**INDEX**

Contents

[Oracle SQL 2](#_Toc453853760)

[SELECT 2](#_Toc453853761)

[INSERT 2](#_Toc453853762)

[DELETE 2](#_Toc453853763)

[UPDATE 2](#_Toc453853764)

[ALTER 2](#_Toc453853765)

[WHERE 2](#_Toc453853766)

[GROUP BY 2](#_Toc453853767)

[HAVING 2](#_Toc453853768)

[ORDER BY 2](#_Toc453853769)

[CREATE SEQUENCE 2](#_Toc453853770)

[FUNCTION 2](#_Toc453853771)

[Unix 2](#_Toc453853772)

[Basic Commands – (mkdir, touch, cd, ls, tree, .., ., pwd, ~, $HOME) 2](#_Toc453853773)

[Multi-user 3](#_Toc453853774)

[VIM 4](#_Toc453853775)

[$() command substitution 8](#_Toc453853776)

[Exercises 9](#_Toc453853777)

[Exercise 1 – Create a directory structure 9](#_Toc453853778)

[Exercise 2 – Change a directory structure 9](#_Toc453853779)

[Exercise 3 – The vi Editor 10](#_Toc453853780)

[Execise 4 - Data Commands 11](#_Toc453853781)

[Exercise 5a: Chmod 11](#_Toc453853782)

[Exercise 5b: Using the Find Command 11](#_Toc453853783)

[Exercise 6 – Stream Redirection on the Command Line 11](#_Toc453853784)

[Exercise 7 – Regular Expressions 12](#_Toc453853785)

[Exercise 8 - $, $(( ... )), $[ ... ], $( ... ) 12](#_Toc453853786)

[Answers Ex 8 13](#_Toc453853787)

[Exercise 9 - Job Control 16](#_Toc453853788)

[Shell Scripting 17](#_Toc453853789)

[Exercise 1 17](#_Toc453853790)

[Answers – Exercise 1 18](#_Toc453853791)

[Exercise 2 19](#_Toc453853792)

[Answers – Exercise 2 21](#_Toc453853793)

[Exercise 3 21](#_Toc453853794)

[Answers – Exercise 3 23](#_Toc453853795)

[Exercise 4 25](#_Toc453853796)

[Passing Arguments 26](#_Toc453853797)

[User Input 27](#_Toc453853798)

[Arrays 27](#_Toc453853799)

[Arithmetic/Math 28](#_Toc453853800)

[Exit Status 28](#_Toc453853801)

[If Statement 28](#_Toc453853802)

[Testing (If statement) 29](#_Toc453853803)

[Testing Example 30](#_Toc453853804)

[Conditional Operations/Operators (If statement) 30](#_Toc453853805)

[Case (if statement) 31](#_Toc453853806)

[Syntax: 31](#_Toc453853807)

[Example: 31](#_Toc453853808)

[With Pattern 31](#_Toc453853809)

# Oracle SQL

SELECT

INSERT

DELETE

UPDATE

-updates table

ALTER

-alters structure of table

WHERE

GROUP BY

HAVING

ORDER BY

CREATE SEQUENCE

FUNCTION

# Unix

Basic Commands – (mkdir, touch, cd, ls, tree, .., ., pwd, ~, $HOME)

Each user their own home directory

**ls** display the content of a directory

**tree** can be used to view a hierarchical structure of the file system

**cd** Navigate your way to other directories

**mkdir** Create a directory or multiple directories. **mkdir –p sub1/sub2/sub3** to create subdirectories

**rmdir** remove empty directories

**rm –r** remove directories with or without content

**rmdir** and **rm** remove directories and there is no way to get them back once they were remove

-eq is for numbers

= is for strings

echo – sends to stdout appending \n at the end. User printf if you don’t want \n

echo –e allows escape characters

printf – sends to stdout without \n

mkdir

mkdir –p -p - Create any missing intermediate pathname components.

touch – create empty file

cd

ls

tree – structure of all files in current directory

.. – previous directory (parent directory)

. – current directory

pwd – path of current directory

$HOME or ~ - home directory

~ - root

${} – length of a variable

Multi-user

-multiple users accessing 1 OS at the same time

**Remove files/directory**

rmdir path/name if the dir is empty

rm –r path/name if the dir is not empty

rm (remove files – can also be used for directories)

rm –i (interactive)(to confirm the delete for each file before doing it)

rm –r (recursive) (to remove non-empty directories)

rm –v (verbose)(print the steps that have been done)

rm –f (force)(even if the file doesn’t exist won’t prompt)

**Globbing (basically regex)**

|  |  |
| --- | --- |
| **\*** | Match zero or more characters |
| **?** | Match one character |
| **[ ]** | Match anything in the [ ] for 1 character position e.g “[Tt]he” matches The or the |
| **[a-e]** | The – is a range separator. This will match a to e |
| **[!0-9]** | ! = negate. This will match anything except 0 to 9 |
| **\** | Escape the metacharacter and treat them as a literal |

Eg. Ls ~/[!0-9][c-t]\*n

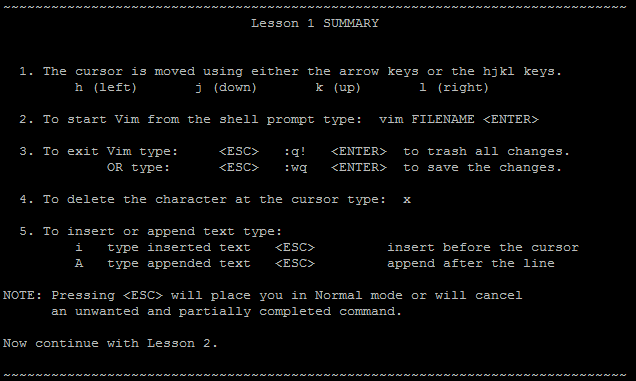
mv, rm, glob matching, and brace expansion

mv “what you want to move” ... “what you want to move” “where you want to move it to”

touch ~/films/action/rocky{1..5}

creates files rocky1, rocky2, rocky3, rocky4, rocky5

## VIM



dw, de, d$

Where:

d - is the delete operator.

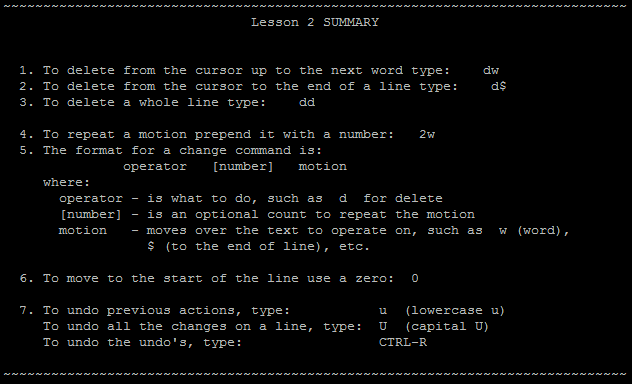
motion - is what the operator will operate on (listed below).

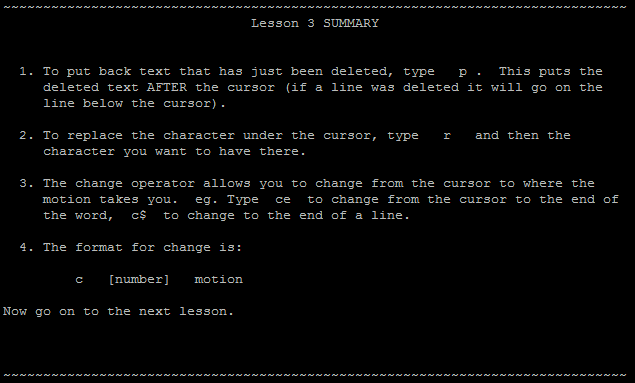
A short list of motions:

w - until the start of the next word, EXCLUDING its first character.

e - to the end of the current word, INCLUDING the last character.

$ - to the end of the line, INCLUDING the last character.





**wc, fgrep, cut, sort**

**echo**

**Scroll through a file**

less path/filename

more path/filename

cat path/filename: displays the contents of a file or files as one continuous stream

head -5 /examples/lionsInTheStreet retrieves the first 5 lines

head -c100 /examples/lionsInTheStreet retrieves the first 100 characters

cut -c2 filename gets the second character of each line

cut -c-5 filename gets the first five characters

cut -c5- filename gets the fifth character to the end of the line

cut -c3-7 filename gets the third through seventh char

cut -c3,7 filename gets the third and seventh char

cut -d ":" -f2 accounts using ":" as a delimiter, retrieve the second field.

cut -d ":" -f2,4 accounts using ":" as a delimiter, retrieves the second and fourth field.

cut -d ":" -f2-4 accounts using ":" as a delimiter, retrieves the second through fourth fields.

fgrep: searches file(s) for lines that match a fixed string with –f

sort –r file1 sort in reverse order

sort –t":" –k4 accounts Using ":" as the delimiter, sort the lines by each key starting with the fourth key to the end of the line. Note similarities and differences from cut.

sort –k2 accounts Using any space (or spaces) as the delimiter, sort by the lines by each key starting with the second key to the end of the line.

sort –t":" –k3,5 accounts Using ":" as the delimiter, sort the lines by the third, fourth and the fifth key. If there are more keys after the fifth, don't bother sorting by those keys.

sort -n –t":" –k2 accounts using ":" as the delimiter, sort the lines by the second key and sort in numeric order rather than alphabetic.

sort –u accounts “Unique” - After sorting, display any line that is repeated

uniq myFile displays lines from the file but eliminates repeats if that repeat occurs on the very next line. To eliminate all duplicates the input file must be sorted.

uniq –c myFile displays the number of times each line is repeated as well as the line from the file.

uniq –d myFile displays only lines that are repeated.

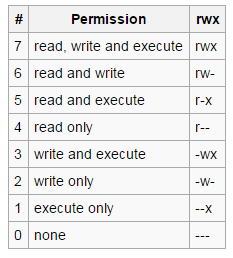
wc -l myFile number of lines

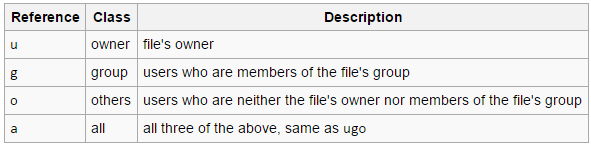
wc -w myFile number of words (delimited by white space)

wc -c myFile number of bytes (Note: this includes the \n character. Giving +1 to result. Same with -m)

wc -m myFile number of characters

chmod +rwx filename





find myFile finds myFile in the current directory

find . -name myFile finds myFile in current directory and its sub-folders

. indicates the current directory, which is the default

find . -size -10k finds any file less than 10k in size

find . -size +10k finds any file over 10k in size

find . -type f -size +10k finds any file over 10k. Type uses f for file, d for directory, l for link

find /usr /bin -name myFile finds an item in specific directories (item – file/directory)

find . -mtime -2 finds items modified during the past 2 days

find . -mmin -60 finds items modified in the past 60 minutes

find . -mmin +60 finds items modified more than 60 minutes ago

find . -name myFile –print -print is the default action and not always required

find . -name "The\*" -exec rm {} \; execute rm on all files starting with “The” in the current directory and every sub-directory.

find . -name "The\*" -exec rm -i {} \; an interactive version

0 std input

1 std output

2 std error

command > file stdout to file

command >> file stdout appending to file

command 2> file stderr to file

2>&1 stderr to stdout

Piping

Tr – replace a pattern with another one

echo James:Taylor | tr ":" " " replaces ":" with a space character

echo chris | tr crh Evl turns chris into Elvis

echo -n abcde123 | tr -c aeiou - outputs a---e--- -c means compliment set. This example turns any character not in aeiou to hyphens.

cat myFile | tr [:lower:] [:upper:] turn lowercase characters into uppercase

echo hello | tr -d h deletes all the 'h' character (ello)

echo hello | tr -d ho deletes all instances of the 'h' and 'o' (ell)

cat myFile | tr -d a-h deletes all lower case chars between a and h (inclusive)

cat myFile | tr -d [:upper:] deletes all capital letters from the output

echo NA1234567D | tr -d [:alpha:] deletes all letters (1234567)

echo NA1234567D | tr -d [:digit:] deletes all numbers

echo ABBBBBBD | tr –s “B” Squeeze Bs down to one B.

Tee - Writes the output of a command to stdout and the specified file simultaneously.

data | tree file -> date stdout is piped to tee -> stdout is sent to monitor

-> Tee sends a copy of stdout to file

Input Redirection (instead of keyboard use)

<<< - A STRING. redirect a string to the std input of a command – eg tr[:digit:] ‘!’ <<< jk81n889ACDB

<< - A BLOCK OF TEXT. (“Here document” - read until the end of the doc marker) block of text

< - A FILE

eg cat << EOF

<Hi

<Hello

<Welcome

<EOF

--displays everything but EOF

Inode/i-node

Ls –i = show the inode number

Ls –l = show you the inode content followed by name

stat –c%i filename: displays the inode of specific file

hard link & soft link

ln –s or ln

## $() command substitution

Usage of the $ like $(echo foo) means run whatever is inside the parentheses in a subshell

Command substitution allows us to treat the output of a command as a value.

line=$(grep "beast" filename)

echo $line

drwxr-x--- 2 raymond.to sg^unix^trainee^home^folder^access 4096 Jun 14 16:11 zombie

$ls -l | sort -r -t " " -k6 -k7 -n -k8 sorts by date time column 6 is month, 7 is day, 8 is time

# Exercises

## Exercise 1 – Create a directory structure

|  |
| --- |
| **Task** |
| Create the following directory structure within your **home** directory.  You can perform this task using only **mkdir** and **touch**, and **cd**.  To check your structure, use **ls** and **tree**. |
| ~/  |  |  films/  |  |  --------------------------------------------  | | |  | | |  horror/ comedy/ action/  theHowling bigMommasHouse rambo  | ghostbusters   -------------------- ghostbusters2  | | shawnOfTheDead  | |  zombie/ slasher/  dawnOfTheDead halloween  fridayThe13th  predator |

## Exercise 2 – Change a directory structure

|  |
| --- |
| **Task** |
| The following tasks involve: **mv, rm, glob matching, and brace expansion**  Your films folder should currently look like this: |
| ~/  |  |  films/  |  |  --------------------------------------------  | | |  | | |  horror/ comedy/ action/  theHowling bigMommasHouse rambo  | ghostbusters   -------------------- ghostbusters2  | | shawnOfTheDead  | |  zombie/ slasher/  dawnOfTheDead halloween  fridayThe13th  predator |

Perform the following actions

1. Navigate to the 'zombie' folder. Stay in the zombie folder throughout this exercise.
2. Move 'predator' from the 'slasher' folder to the 'action' folder.
3. Rename 'rambo' to 'firstBlood'.
4. Create a sub-folder within comedy called 'supernaturalComedy'.
5. Using only one command, move both 'Ghostbusters' films into the new folder.
6. Delete 'bigMommasHouse' in interactive mode.
7. With a single command, create ‘rockyI’, ‘rockyII’, … ‘rockyV’ in the action folder.

## Exercise 3 – The vi Editor

Use vi to create the “edit\_me” file.

Try the following commands in command mode

i - lowercase i

I - uppercase i

dw

dd

D

p

x 5x 10x

o - lowercase "oh"

O - uppercase "oh"

A

ZZ

Use vi to create the ‘accounts’ file:

**PIN534:Sheldon Cooper:1:1024**

**PIN756:Leonard Hofstadter:2:620**

**PIN769:Howard Wolowitz:4:213**

**PIN210:Rajesh Koothrappali:3:556**

The first field is the pin. The second field is the account holder’s name. The third field is the account number. The fourth field is the account's balance.

## Execise 4 - Data Commands

Commands Needed: **wc, fgrep, cut, sort**

1. Count the number of characters in the ‘accounts’ file. wc –m accounts
2. Count the number of words in the ‘accounts’ file. wc –w accounts
3. Count the number of lines in the ‘accounts’ file. wc –l accounts
4. Display the line containing ‘Sheldon Cooper’. grep “Sheldon Cooper” accounts
5. Display the account that has id of 3. (tip: look for ":3:") grep “:3:” accounts
6. Display the account with a balance of 620. grep “:620” accounts
7. Retrieve and display the name of every person represented in the ‘accounts’ file. cat accounts
8. Retrieve the pin numbers for every person represented in the ‘accounts’ file. cut -d ":" -f1 accounts
9. Retrieve the balance and the real name for every person in the ‘accounts’ file. cut -d ":" –f2,4 accounts
10. Display the entries in the ‘accounts’ file sorted by account balance in descending order. sort -t":" -n -r -k4 accounts

## Exercise 5a: Chmod

Create a regular file and run through the following steps:

1. Alter permissions using **chmod** so that you have full access to the file and nobody else has any access. Verify that this has worked. chmod u+rwx filename, chmod go-rwx filename
2. Give yourself read-write access, the file's group read access only and no access to anyone else. chmod u+rw filename, chmod g+r filename, chmod o-rwx filename
3. Give yourself read-execute access, and execute only access to all others.

Create a directory.

1. Give yourself read and execute access but no write access. What does this prevent you from doing?
2. Give yourself all privileges except for read access. What does this prevent you from doing?
3. Deny yourself execute access on the directory. What does this prevent you from doing?

## Exercise 5b: Using the Find Command

From the films directory:

1. Locate all files named ‘shawnOfTheDead’ in your films directory and sub-directories. find . -name "shawnOfTheDead"
2. Locate all files in all subdirectories that were modified in the last 30 minutes. find . –mmin -30
3. Locate all files in all subdirectories with either ‘the’ or ‘The’ in the name. find . -name "\*[Tt]he\*"
4. Turn off write permission for all files that contain ‘the’ or ‘The’ in the name. find . -name "\*[Tt]he\*" -exec chmod -w {} \;
5. Interactively remove all files with ‘the’ or ‘The’ in the name. (You can keep the files if you answer ‘n’ when prompted.) find . -name "\*[Tt]he\*" -exec rm -r {} \;

## Exercise 6 – Stream Redirection on the Command Line

Perform the following actions using a single command on the command line.

1. Display the contents of the ‘accounts’ file in upper case. cat accounts | tr [:lower:] [:upper:]
2. Display the contents of the ‘accounts’ file with all the colons changed to spaces. cat accounts | tr ":" " "
3. Retrieve the line containing ‘Sheldon Cooper’ from the accounts file and redirect it into a file called "sheldon". If the file "sheldon" already exists then overwrite the file. grep "Sheldon Cooper" accounts > sheldon
4. Retrieve the line containing ‘Leonard’ and append it into the sheldon file. grep "Leonard" accounts >> sheldon
5. Use echo and wc to count the characters in the word “characters”. echo characters | wc -m . use printf instead of echo
6. Display the number of lines in the ‘accounts’ file without displaying the name of the file. wc -l accounts | cut -d " " -f1
7. Use sort to order the output of ls -l based on various different fields. ls -l | sort -t" " -k9

## Exercise 7 – Regular Expressions

Paste the following data into a file:

07999234123

075435345623

07AAAAAAAAA

07bbbbbbbbb

B41RTG

01223567345

A345GUF

B3GHJ

768745

KL563478K

kl563478kx

[l563478k

6535345354554234

B45JKHL

65-34-76

012234567867

07856456636

BB4JUK

56-567-67

B546HUY

BB345614H

07666345234

45-34-25

01223456755

JH761423G

Use **grep or egrep** to extract the following:

1. Mobile phone numbers (11 digits starting in '07') egrep "^07[0-9]{9}$" filename
2. National insurance numbers (2 letters, 6 digits, 1 letter)

egrep "^[a-z|A-Z]{2}[0-9]{6}[a-z|A-Z]$" filename OR

egrep "[a-zA-Z]{2}[[:digit:]]{6}.$” filename OR

egrep "^[[:alpha:]]{2}[[:digit:]]{6}[[:alpha:]]$" filename

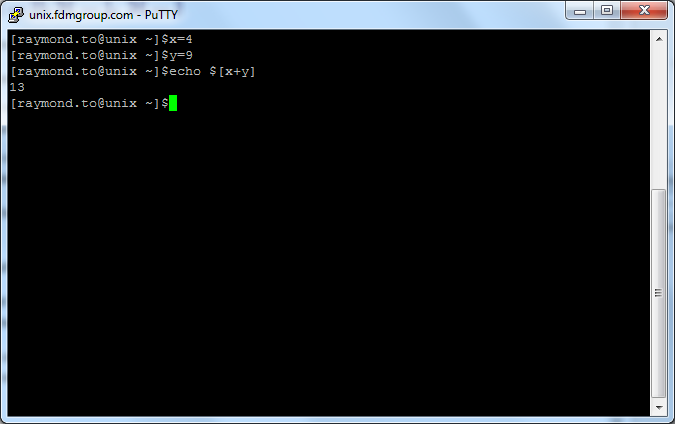
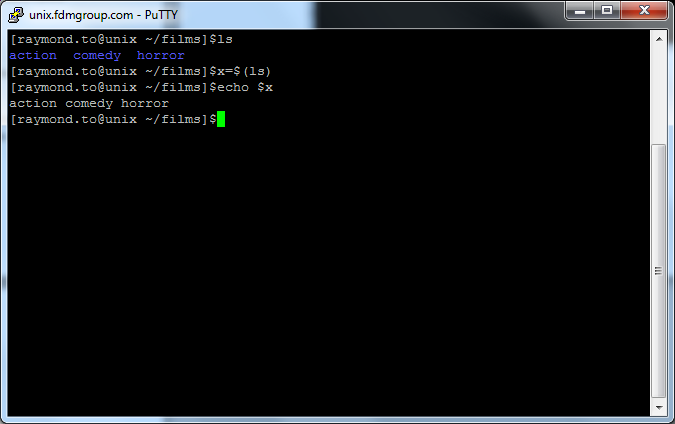
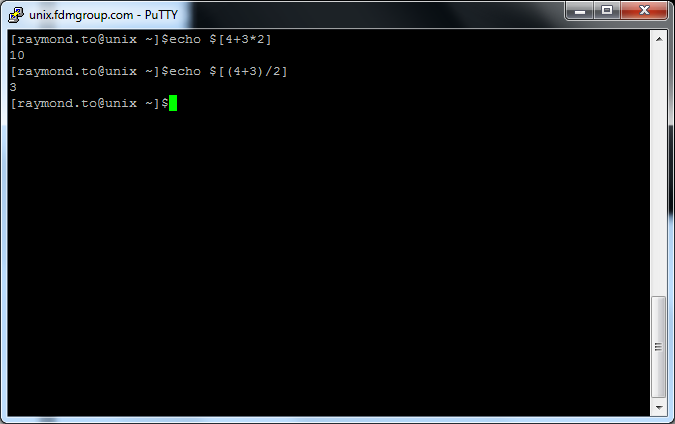
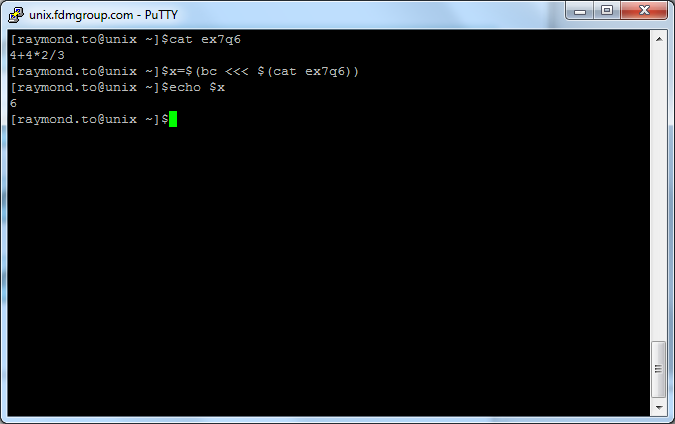
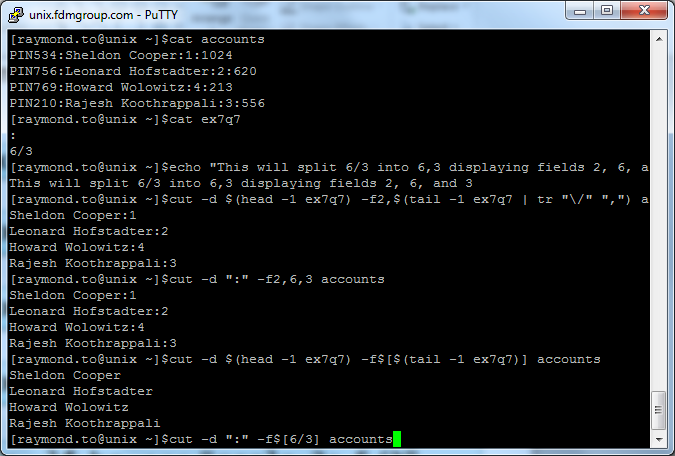
1. Cambridge telephone numbers (11 digits starting in '01223') egrep "^01223[0-9]{6}$" filename
2. Bank sort codes (2 digits, dash, 2 digits, dash, 2 digits) egrep "^[0-9]{2}\-[0-9]{2}\-[0-9]{2}$" filename
3. B-reg number plates ('B', 1-3 digits, 3 letters) egrep "^B[0-9]{1,3}[a-zA-Z]{3}$" filename
4. All lines that do not contain letters egrep "^[^a-zA-Z]\*$" filename

## Exercise 8 - $, $(( ... )), $[ ... ], $( ... )

Perform the following actions on the command line.

1. Set a variable x equal to "Andy". Set a variable y equal to "Bruce". Try the following: echo xy x y $x$y $x $y
2. Set the variable x equal to 4 and the variable y equal to 9. Display the the result of summing the contents of variable x and variable y.
3. Set the variable x equal to the output of the command: ls  
   Display the contents of variable x.
4. Retrieve all the balances from the accounts file and store the result in a variable called "balances". Display variable "balances".
5. Display the result of 4 + 3 \* 2. Display the result of (4+3)/2.
6. Create a file that has some mathematical formula. Send the contents of the file to bc (bench calculator) and store the result in a shell variable.
7. Use vi to create a file with “:” on line 1 and “6/3” on line 2. Using line 1 as the delimiter and line 2 as the field, cut and display the real names from the accounts file.

### Answers Ex 8

1. 
2. 
3. 
4. 
5. 
6. 
7. 

## Exercise 9 - Job Control

|  |
| --- |
| **Perform the following tasks** |
| 1. Run the sleep commands for 1000 seconds in the background   Sleep 1000& |
| 1. Use the ps command to identify the PID of the sleep command   ps |
| 1. Kill the sleep command and check the sleep process was terminated successfully.   kill PIDvalue |
| 1. Start another sleep command for 1000 seconds in the background |
| 1. Use the jobs command to identify the JID of the sleep command   jobs |
| 1. Bring the sleep command into the foreground   fg JID |
| 1. Kill the sleep command and check the process was terminated successfully.   Ctrl+C |

# Shell Scripting

## Exercise 1

|  |
| --- |
| **Introduction to Shell Scripting** |
| 1. Write a script called myVars that displays the following variables $HOME, $LOGNAME and $PATH. |
| 1. Write a script called greeting which displays a welcome message as follows “Hello firstname.lastname”. In your script replace firstname.lastname with your actual UNIX username.   For example if your UNIX username is joe.bloggs, your script should display  **Hello joe.bloggs** |
| 1. Write a script called createFiles that creates a directory called mydir and three files called myfile1, myfile2 and myfile3. (Note this script will generate no output, you will need to check it has worked) |
| 1. Write a script called moveFiles that moves myfile1, myfile2 and myfile3 into mydir. Display a message informing the user that the move operation is complete and then display the contents of the mydir directory.   Your output should be:-  **move operation completed**  **myfile1 myfile2 myfile3** |
| 1. Write a script called lions6 that displays the first 6 lines of /examples/lionsInTheStreet file. |
| 1. Write a script called fileCount that counts how many files you have in your ~ directory. |

### Answers – Exercise 1

## Exercise 2

|  |  |
| --- | --- |
| **Variables and Command Substitution** |  |
| 1. Write a script called enterWord which prompts the user to enter a word. The script should display ‘You entered ………..’   Your script should run as follows:-  **Please enter word**  hello  **You entered hello** | | |
| 1. Write a script called countLetters which prompts the user to enter a word. The script should then display the number of letters in the word.   Your script should run as follows:-  **Please enter word**  hello  **5** | | |
| 1. Modify your previous countLetters script so that it displays the number of letters as part of a sentence: ‘The word …… has ….. letters’ (hint: use command substitution)   Your script should run as follows:-  **Please enter word** hello  **The word hello has 5 letters** | | |
| 1. Write a script called displayArg which takes an argument and then displays ‘The argument is …….’   Your script should be run as follows:-  **displayArg hello**  And would output:-  **The argument is hello**   1. Write a script called charCount which takes an argument and then displays the number of characters in the argument.   Your script should be run as follows:-  **charCount hello**  And would output:-  **5** | | |
| 1. Write a script called nameFile which takes your first name as an argument and then creates a file with your first name as its name. The script should then display the filename and i-node of your new file (hint: use ls to display i-node)   Your script should be run as follows:-  **nameFile joe**  And would output:-  **2099456 joe** where 2099456 is the i-node | | |
| 1. Write a script called renameFile which takes your first and last names as arguments. The script should then rename the file you created in the previous exercise so that its name has the following format: firstname.lastname. Finally the script should display the i-node of this file.   Your script should be run as follows:-  **renameFile joe bloggs**  And would output:-  **2099456 joe.bloggs** | | |

### Answers – Exercise 2

## Exercise 3

|  |  |
| --- | --- |
| **Conditional scripts** |  |
| 1. Write a script called numComp that accepts 2 numbers as arguments and compares the integers. The script will output 1 of the following messages   The first number is greater than the second  The second number is greater than the first  The 2 numbers are equal. | |
| 1. Write a script called stringComp that accepts two strings as arguments. It will compare the length of both strings and output 1 of the following messages   The first string is longer than the second  The second string is longer than the first  Both strings are of equal length | |
| 1. Modify your scripts for previous questions (1 and 2) to check the user supplies 2 arguments.   If the wrong number of arguments are supplied display an error message “Incorrect number of arguments” and exit the script with an error exit status.  In addition if no arguments are supplied display an error message “No argument supplied” and exit the script with an error exit status. | |
| 1. Write a script called delFile that asks a user to enter a file name.   Firstly the script checks the file exists and if doesn’t displays an error message “Invalid filename” and exits the script with an error exit status.  Assuming the file exists the script asks the user if they would like to delete the file.  If they answer y or Y, or anything starting with y it will delete the file.  If they answer n or N or anything starting with n it will not delete the file and exit the script.  Any other response and the script will generate an “Invalid response” message. | |
| 1. Write a menu script called myMenu to enable users to do the following:-   1 – Display the date and time  2 – Display the current working directory  3 – Display the processes being run by the current user  The script should display the menu and prompt the user to select an option (1-3).  If the user selects anything other than 1-3 the script should display an error message and terminate with an error exit status. | |

1. Modify your greeting script (Exercise 1 task 3) to greet you with 1 of the following messages

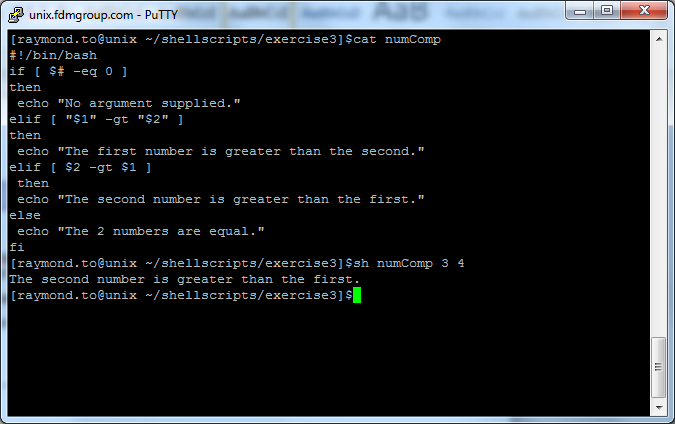
“Good morning firstname.lastname” if the time is before 1pm

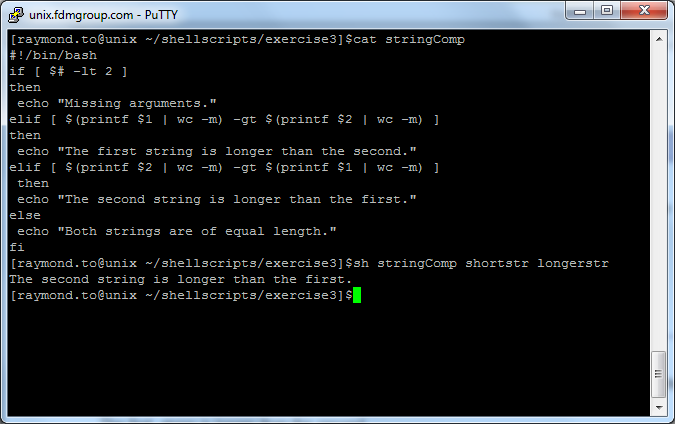
“Good afternoon firstname.lastname” if the time is after 1pm and before 5pm

“Good Evening firstname.lastname” if the test is after 5pm

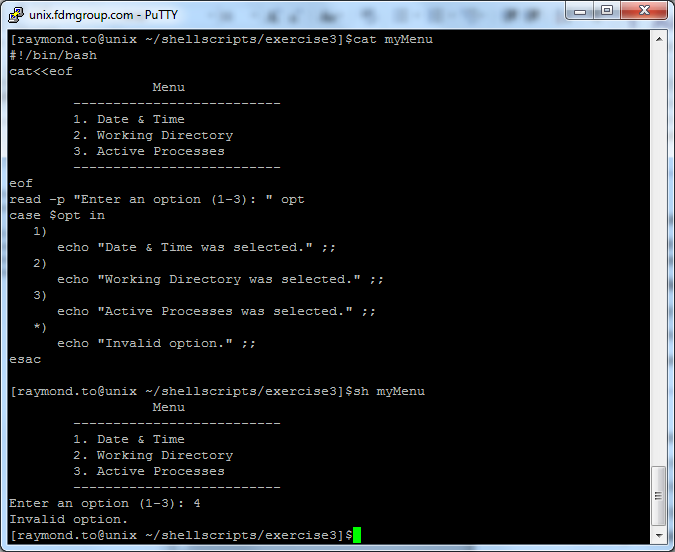
(Hint: Use the manual page to find out more about the date command)

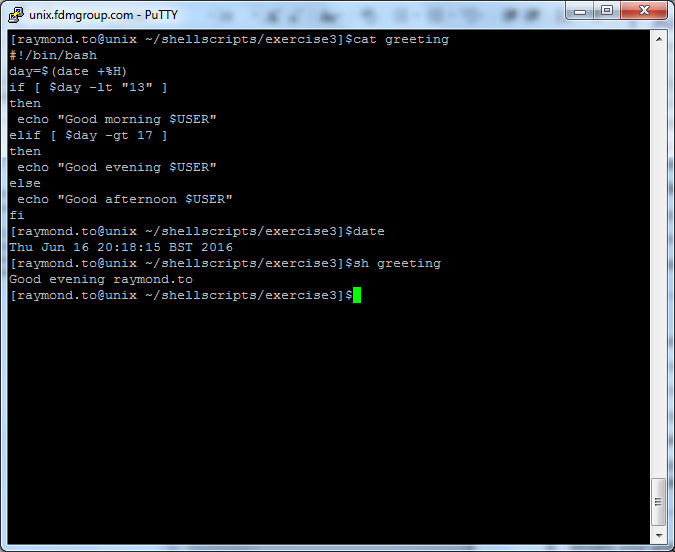
### Answers – Exercise 3











## Exercise 4

|  |  |
| --- | --- |
| **Using Flow control – Loops** |  |

1. Write a script called fileCounter, which accepts any number of filenames as arguments. For each file it will display the file size in characters.

|  |
| --- |
| 1. Modify your fileCounter script to also check each file exists, and if it is doesn’t exist to move onto the next file in the list. |
| 1. Write a script called total which accepts a number as an argument and adds all numbers from 1 to that number and displays the total.   For example if 5 was the argument it would perform 1 + 2 + 3 + 4 + 5 giving an answer of 15. |
| 1. Write a script called lineCounter that accepts a filename as an argument. It should run through the file line by line, displaying a character count for each line. The script should include a line counter, which records the line number and is incremented when each line is processed.   The output should look like:-  Line 1 contains 23 characters  Line 2 contains 12 characters  ….. |
| 1. Modify your myMenu script from the previous exercise (Exercise 3 task 5), to enable users to return to the menu and make another selection, after sucessfully completed an option. In addition add an extra option to allow the user to Quit the menu. |

1. Write a script called letterLoop that will prompt the user to enter a word. It should then use a loop to output each letter in turn. For example, if the user enters "picasso", the output should be:

Letter 1: p

Letter 2: i

Letter 3: c

Letter 4: a

Letter 5: s

Letter 6: s

Letter 7: o

## Passing Arguments

#!/bin/bash

Echo “My name is $1 and I come from $2.”

--

argsExample Ben America

“My name is Ben and I come from America.”

**Positional Parameters - variables**

* **$0** script name
* **$#**  number of command line parameters
* **$\*** contains all command line parameters
* **$@** contains all command line arguments

**Other useful variables**

* **$?** exit status of the most recently executed command
* **$$** - Process ID of the shell
* **$!** - process ID of the most recently executed background command

#!/bin/bash

echo "Script name is : $0"

echo "First argument is : $1"

echo "Second argument is : $2"

echo "Third argument is : $3"

echo "Number of arguments is : $#"

echo "All the command line arguments: $\*"

--

specialVars "Monday" "Tuesday" "Wednesday"

Script name is : specialVars

First argument is : Monday

Second argument is : Tuesday

Third argument is : Wednesday

Number of arguments is : 3

All the command line arguments : Monday Tuesday Wednesday

## User Input

read var – asks user for a value and stores it in var

-p – allows you to enter a string to prompt the user.

#!/bin/bash

echo "please type in value"

read var

echo $var

#!/bin/bash

echo "Enter username : "

read name

echo "Your username is $name"

#!/bin/bash

read -p "Enter username : " name

echo "Your username is $name"

## Arrays

Two ways of creating arrays

#!/bin/bash

names[0]="Bob"

names[1]="Julie"

names[2]="Andrew"

names[3]="Alice"

OR

#!/bin/bash

names=("Bob" "Julie" "Andrew" "Alice")

Access an array (access entire array using \*):

#!/bin/bash

echo ${names[0]} #First value

echo ${names[3]} #Fourth value

echo ${names[\*]} #All values

echo ${#names[\*]} #Number of values

## Arithmetic/Math

bc – bench calculator. Supports decimal values.

result1=$(echo "scale=3; 8/3" | bc)

echo $result1

result2=$(echo "scale=3; (8/3) ^ 3" | bc)

echo $result2

$((mathExpression))

$[mathExpression]

expr mathExpression

echo $((4/2))

echo $[3+2\*5]

result=$[3\*5]

echo $result

num1=7

num2=2

num3=4

echo $[ ($num1-$num2) \* $num3 ]

expr 4 + 7

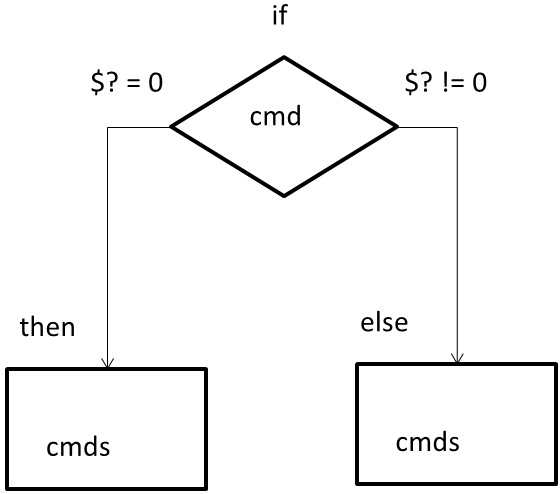
## Exit Status

$? – The status of the previously executed statement.

When $? = 0, statement executed successfully

When $? != 0, statement failed.

## If Statement



#!/bin/bash

if who | grep -q lee

then

echo "Lee has logged into the server."

elif command2

then

code

else

echo "Lee has not logged in yet."

fi

## Testing (If statement)

Square brackets are short hand representation of test command.

Test command simply evaluates the Boolean expressions.

Test command has an alias, “[ ”. The square bracket is used as a shorthand for conditionals and loops

Test expression: [ expression ]

$ test 5 –gt 3

$ echo $?

$ [ 5 –gt 3 ]

$ echo $?

We can perform 3 types of tests

* File test (use man to find more commands)
  + -e name File name exists.
  + -d name File name is a directory.
  + -f name File name is a regular file.
  + -h name File name is a symbolic link.
  + -r name File name exists and is readable.
  + -w name File name exists and is writeable.
  + -x name File name exists and is executable.
  + -s name File name exists and has nonzero size.
* String test (note the spacing)\_
  + s1 = s2 String s1 equals string s2
  + s1 != s2 String s1 does not equal string s2
  + -z s1 String s1 has zero length
  + -n s1 String s1 has nonzero length
* Numeric test
  + n1 –eq n2 Integer n1 and n2 are equal
  + n1 –ne n2 Integer n1 and n2 are not equal
  + n1 –lt n2 Integer n1 is less than n2
  + n1 –le n2 Integer n1 is less than or equal to n2
  + n1 –gt n2 Integer n1 is greater than n2
  + n1 –ge n2 Integer n1 is greater than or equal to n2

### Testing Example

#!/bin/bash

if [ $# -eq 0 ] ; then

echo "No argument has been passed into the script"

elif [ $# -eq 1 ] ; then

echo "Only one argument has been passed into the script"

elif [ $# -eq 2 ] ; then

echo "2 arguments have been passed into the script"

else

echo "More than 2 arguments have been passed into the script"

fi

## Conditional Operations/Operators (If statement)

We can combine more than one expression by using conditional operators.

! t1 Negate operator. Negate the test

t1 II t2 OR operator. Either test t1 or test t2 are true

t1 –o t2 Shell syntax of OR

t1 && t2 AND operator. Both test t1 and test t2 are true

t1 –a t2 Shell syntax of AND

The native shell syntax of AND and OR operators are –a and –o.

&& and || are used for backward compatibility.

Examples

#!/bin/bash

if [[ $# -eq 0 || $# -eq 2 || $# -eq 4 ]] ; then

echo "Number of arguments is even“

elif [[ $# -eq 1 || $# -eq 3 || $# -eq 5 ]] ; then

echo "Number of arguments is odd"

else

echo "More than 5 arguments have been passed into the script"

fi

--

orExample a b

orExample a b c

orExample a b c d e f

test condition1 || test condition2

test condition1 && test condition2

test condition1 -o test condition2

test condition1 -a test condition2

[[ condition1 || condition2 ]]

[[ condition1 && condition2 ]]

[ condition1 ] || [ condition2 ]

[ condition1 ] && [ condition2 ]

[ condition2 –o condition2 ]

[ condition1 –a condition2 ]

## Case (if statement)

Case statement can use wildcard characters to perform pattern matching.

Therefore, case statement is more readable and concise than if statement.

### Syntax:

case value in

pattern1)

code ;;

pattern2)

code ;;

patternN)

code ;;

\*)

code ;;

esac

### Example:

#!/bin/bash

cat<<eof

ATM

-------------------

1. Deposit

2. Withdraw

-------------------

eof

read -p "Enter an option : " opt

case $opt in

1)

echo "Deposit option was selected." ;;

2)

echo "Withdraw option was selected." ;;

\*)

echo "Invalid option." ;;

esac

### With Pattern

#!/bin/bash

read -p "Enter a character: " input

case $input in

[[:alpha:]])

echo "You entered a letter " ;;

[[:digit:]])

echo "You entered a digit " ;;

\*)

echo "Invalid character"

exit 1 ;;

esac