use -b, use c instead of b**

**-d --delimiter (use DELIM instead of TAB for field delimiter)**

**The -d is usually used in conjunction with the -f option (field) to specify the field that should be cut**

**echo "PK, Johnson, 40, MD" |cut -d ',' -f 1**

**PK**

**echo will yield PK, Johnson, 40, MD**

**the cut -d means base your cutting on the comma (,) delimiter**

**-f 1 means look/use field one of the comma delimiter**

**Watch this**

**echo "PK, Johnson, 40, MD" |cut -d ',' -f1 ##Note: the field count begins just before the first delimiter.**

**40**

**echo "PK, Johnson, 40, MD" |cut -d ',' -f4 ##Note: the field count begins just before the first delimiter.**

**MD**

**Q. if you want 40 what will be the command for both examples above**

**Using a file**

**vi test.txt**

**/c01:/c02:/c03:/c04**

**/p01:/p02:/p03:/p04**

**/a01:/a02:/a03:/a04**

**cut -d ':' -f4 test.txt**

**/c04**

**/p04**

**/a04**

**vi test2.txt**

**/c01/app/oracle/product/12.1.0.2**

**/c01/app/oracle/product/12.2.0.2**

**/c01/app/oracle/product/11.2.0.4**

**/c01/app/oracle/product/11.1.0.7**

**cut -d '/' -f 6 test2.txt ##Note: the field count begins just before the first delimiter.**

**12.1.0.2**

**12.2.0.2**

**11.2.0.4**

**11.1.0.7**

**cut -d '/' -f4,6 test2.txt**

**oracle/12.1.0.2**

**oracle/12.2.0.2**

**oracle/11.2.0.4**

**oracle/11.1.0.7**

**Q. what will be the command if you want to get something like oracle/product/12.1.0.2**

# Operators and their Meanings

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| **$**{***variable*#*pattern***} | If the pattern matches the **beginning** of the variable's value, delete the shortest part that matches and return the rest. |
| **$**{**variable##*pattern***} | If the pattern matches the **beginning** of the variable's value, delete the longest part that matches and return the rest. |
| **$**{**variable%*pattern***} | If the pattern matches the **end** of the variable's value, delete the shortest part that matches and return the rest. |
| **$**{***variable*%%*pattern***} | If the pattern matches the **end** of the variable's value, delete the longest part that matches and return the rest. |

Note: **#** matches the front because number signs *precede* numbers;

**%** matches the rear because percent signs *follow* numbers.

**% and %% patterns**

**fn=teacher.xml.picked**

**echo ${fn%.\*}**

**teacher.xml**

**echo ${fn%%.\*}**

**teacher**

**Example**

**Change teacher.xml.picked to teacher\_final**

**fn=teacher.xml.picked**

**echo ${fn%%.\*}\_final**

**teacher\_final**

**Q. What will command will you use to produce teacher.xml\_org**

**echo ${fn%.\*}\_org**

**# and ## patterns**

**fn=/u01/app/oracle/12.1.0.2/DB1.env**

**echo ${fn#/\*/} #(remove the first string within /<string>/)**

**app/oracle/12.1.0.2/DB1.env #(/u01/ is removed)**

**echo ${fn##/\*/} #Remove all the strings within /<strings>/ except the last one**

**DB1.env # Everything is removed except the last string**

**Example : change /u01/oracle/12.1.0.2/DB1.env to /s06/oracle/12.1.0.2/DB1.env**

**fn=****/u01/app/oracle/12.1.0.2/DB1.env**

**echo /s06/${fn#/\*/}**

**/s06/app/oracle/12.1.0.2/DB1.env**

**Q. Write the command to change /u01/app/oracle/12.1.0.2/DB1.env to oracleDB1.env**

# Regular Expressions (Regex)

|  |  |
| --- | --- |
| Regex | Meaning |
| ^ | **Matches Beginning of the line** |
| $ | **End of the line** |
| \* | **One or more matches of the proceeding element** |
| . (dot) | **Matches any character** |
| ? | **zero or one match of the proceeding element** |
| [char] | **Matches any character from a given set** |
| [^char] | **Matches any character not in a given set** |
| [n] | **Matches exactly n instance of the proceeding element** |

# Bourne(sh), Bourne Again Shell(bash) vs Korn Shell(ksh)

**Any shell can be used for writing a shell script. We begin a shell script with the path where the shell executables are be found:**

**#!/path/to/shell (e.g. #!/bin/sh) -Bourne shell**

**#!/path/to/shell (e.g. #!/bin/bash) -Bourne again shell – a POSIX compliant (a superset of sh)**

**#!/path/to/shell (e.g. #!/bin/ksh) -Korn shell (a superset of sh)**

**The #! characters tell the system to locate the following pathname, start it up and feed it the rest of the file as input**

**-Bourne(sh) shell is portable, once you write a shell script in bourne (sh), you can easily use it in any shell, bash/Korn shell on the other hand is very powerful with new features but may not work with bourne. For maximum portability, always us sh**

**POSIX**

**The Portable Operating System Interface is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems**

# Shell Basic Operations

**< Less than**

**> Greater than**

**<= Less than or equal**

**>= Greater than or equal**

**== Equal**

**!= Not equal**

**&& Logical and**

**|| Logical or**

# Shell Decision Making (if..then..elif...else,case)

In shell scripts you usually use tests in **a if or while** statements.

A **test** command or [ expr ] is used to see if an expression is true, and if it is true it return zero(0), otherwise returns nonzero for false.

An expression [expr ] is a combination of values, relational operator (such as >,<, <> etc) and mathematical operators (such as +, -, / etc )."

examples

**5 > 2**

**3 \* 65**

**c > 5 + 30 -1**

**Test Comparison**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Shell Maths operators | Meaning | Mathematical example | Shell Script test example | Shell script expression example |
| -lt | Less than | 4 < 5 | if test 4 -lt 5 | if [ 4 -lt 5 ] |
| -le | **Less than or equal** | **4 <=5** | **if test 4 -lt 5** | **if [ 4 -lt 5 ]** |
| -eq | **Equal** | **4 ==4** | **if test 4 -eq 4** | if [ 4 -eq 4 ] |
| -ge | **Greater than or equal** | **5 >=2** | **if test 5 -ge 2** | **if [ 5 -ge 2 ]** |
| -gt | **Greater than** | **5 > 2** | **if test 5 -gt 2** | **if [ 5 -gt 2 ]** |
| -ne | **Not equal** | **3 != 1** | **if test 3 -ne 1** | **if [ 3 -ne 1 ]** |

|  |  |  |
| --- | --- | --- |
| Test/Operator | Meaning | Example |
| -s file | **True if FILE exists and has a size greater than zero** | **if [ -s names.sh ]; then**  **echo "The file is not empty"**  **else**  **echo "this file is empty"**  **fi** |
| -e file | **True if FILE exists** |  |
| -f file | **True if FILE exists and is a regular file** |  |
| -h file | **True if FILE exists and is a symbolic link** |  |
| -L file | **True if FILE exists and is a symbolic link** |  |
| -d dir | **True if FILE exists and is a directory** | **if [ -d util ]; then**  **echo "yes"**  **fi** |
| -w file | **True if FILE exists and is writable** |  |
| -r file | **True if FILE exists and is readable** |  |
| -x file | **True if FILE exists and is executable** |  |
|  |  |  |
| ! EXPRESSION | **The EXPRESSION is false** |  |
| -n STRING | **The length of STRING is greater than zero** | **STR="Shut up"**  **if [ -n "$STR" ]; then**  **echo "you get something to say?"**  **fi** |
| -z STRING | **The length of STRING is zero (ie it is empty)** | **STR=" "**  **if [ -z "$STR" ]; then**  **echo "Say something"**  **else**  **echo "Naniama"**  **fi** |
| STRING1 = STRING2 | **STRING1 is equal to STRING2** |  |
| STRING1 != STRING2 | **STRING1 is not equal to STRING2** |  |

# If statement

**if [ <some test> ] ##Begin with if**

**then**

**<commands>**

**fi ##always end with fi**

**OR**

**if [ <some test> ]; then**

**<commands>**

**fi**

## **for example**

**if [ -d util ]; then**

**echo "yes"**

**fi**

# if...else statement

**if [ <some test> ]; then**

**<execute commands if true>**

**else**

**<execute this command if NOT true>**

**fi**

## **for example**

**if [ -s names.sh ]; then**

**echo "The file is not empty"**

**else**

**echo "empty file"**

**fi**

# if..elif..else..fi

**if [ <some test> ]; then**

**<commands>**

**elif [ <some test> ]; then**

**<different commands>**

**else**

**<other commands>**

**fi**

## **for example**

**vi check\_if\_elif.sh**

**echo "How old are you?"**

**read age**

**echo "Do you have a letter from your parents?"**

**read ans**

**if [ $age -ge 18 ]; then**

**echo "You may go to the party."**

**elif [ $ans == 'yes' ]; then**

**echo "You may go to the party but be back before midnight."**

**else**

**echo "You cannot go to the party, go home and study."**

**fi**

#### **Try this**

**example1**

**#!/bin/sh ##Using a Bash shell**

**echo "How old are you?"**

**read age**

**if [[ $age -ge 18 ]]; then ##Needs semi-colon to work**

**echo "You may go to the party."**

**fi**

**example2**

**#!/bin/sh ##Using a bash shell**

**echo "How old are you?"**

**read age**

**if [ $age -ge 18 ]; then ## Needs semi-colon to work**

**echo "You may go to the party."**

**fi**

**example3**

**#!/bin/sh ##Using a bash shell**

**echo "How old are you?"**

**read age**

**if [[ $age -ge 18 ]] then ##No, semi-colon so the script will fail**

**echo "You may go to the party."**

**fi**

**example3**

**#!/bin/ksh ##Using a Korn shell**

**echo "How old are you?"**

**read age**

**if [[ $age -ge 18 ]] then ##No semi-colon and still work because it is a Korn shell**

**echo "You may go to the party."**

**fi**

**Conclusion: It is always better to use [[ ]] even if you are using bash/sh shell**

#### **The CASE statement**

case provides a much cleaner, easier-to-write, and far more readable alternative to the if/then/else construct, particularly when there are a lot of possible values to test for. With case, you list the values you want to identify and act upon, and then provide a block of code for each one.

**case $VAR in**

**pattern1)**

**Statement(s) to be executed if pattern1 matches**

;;

**pattern2)**

**Statement(s) to be executed if pattern2 matches**

**;;**

**pattern3)**

**Statement(s) to be executed if pattern3 matches**

**;;**

**\*)**

**Default condition to be executed**

**;;**

**esac**

example1

**document\_type=xml**

**case ${document\_type} in**

**xml)**

**BOOK\_TYPE="Online Directory"**

**;;**

**text)**

**BOOK\_TYPE="Library Directory"**

**;;**

**pdf)**

**BOOK\_TYPE="Online Class"**

**;;**

**esac**

**echo "${BOOK\_TYPE} is perfect for your training"**

example2

**DTIME=$(date +"%Y%m%d\_%H%M%S")**

**case ${ORACLE\_SID} in**

**dev)**

**ORACLE\_HOME=/d01/app/oracle/product/12.1.0.2**

**DUMPFILE=DEV\_${DTIME}.dmp**

**DIRECTORY=/d01/app/datapump**

**USERID=system/dev123**

**;;**

**qa)**

**ORACLE\_HOME=/q01/app/oracle/product/12.1.0.2**

**DUMPFILE=QA\_${DTIME}.dmp**

**DIRECTORY=/q01/app/datapump**

**USERID=system/qa123**

**;;**

**prd)**

**ORACLE\_HOME=/p01/app/oracle/product/12.1.0.2**

**DUMPFILE=PRD\_${DTIME}.dmp**

**DIRECTORY=/p01/app/datapump**

**USERID=system/prd123**

**;;**

**esac**

**sqlplus -s "/as sysdba"<<-EOF**

**create or replace directory expdp\_dir as '${DIRECTORY}';**

**grant read,write on directory expdp\_dir to system;**

**EOF**

**${ORACLE\_HOME}/bin/expdp userid=${USERID} dumpfile=${DUMPFILE} directory=expdp\_dir full=y**

**## if the export is successful, do something otherwise**

**if (( $?==0 )); then**

**mailx -s "export successful for $ORACLE\_SID on $(hostname)" ishglo@gmail.com**

**else**

**mailx -s "export FAILED for $ORACLE\_SID on $(hostname)" ishglo@gmail.com**

**Question: write a script that will use case instead for this last step?**

**case $? in**

**0)**

**mailx -s "export successful for $ORACLE\_SID on $(hostname)" ishglo@gmail.com**

**;;**

**1)**

**mailx -s "export FAILED for $ORACLE\_SID on $(hostname)" ishglo@gmail.com**

**;;**

**esac**

## **TIP1: echo vs printf**

* Both **echo** and **printf** are built-in commands
* **printf** is more portable than echo
* **echo** always exits with a 0 status, and simply prints arguments followed by an end of line character
* **printf** allows for definition of a formatting string and gives a non-zero exit status code upon failure.
* **printf** can be used to specify the field width to use for item, as well as various formatting choices for numbers. (such as what output base to use, whether to print an exponent, whether to print a sign, and how many digits to print after the decimal point). This is done by supplying the format string, that controls how and where to print the other arguments and has the same syntax as C language (%03d, %e, %+d,...)
* **For printf to work perfectly, you need to add the formatting command, for \n for newline, \t for new tab etc**

# LOOPING

#### **The for loop**

We use **for loop** when we know the number of iterations that needs to be done, note you don’t necessarily have to know the exact count, all you need to know to use a for loop is that, **the iteration has a beginning and an end.**

For example,

* numbers from 1 to 10
* list of names in a file, it could be a 100, 500 and 1212, however it has a last name in the file.

**for variable [in list]**

**do #Begin your**

**statements that can use $variable...**

**done #End the Loop**

**example1**

**for i in 01 02 03 04 05**

**do**

**echo "count $i"**

**done**

**for i in 01 02 03 04 05**

**do**

**printf "count $i\n"**

**done**

**example2**

**for names in Fred John Dave Ben Dan**

**do**

**echo "Good Morning Mr. $names"**

**done**

**for names in Fred John Dave Ben Dan**

**do**

**printf "Good Morning Mr. $names\n"**

**done**

**example3**

**## this list all the files that ends with \*.txt, you may not know the total number of files but you know for sure it is not infinite, that is why a for loop is the best option**

**for i in $(ls \*.txt)**

**do**

**cp $i $i.bkup**

**done**

#### ##**Backup of the datafiles that begin sales and safe the backup in the /home/oracle directory**

#### **cd <datafile location>**

**for dbf in $(ls sales\*.dbf)**

**do**

**cp $dbf /home/oracle/$dbf.bkup**

**done**

**to restore the backup files to /u01/app/oracle**

**for i in $(ls sales\*.bkup)**

**do**

**cp $i /u01/app/oracle/${i%%.bkup}"**

**done**

cp sales01.dbf.bkup /u01/app/oracle/sales01.dbf

cp sales02.dbf.bkup /u01/app/oracle/sales02.dbf

cp sales03.dbf.bkup /u01/app/oracle/sales03.dbf

**example4**

#### **##if the content is in a file already, you first use cat to open the file ,see example**

**for LN in $(cat states.txt)**

**do**

**echo "$LN is a state in America"**

**done**

**example5**

**## Note: for loop splits when it sees any whitespace like space, tab. So, you should use IFS (Internal Field Separator)**

**## IFS=$'\n' means make newlines the only field separator**

**IFS=$'\n'**

**for LN in $(cat states.txt)**

**do**

**echo "$LN is a state in America"**

**done**

**example6**

cat town.txt

Virginia, South Carolina, New York, Maryland, New Jersey, Ohio, Texas,

Montana, California, North Carolina, Alabama

**IFS=$','** **## means $',' is the only field separator**

**for i in $(cat town.txt)**

**do**

**echo $i**

**done**

#### **The While loop**

The while loop comes in handy when the number of iterations are not exactly known. The looping will continue until something stops it, for example, you run out of data, you stop it by using control C.

The while loop will execute a piece of code if the **control expression is true**, and only stops when it is false.

Example1

**while true**

**do**

**echo Hello**

**done**

example2

**COUNTER=0**

**while [ $COUNTER -lt 10 ]; do** ##condition

**echo "The counter is $COUNTER "**

**let COUNTER=COUNTER+1** ##(let COUNTER+=1)

**done**

example3

**k=0**

**while [ $k -lt 10 ]**

**do**

**echo $k**

**k=$(( k + 1 ))**

**done**

**example4**

**i=1**

**while [ $i -le 40 ]**

**do**

**if (( i % 3 == 0 || i % 7 == 0 ))** ##| | means OR

**then**

**echo "$i is either divisible by 3 or by 7 or both"**

**else**

**echo $i**

**fi**

**i=$(( i + 1 ))**

**done**

**example4b**

**i=1**

**while [ $i -le 40 ]**

**do**

**if (( i % 3 == 0 && i % 7 == 0 ))** ## && means and

**then**

**echo "$i is either divisible by 3 or by 7 or both"**

**else**

**echo $i**

**fi**

**i=$(( i + 1 ))**

**done**

example5

**j=1**

**while [ $j -le 10 ]**

**do**

**if (( j % 2 == 0 ))**

**then**

**echo "$j is an even number"**

**else**

**echo "$j is either odd or a prime number"**

**fi**

**j=$(( j + 1 ))**

**done**

example6

**## The while read also works better in areas where the for loop has a limitation, it doesn’t split when it sees any whitespace like space, tab**

**cat states.txt|while read line**

**do**

**echo "$line is in USA"**

**done**

#### **The until loop**

**Note that the only difference between while and until is the way the condition is handled. In while, the**

**loop executes as long as the condition is true; in until, it runs as long as the condition is false.**

**BUT: the until condition is checked at the top of the loop**

**COUNTER=20**

**until [ $COUNTER -lt 10 ]**

**do**

**echo COUNTER $COUNTER**

**let COUNTER-=1 ## (let COUNTER=COUNTER-1)**

**done**

# What is the difference between double ([[ ]]) and single ([ ]) square brackets in Shell Scripting?

* [ is the same as the test builtin, and works like the test binary (man test)
  + works about the same as [ in all the other sh-based shells in many UNIX-like environments
  + only supports a single condition. Multiple tests with the bash && and || operators must be in separate brackets.
  + doesn't natively support a 'not' operator. To invert a condition, use a ! outside the first bracket to use the shell's facility for inverting command return values.
  + == and != are literal string comparisons
* [[ is a bash
  + is bash-specific, though others shells may have implemented similar constructs. Don't expect it in an old-school UNIX sh.
  + == and != apply bash pattern matching rules, see "Pattern Matching" in man bash
  + has a =~ regex match operator
  + allows use of parentheses and the !, &&, and || logical operators within the brackets to combine subexpressions

1. **if [ condition ]**

This is the traditional shell test command. It is available on all POSIX shells. The test command sets an exit code and the if statement acts accordingly. Typical tests are whether a file exists or one number is equal to another.

**example**

**#!/bin/sh ##Using a bash shell**

**echo "How old are you?"**

**read age**

**if [ $age -ge 18 ]; then**

**echo "You may go to the party."**

**fi**

1. **if [[ condition ]]**

This is a new upgraded variation on test from ksh that bash and zsh also support. This test command also sets an exit code and the if statement acts accordingly. Among its extended features, it can test whether a string matches a regular expression.

**example1**

**#!/bin/sh ##Using a Bash shell**

**echo "How old are you?"**

**read age**

**if [[ $age -ge 18 ]]; then ##Needs semi-colon to work**

**echo "You may go to the party."**

**fi**

**example2**

**#!/bin/ksh ##Using a Korn shell**

**echo "How old are you?"**

**read age**

**if [[ $age -ge 18 ]] then ##No semi-colon and still work because it is a Korn shell**

**echo "You may go to the party."**

**fi**

1. **if (( condition ))**

Another ksh extension that bash and zsh also support. **This performs arithmetic**. As the result of the arithmetic, an exit code is set and the if statement acts accordingly. It returns an exit code of zero (true) if the result of the arithmetic calculation is nonzero. Like [[...]], this form is not POSIX and therefore not portable.

Example

**if (( j % 2 == 0 ))**

**then**

**echo "$j is an even number"**

**else**

**echo "$j is either odd or a prime number"**

**fi**

1. **if command**

command is executed and the if statement acts according to its exit code.

For example

#### ##check if the file grantuser40.sql has the phrase user40 in it,if ture , make directory user40

**if echo grantuser40sql| grep -q user40; then**

**echo "making directory"**

**mkdir user40**

**else**

**echo "No match"**

**fi**

**Example 2 -opposite**

**if echo grantuser40sql| grep -q gov; then**

**echo "making directory"**

**mkdir user40**

**else**

**echo "No Match"**

**fi**

# FUNCTIONS

**Functions in shell scripts, routine or methods in other programming languages are created for reusability. We also use functions to divide and conquer a big code, for example different parts of a code can be written using different functions.**

**function\_name ( ) {**

**command**

**}**

#### *## create a new function called Hello*

**\_hello () {**

**echo "Hello"**

**}**

**if [ -d util ]; then**

**\_hello**

**fi**

#### *## create a new function called Hello with an argument*

**\_hello () {**

**echo "Good Morning Mr. $1"**

**}**

**if [ -d util ]; then**

**\_hello Yaw**

**fi**

#### *##create a function to backup several databases on the same server*

**\_backup\_dbf\_files() {**

**cd $1**

**for dbf in $(ls sales\*.dbf)**

**do**

**cp $dbf $2/$dbf.bkup**

**done**

**}**

**##RMANCAT DB**

**\_backup\_dbf\_files /u01/app/oracle/oradata/RMANCAT/datafile /backup/RMANCAT**

**##database2**

**#\_backup\_dbf\_files /q01/app/oradata /backup/db2**

**##database3**

**#\_backup\_dbf\_files /d01/app/oradata /backup/db3**

**\_checkdivisible() {**

**i=1**

**while [ $i -le $1 ]**

**do**

**if (( i % 3 == 0 || i % 5 == 0 ))**

**then**

**echo "$i is either divisible by 3 or by 5 or both"**

**else**

**echo $i**

**fi**

**i=$(( i + 1 ))**

**done**

**}**

**\_checkdivisible 40**

#### *##Verify Database user’s passwords*

**\_verifypassword() {**

**sqlplus -s /nolog <<- EOF**

**whenever sqlerror exit 1**

**connect $1**

**exit**

**EOF**

**return $?**

**}**

**Please enter username/password**

**\_verifypassword scott/tiger**

# TIPS2

#### ***Wildcarding***

|  |  |
| --- | --- |
| Wildcard | Matches |
| ? | **Any single character** |
| \* | **Any string of characters** |
| [set] | **Any character in set** |
| [!set] | **Any character not in set** |

**farming.? could match farming.a , farming.l ,but can never be farming.co or farming.com**

**farming.\* could match anytime anything the begins with farming. ,for example farming.c, farming.xxx, farming.tttttttttttttttttttttttttttttt**

|  |  |
| --- | --- |
| Expression | Single character matched/meaning |
| [abc] | **a, b, or c** |
| [.,;] | **Period, comma, or semicolon** |
| [-\_] | **Dash or underscore** |
| [a-c] | **a, b, or c** |
| [a-z] | **Any lowercase letter** |
| [0-9] | **Any digit** |
| [!0-9] | **Any non digit** |
| [a-zA-Z] | **Any lower- or uppercase letter** |
| [a-zA-Z0-9\_-] | **Any letter, any digit, underscore, or dash** |

#### ***This will copy all the files that starts with sales and has numbers from 01-04 written as 0[1-4] and ends with .dbf***

**ls /u01/app/oracle/oradata/TRNGDB/sales0[1-4].dbf**

**sales01.dbf ,sales02.dbf to sales04.dbf**

#### ***This will list all .dbf files that begins with any alphabet from (c to s) in the directory***

**ls /u01/app/oracle/oradata/TRNGDB/[c-s]\*.dbf**

**example01.dbf sales01.dbf salesAR01.dbf sysaux01.dbf system01.dbf**

#### ***To scp all 20 dump files from one server to another***

**option1 -copy all the files one after the other (20times)**

scp scott\*.dmp server2:/tmp

**option2 -copy all the files one after the other (20times)**

scp scott01.dmp server2:/tmp

scp scott02.dmp server2:/tmp

scp scott03.dmp server2:/tmp

scp scott14.dmp server2:/tmp

............................

scp scott20.dmp server2:/tmp

**option3 -copy all the 20 files at the same time(Effect: slows down the system, as all the resources on the system will be used, network, disk contention, CP etc)**

scp scott01.dmp server2:/tmp &

scp scott02.dmp server2:/tmp &

scp scott03.dmp server2:/tmp &

scp scott14.dmp server2:/tmp &

............................

scp scott20.dmp server2:/tmp &

**option4 - copy all the 20 files at the same time(Effect: slows down the system, as all the resources on the system will be used, network, disk contention, CP etc)**

**scp scott\*.dmp server2:/tmp &**

**option5 -For each line below copy one file at a time(\*\*\*BEST\*\*- little effect on system resources)**

scp scott0[1-5].dmp server2:/tmp &

scp scott0[6-9].dmp server2:/tmp &

scp scott1[0-5].dmp server2:/tmp &

scp scott1[6-9].dmp server2:/tmp &

scp scott2[0-5].dmp server2:/tmp &

#### ***Note: In option 5, five files will be copied at the same time, one from each line, for example***

#### ***at one point scott01.dmp from line2 ,scott06.dmp from line2, scott10.dmp from line3,***

#### ***scott15 from line4 and scott20.dmp from line5 will be copied.***

#### ***To backup all your files using wildcard***

cp /u01/app/oracle/oradata/TRNGDB/[a-r]\*.dbf /backup/RMANCAT &

cp /u01/app/oracle/oradata/TRNGDB/[s]\*0[1-2].dbf /backup/RMANCAT &

cp /u01/app/oracle/oradata/TRNGDB/[s]\*0[3-4].dbf /backup/RMANCAT &

cp /u01/app/oracle/oradata/TRNGDB/[t-z]\*.dbf /backup/RMANCAT &

#### ***The grep command***

**grep is one of the most powerful command in UNIX and it is also very useful in shell scripting**

**grep is used to search for files, directories, phrase etc**

|  |  |
| --- | --- |
| Matching control | Meaning |
| -i | **Ignore case** |
| -v | **prints out only those lines that do not contain the word you are searching for. Exclude that specific command/pattern you are searching for** |
| -r | **Recursive search** |
| -w | **Select only those lines containing matches that forms the whole word** |
| -l | **Suppress normal output; instead print the name of each input file** |
| -q | **Quiet; do not write anything to standard output, works well with if/then/fi since it Exit immediately with zero status if any match is found** |
| -n | **Show the line number** |

#### ***Using grep -n to determine the line number of Maryland in the file states.txt***

**grep -n Maryland states.txt**

**20:Maryland**

**grep -n Maryland states.txt | cut -d':' -f1**

**20**

#### ***Things to know about grep***

**grep -w 't[a-i]e'**

Matches the words tee, the, and tie. The brackets have a special significance. They mean to match one character that can be anything from a to i.

**grep -w 't[i-z]e'**

Matches the words tie and toe.

**grep -w 'cr[a-m]\*t'**

Matches the words craft, credit, and cricket. The \* means to match any number of the previous character, which in this case is any character from a through m.

**grep -w 'kr.\*n'**

Matches the words kremlin and krypton, because the . matches any character and the \* means to match the dot any number of times.

**egrep -w '(th|sh).\*rt'**

Matches the words shirt, short, and thwart. The | means to match either the th or the sh. egrep is just like grep but supports extended regular expressions that allow for the | feature. [ The | character often denotes a logical OR, meaning that either the thing on the left or the right of the | is applicable. This is true of many programming languages. ] Note how the square brackets mean one-of-several-characters and the round brackets with |'s mean one-of-several-words.

**grep -w 'thr[aeiou]\*t'**

Matches the words threat and throat. As you can see, a list of possible characters can be placed inside the square brackets.

**grep -w 'thr[^a-f]\*t'**

Matches the words throughput and thrust. The ^ after the first bracket means to match any character except the characters listed

**ps -ef|grep ora|grep -v grep|grep -v TRNGDB**

**[oracle@trainingvm ~]$ ps -ef|grep smon**

**oracle 3904 1 0 Mar18 ? 00:00:08 ora\_smon\_TRNGDB**

**oracle 25296 20739 0 20:36 pts/0 00:00:00 grep smon**

**[oracle@trainingvm ~]$ ps -ef|grep smon|grep -v grep**

**oracle 3904 1 0 Mar18 ? 00:00:08 ora\_smon\_TRNGDB**

# AWK Programming

The awk command or GNU awk (GAWK) in specific provides a scripting language for text processing. With awk scripting language, you can make the following:

Define variables.

Use string and arithmetic operators.

Use control flow and loops.

**Generate formatted reports.**

**Actually, you can process log files that contain maybe millions of lines to output a readable report that you can benefit from.**

WHAT CAN WE DO WITH AWK ?

1. AWK Operations:

(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

2. Useful For:

(a) Transform data files

(b) Produce formatted reports

3. Programming Constructs:

(a) Format output lines

(b) Arithmetic and string operations

(c) Conditionals and loops

Syntax:

**awk options 'selection \_criteria {action }' input-file > output-file**

Awk can take the following options:

**-F fs To specify a file separator.**

**-f file To specify a file that contains awk script.**

**-v var=value To declare a variable.**

**$0 for the whole line.**

**$1 for the first field.**

**$2 for the second field.**

**$n for the nth field.**

cat addresses.txt

John Daggett, 341 King Road, Plymouth MA

Eric Adams, 20 Post Road, Sudbury MA

Sal Carpenter, 73 6th Street, Boston MA

#### ***Search for all the lines that contains MA in the address.txt and print that line***

**awk '/MA/ { print $0 }' addresses.txt**

John Daggett, 341 King Road, Plymouth MA

Eric Adams, 20 Post Road, Sudbury MA

Sal Carpenter, 73 6th Street, Boston MA

#### ***Search for all the lines that contains MA in the address.txt and print the first field of that line***

**awk '/MA/ { print $1 }' addresses.txt**

John

Eric

Sal

#### ***Search for all the lines that contains MA in the address.txt and print the first field of that line***

**awk '/MA/ { print $4 }' addresses.txt**

**awk '/MA/ { print $1"->"$3 }' addresses.txt**

**John->341**

**Eric->20**

**Sal->73**

**awk '/MA/ { print $1 "->" $3 }' addresses.txt**

**John->341**

**Eric->20**

**Sal->73**

**awk '/MA/ { print $1$3 }' addresses.txt**

**John341**

**Eric20**

**Sal73**

**awk '/MA/ { print $1 $3 }' addresses.txt**

**John341**

**Eric20**

**Sal73**

#### ***The -F below makes the Field separator to now be a comma instead of space***

**awk -F, '/MA/ { print $1 }' addresses.txt**

[oracle@trainingvm ~]$ awk -F, '/MA/ { print $1 }' addresses.txt

John Daggett

Eric Adams

Sal Carpenter

[oracle@trainingvm ~]$ awk -F, '/MA/ { print $2 }' addresses.txt

341 King Road

20 Post Road

73 6th Street

[oracle@trainingvm ~]$ awk -F, '/MA/ { print $3 }' addresses.txt

Plymouth MA

Sudbury MA

Boston MA

ps -ef|grep pmon

oracle 3874 1 0 Mar18 ? 00:00:16 ora\_pmon\_TRNGDB

oracle 24839 20739 0 19:54 pts/0 00:00:00 grep pmon

ps -ef|grep pmon|awk '{ print $2 }'

3874

24854

[oracle@trainingvm ~]$ ps -ef|grep pmon|grep -v grep|awk '{ print $2 }'

3874

[oracle@trainingvm ~]$ ps -ef|grep pmon|grep -v grep| awk '{ print **"User is :"$1"PID is:"$2** }'

User is :oraclePID is:3874

[oracle@trainingvm ~]$ ps -ef|grep pmon|grep -v grep| awk '{ print "User is :"$1, "PID is:"$2 }'

User is :oracle PID is:3874

[oracle@trainingvm ~]$ ps -ef|grep pmon|grep -v grep| awk '{ print "User is:"$1, ",PID is:"$2 }'

User is:oracle ,PID is:3874

**cat employee.txt**

ajay manager account 45000

sunil clerk account 25000

varun manager sales 50000

amit manager account 47000

tarun peon sales 15000

deepak clerk sales 23000

sunil peon sales 13000

satvik director purchase 80000

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

#### **awk '{print}' employee.txt**

ajay manager account 45000

sunil clerk account 25000

varun manager sales 50000

amit manager account 47000

tarun peon sales 15000

deepak clerk sales 23000

sunil peon sales 13000

satvik director purchase 80000

#### ***2. Print the lines which matches with the given pattern.***

#### **awk '/manager/ {print}' employee.txt**

ajay manager account 45000

varun manager sales 50000

amit manager account 47000

#### ***2b. Prints all the lines in the employee.txt file that contains numbers from 0 to 9.***

**awk '/[0-9]/{print}' employee.txt**

ajay manager account 45000

sunil clerk account 25000

varun manager sales 50000

amit manager account 47000

tarun peon sales 15000

deepak clerk sales 23000

sunil peon sales 13000

satvik director purchase 80000

#### 2c. **This will print all the lines in employee.txt that begins with a number (^ -beginning of a line)**

#### 

#### **awk '^/[0-9]/{print}' employee.txt**

2d. **This will print all the lines in employee.txt that ends with a number ( $ >end of a line)**

**awk '/[0-9]$/{print}' employee.txt**

3. **Spliting 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}' employee.txt**

ajay 45000

sunil 25000

varun 50000

amit 47000

tarun 15000

deepak 23000

sunil 13000

satvik 80000

#### ***Built In Variables In 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.

#### ***Example***

echo 'one two three four' | awk '{ print $1, $2, $3 }'

one two three

$ echo 'one two three four' | awk '{ OFS = "..."; print $1, $2, $3 }'

one...two...three

$ echo 'one two three four' | awk '{ OFS = "\n"; print $1, $2, $3 }'

one

two

three

**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.

Examples:

#### ***Use of NR built-in variables (Display Line Number)***

**awk '{print NR, $0}' employee.txt**

1 ajay manager account 45000

2 sunil clerk account 25000

3 varun manager sales 50000

4 amit manager account 47000

5 tarun peon sales 15000

6 deepak clerk sales 23000

7 sunil peon sales 13000

8 satvik director purchase 80000

#### ***Use of NF built-in variables (Display Last Field)***

**[oracle@trainingvm ~]$ awk '{print $1, $NF}' addresses.txt**

**John MA**

**Eric MA**

**Sal MA**

**[oracle@trainingvm ~]$ awk '{print $1, $NF, NR}' employee.txt**

**ajay 45000 1**

**sunil 25000 2**

**varun 50000 3**

**amit 47000 4**

**tarun 15000 5**

**deepak 23000 6**

**sunil 13000 7**

**satvik 80000**

#### ***Another use of NR built-in variables (Display Line From 3 to 6)***

**awk 'NR==3, NR==6 {print NR,$0}' employee.txt**

3 varun manager sales 50000

4 amit manager account 47000

5 tarun peon sales 15000

6 deepak clerk sales 23000

#### ***To print the first item along with the row number(NR) separated with “->”***

**awk '{print NR,$1"->"$NF}' employee.txt**

1 ajay->45000

2 sunil->25000

3 varun->50000

4 amit->47000

5 tarun->15000

6 deepak->23000

7 sunil->13000

8 satvik->80000

#### ***To count the total lines in a file***

**awk 'END { print NR }' employee.txt**

8

#### ***Printing lines with more than 10 characters:***

**awk 'length($0) > 10' employee.txt**

**awk '/ [0-9] / {print}' employee.txt**

#### ***Example1 and Example 2 below perform the same thing (Note: ~ means matches)***

**[oracle@trainingvm ~]$ awk -F= '$1 ~ /th/ { print $0 }' addresses.txt**

John Daggett, 341 King Road, Plymouth MA

Sal Carpenter, 73 6th Street, Boston MA

**[oracle@trainingvm ~]$ awk -F= '/th/ { print $0 }' addresses.txt**

John Daggett, 341 King Road, Plymouth MA

Sal Carpenter, 73 6th Street, Boston MA

#### ***These matches all the lines that begins with Jo***

[oracle@trainingvm ~]$ **awk -F= '/^Jo/ { print $0 }' addresses.txt**

John Daggett, 341 King Road, Plymouth MA

[oracle@trainingvm ~]$ **awk -F= '$1 ~ /^Jo/ { print $0 }' addresses.txt**

John Daggett, 341 King Road, Plymouth MA

# SED and EX programming

#### ***Syntax***

**sed 's/pattern/replacement/g' inputFileName > outputFileName**

**or**

**sed -e 's/pattern/replacement/g' inputFileName > outputFileName**

-e indicate that an expression follows.

s stands for substitute,

g stands for global, which means that all matching occurrences in the line would be replaced.

#### *This command will change all occurrences of New to Old in states.txt and put them in the states2.txt. Note that states.txt will not be modified only states2.txt will be changed*

**[oracle@trainingvm ~]$ sed 's/New/Old/g' states.txt >states2.txt**

**[oracle@trainingvm ~]$ vi states2.txt**

|  |  |
| --- | --- |
| Regular expression | Meaning |
| caret (^) | means the beginning of a line. |
| dollar sign ($) | means the end of a line |
| Empty line (^$) | means **empty line**, that is beginning (^) and end ($) of line with no data |
| asterisk (\*) | matches zero or more occurrences of the previous character. |
| Dot (.) | Match any single character |
| ab|cd | Either ab or cd |
| a(b\*|c\*)d | Matches any string beginning with a, followed by b’s (could be no b’s at all or some b’s), followed by c’s (could be no c’s at all or some c’s), and finally d |
| (character)\* | Match as many occurrences of any character |
| (character)? | Match 0 or 1 instance of characters**, no duplicate** |
| (character)+ | Match 1 or more instance of characters, |

#### Examples

**[oracle@trainingvm ~]$ echo busx lazx funnx**

**busx lazx funnx**

**[oracle@trainingvm ~]$ echo busx lazx funnx | sed 's/x/y/g'**

**busy lazy funny**

#### **Options**

The -i option, allows in-place editing of files (it creates a temporary file behind the scene, and then the original file is replaced by the temporary file and save it).

For example:

#### *This command will change all occurrences of New to Old in states.txt*

**[oracle@trainingvm ~]$ sed -i 's/New/Old/g' states.txt**

sed -i 's/abc/def/g' sample.txt

cat sample.txt

abcdefgabcgabcdeiwabc

#### **To replace any instance of a certain word in a file with another, for replace manager with supervisor and save the file**

sed -i 's/manager/supervisor/g' employee.txt

[oracle@trainingvm ~]$ sed -i 's/clerk/salesman/g' employee.txt

[oracle@trainingvm ~]$ vi employee.txt

#### **To change different patterns at the same time in a file, for example this command changes salesman to clerk and supervisor to manager in the employee.txt file**

cat employee.txt

ajay supervisor account 45000

sunil salesman account 25000

varun supervisor sales 50000

amit supervisor account 47000

tarun peon sales 15000

deepak salesman sales 23000

sunil peon sales 13000

satvik director purchase 80000

[oracle@trainingvm ~]$ **sed -i 's/salesman/clerk/g ; s/supervisor/manager/g' employee.txt**

cat employee.txt

ajay manager account 45000

sunil clerk account 25000

varun manager sales 50000

amit manager account 47000

tarun peon sales 15000

deepak clerk sales 23000

sunil peon sales 13000

satvik director purchase 80000

#### **d**

#### **To delete any line containing the word "New York"**

#### **sed /"New York"/d states2>states3**

#### [oracle@trainingvm ~]$ vi states3

#### [oracle@trainingvm ~]$ **sed -i /"North Carolina"/d states2**

#### [oracle@trainingvm ~]$ vi states2

#### ***-n***

#### ***The -n option tells sed to print only those lines matching the pattern.***

#### Advance Sed

cat application.xml

|  |
| --- |
| <oa\_environments<  <adconfig<  <adconfig\_file oa\_var="s\_adconfig\_file">/u01/R122\_EBS/EBSapps/appl/admin/adconfig.txt</adconfig\_file>  <APPL\_TOP oa\_var="s\_at\_adconfig>">/u01/R122\_EBS/EBSapps/appl</APPL\_TOP>  <APPL\_TOP\_CSET oa\_var="s\_at\_cset">AL32UTF8</APPL\_TOP\_CSET>  <APPS\_ENV\_NAME oa\_var="s\_appsEnvName">VISION</APPS\_ENV\_NAME>  <TIER\_ADADMIN oa\_var="s\_isAdAdmin">YES</TIER\_ADADMIN>  <TIER\_ADWEB oa\_var="s\_isAdWeb">YES<TIER\_ADWEB>  <TIER\_ADFORMS oa\_var="s\_isAdForms">YES</TIER\_ADFORMS>  <TIER\_ADNODE oa\_var="s\_isAdConc">YES</TIER\_ADNODE>  <domain oa\_var="s\_domainname">ishglo.com</domain>  <APPL\_TOP\_NAME oa\_var="s\_atName">testsys100</APPL\_TOP\_NAME>  <TIER\_ADFORMSDEV oa\_var="s\_isAdFormsDev">YES</TIER\_ADFORMSDEV>  <TIER\_ADNODEDEV oa\_var="s\_isAdConcDev">YES</TIER\_ADNODEDEV>  <TIER\_ADWEBDEV oa\_var="s\_isAdWebDev">YES</TIER\_ADWEBDEV>  <STAGING\_DIRECTORY oa\_var="s\_staging\_directory"</u01/app/oracle/product/12.1.0.2/appltmp/stage</STAGING\_DIRECTORY>  </adconfig<  <oa\_environment type="generic\_service">  <DISPLAY oa\_var="s\_display" osd="unix">LOCALHOST:5.0</DISPLAY>  <java\_awt\_headless oa\_var="s\_java\_awt\_headless">TRUE</java\_awt\_headless>  </oa\_environment>  <oa\_environment type="rapid\_install"<  <APPS\_BASE oa\_var="s\_base"</u01/R122\_EBS</APPS\_BASE>  <PATCH\_BASE oa\_var="s\_patch\_base"</u02/R122\_EBS</PATCH\_BASE>  <INST\_BASE oa\_var="s\_inst\_base"</u01/R122\_EBS/inst</INST\_BASE>  <HTML\_TOP oa\_var="s\_html"</u01/R122\_EBS/FMW\_Home/Oracle\_EBS-app1/applications/oacore/html</HTML\_TOP>  <JDK\_TOP oa\_var="s\_jdktop" osd="LINUX\_X86-64"</u01/R122\_EBS/FMW\_Home/jrockit\_160\_22\_D1.1.1-3</JDK\_TOP>  <JRE\_TOP oa\_var="s\_jretop"</u01/R122\_EBS/FMW\_Home/jrockit\_160\_22\_D1.1.1-3/jre</JRE\_TOP> |

#### **To get the staging directory path, here are the steps to follow**

**[oracle@trainingvm ~]$ grep -i s\_staging\_directory application.xml**

***<STAGING\_DIRECTORY oa\_var="s\_staging\_directory"</u01/app/oracle/product/12.1.0.2/appltmp/stage</STAGING\_DIRECTORY>***

**[oracle@trainingvm ~]$ grep -i s\_staging\_directory application.xml |sed 's/<.\*"<//'**

**/u01/app/oracle/product/12.1.0.2/appltmp/stage</STAGING\_DIRECTORY>**

**[oracle@trainingvm ~]$ grep -i s\_staging\_directory application.xml |sed 's/<.\*"<//'|sed 's/<.\*>//'**

**/u01/app/oracle/product/12.1.0.2/appltmp/stage**

#### **You can use this as a variable**

**STAGING\_DIR=$(grep -i s\_staging\_directory application.xml |sed 's/<.\*"<//'|sed 's/<.\*>//')**

**echo ${STAGING\_DIR}**

#### **To get the domain name(ishglo.com), here are the steps to follow**

**[oracle@trainingvm ~]$ grep -i s\_domainname application.xml**

**<domain oa\_var="s\_domainname">ishglo.com</domain>**

**[oracle@trainingvm ~]$ grep -i s\_domainname application.xml |sed 's/<.\*">//'**

**ishglo.com</domain>**

**[oracle@trainingvm ~]$ grep -i s\_domainname application.xml |sed 's/<.\*">//'|sed 's/<.\*>//'**

**ishglo.com**

#### **You can use this as a variable**

**DOMAIN\_NAME=$(grep -i s\_domainname application.xml |sed 's/<.\*">//'|sed 's/<.\*>//')**

**ccho ${DOMAIN\_NAME}**

#### **To get the ORACLE\_HOME, here are the steps to follow**

**[oracle@trainingvm ~]$ grep -i staging\_directory application.xml |sed 's/<.\*"<//'|sed 's/<.\*>//'|cut -f 1-6 -d'/'**

**/u01/app/oracle/product/12.1.0.2**

**ORACLE\_HOME=$(grep -i staging\_directory application.xml |sed 's/<.\*"<//'|sed 's/<.\*>//'|cut -f 1-6 -d'/')**

#### **Using Variables in SED**

**This will not work**

**for i in 1 2 3 4**

**do**

**sed 's/supervisor/supervisor$i/' employee.txt>employee$i.txt**

**done**

**This works (replace $i with ' "$i" ' )**

#### **This will generate multiple(1-10) files and update the content of each file with a corresponding number**

**for i in 1 2 3 4 5 6 7 8 9 10**

**do**

**sed 's/supervisor/supervisor'"$i"'/' employee.txt>employee$i.txt**

**done**

#### **output**

-rw-r--r--. 1 oracle oinstall 227 Mar 30 11:19 employee1.txt

-rw-r--r--. 1 oracle oinstall 227 Mar 30 11:19 employee2.txt

-rw-r--r--. 1 oracle oinstall 227 Mar 30 11:19 employee3.txt

-rw-r--r--. 1 oracle oinstall 227 Mar 30 11:19 employee4.txt

-rw-r--r--. 1 oracle oinstall 227 Mar 30 11:19 employee5.txt

-rw-r--r--. 1 oracle oinstall 227 Mar 30 11:19 employee6.txt

-rw-r--r--. 1 oracle oinstall 227 Mar 30 11:19 employee7.txt

-rw-r--r--. 1 oracle oinstall 227 Mar 30 11:19 employee8.txt

-rw-r--r--. 1 oracle oinstall 227 Mar 30 11:19 employee9.txt

-rw-r--r--. 1 oracle oinstall 230 Mar 30 11:19 employee10.txt

#### **cat employee2.txt**

ajay supervisor2 account 45000

sunil salesman account 25000

varun supervisor2 sales 50000

amit supervisor2 account 47000

tarun peon sales 15000

deepak salesman sales 23000

sunil peon sales 13000

satvik director purchase 80000

#### **cat employee8.txt**

ajay supervisor8 account 45000

sunil salesman account 25000

varun supervisor8 sales 50000

amit supervisor8 account 47000

tarun peon sales 15000

deepak salesman sales 23000

sunil peon sales 13000

satvik director purchase 80000

#### **This script looks for the keyword "#extproc", if this there (commented out), exit otherwise use grep to determine the line number of extrpoc and add 9 lines to it and comment out using # from the line number of the extproc to the last 9 lines**

set -x

. $HOME/.bash\_profile

TNSNAMESORA=${TNS\_ADMIN}/tnsnames.ora

cp ${TNSNAMESORA} ${TNSNAMESORA}~

CHECKEXTPROC=/tmp/checkextproc.$$

grep -n '#extproc' ${TNSNAMESORA}>${CHECKEXTPROC}

FIRSTNUMBER=$(grep -n 'extproc' ${TNSNAMESORA}|cut -d':' -f1)

let LASTNUMBER=( ${FIRSTNUMBER} + 9 )

if [ -s ${CHECKEXTPROC} ]; then

echo "extproc is commented out"

exit 0;

else

sed -i ''"${FIRSTNUMBER}"','"${LASTNUMBER}s"'/^/#/' ${TNSNAMESORA}

fi

rm ${CHECKEXTPROC}

# EX

#### **The simplest global replacements substitute one word (or a phrase) for another.**

**:%s/director/editor/g**

This substitutes ***editor*** for every occurrence of ***director*** throughout the file.

**ex -s employee.txt <<EOF**

**:%s/supervisor/Director/g**

**wq**

**EOF**

#### **More complex syntax for global replacement.**

This syntax lets you search for a pattern, and then, once you find the line with the pattern, make a substitution on a string different from the pattern. You can think of this as context-sensitive replacement.

The syntax is as follows:

**:g/*pattern*/s/*old*/*new*/g**

The first g tells the command to operate on all lines of a file. *pattern* identifies the lines on which a substitution is to take place. On those lines containing *pattern*, *ex* is to substitute (s) for *old* the characters in *new*. The last g indicates that the substitution is to occur globally *on that line*.

#### **This will search for ALL password keywords and replace the one with sql-wqa to change\_4now**

**ex -s ishie.xml <<EOF**

**g/password/s/sql-wqa/change\_4now/g**

**wq**

**EOF**

#### **This will search for the first password keyword it finds and replace it with sql-wqa to change\_4now**

**ex -s ishie.xml <<EOF**

**/password/s/sql-wqa/change\_4now/**

**wq**

**EOF**

#### **Note: whiles the search pattern needs to be in this format /pattern/, the substitute can take any legal character,for example**

**ex -s ishie.xml<<EOF**

**/connect\_string/s;JOHN;CAR;**

**wq**

**EOF**