ASSIGNMENTS.md

```
<!-- vim-markdown-toc GFM -->
* [Shell Programming Lab Assignments](#shell-programming-lab-assignments)
* [Mini Project - shell programming](#mini-project---shell-programming)
<!-- vim-markdown-toc -->
## Shell Programming Lab Assignments
1. Write shell script that will add two nos, which are supplied as command line argument, and if this two nos are not given show error and its usage
2. Write Script to find out biggest number from given three nos. Nos are supplied as command line arguments. Print error if sufficient arguments are not supplied.
3. Write script to print given number in reverse order, for eg. If no is 123 it must print as 321
4. Write script to determine whether given file exist or not, file name is supplied as command line argument, Also check for sufficient number of command line arguments.
5. How to write script, that will print, Message ""Hello World"", in Bold and Blink effect, and in different Colors like red, brown etc using echo command."
6. Write a shell script that accepts a file name, starting and ending line numbers as arguments and displays all the lines between the given line numbers.
7. Write a shell script that deletes all lines containing a specified word in one or more files supplied as arguments to it.
8. Write a shell script that displays a list of all the files in the current directory to which the user has read, write and execute permissions.
9. Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file or a directory and reports accordingly. Whenever the argument is a file, the number of lines on it is also reported.
10.Write a shell script to list all of the directory files in a directory.
11.Write a shell script to find factorial of a given integer.
## Mini Project - shell programming
Write a network monitor application server_monitor.sh which show the below information
OS name
Architecture version
Kernel version
Internet status
IP Address
Memory Usage
Disk file systems usage
system uptime
\ No newline at end of file
```

README.md

```
# shell-scripting
<!-- vim-markdown-toc GFM -->
shell scripting class notes & examples
\ No newline at end of file
* [Shell Scripting class notes with examples](#shell-scripting-class-notes-with-examples)
* [1. Shell Scripting Introduction](#1-shell-scripting-introduction)
* [1.1 What is Shell?](#11--what-is-shell)
* [1.2. Shell Script](#12-shell-script)
* [1.3. Hello World Shell Program](#13-hello-world-shell-program)
* [2. Quotations & Substitutions:](#2-quotations--substitutions)
* [3. Shell Parameters:](#3-shell-parameters)
* [4. Shell Conditions: [conditions.sh]](#4-shell-conditions-conditionssh)
* [5. Shell Control Statements:](#5-shell-control-statements)
* [6. Shell ERROR Handling:](#6-shell-error-handling)
* [7. Shell Debugging:](#7-shell-debugging)
* [8. Functions in Shell: [ Example: functions.sh ]](#8-functions-in-shell--example-functionssh-)
* [9. Export user defined environment variables: [ Example: export1.sh, export2.sh ]](#9-export-user-defined-environment-variables--example-export1sh-export2sh-)
* [10. Usage of Shell scripting:](#10-usage-of-shell-scripting)
* [11. Background Process vs foreground Process:](#11-background-process-vs-foreground-process)
* [11.1. What is Process:](#111-what-is-process)
* [11.2. Create a process:](#112-create-a-process)
* [11.3. List of running process:](#113-list-of-running-process)
* [11.4. Process Execution:](#114-process-execution)
<!-- vim-markdown-toc -->
# Shell Scripting class notes with examples
## 1. Shell Scripting Introduction
### 1.1 What is Shell?
Shell is a tool to execute commands.
### 1.2. Shell Script
A shell script is a computer program designed to be run by the Unix shell, a command line interpreter.
### 1.3. Hello World Shell Program [hello.sh]
hello.sh: Hello World Shell Program (echo and read commands).
## 2. Quotations & Substitutions: [quotations.sh]
```

```
quotations.sh: How to use environment variables and quotations in shell.
## 3. Shell Parameters: [parameters.sh]
parameters.sh: shell command line arguments.
$0 The name of the shell script
$# The number of parameters passed
$$ The process ID of the shell script
$1,$2,... The parameters given to the script
$* list of all the parameters, in a single variable, separated by the first character in the environment variable IFS.
$@ A subtle variation on $*; it doesn't use the IFS environment variable, so parameters are not run together even if IFS
is empty
## 4. Shell Conditions: [conditions.sh]
3 types of conditions
String Comparison Result
---------------------
string1 = string2 True if the strings are equal
string1 != string2 True if the strings are not equal
-n string True if the string is not null
-z string True if the string is null (an empty string
Arithmetic Comparison Result
----------------------------
expression1 -eq expression2 True if the expressions are equal
expression1 -ne expression2 True if the expressions are not equal
expression1 -gt expression2 True if expression1 is greater than expression2
expression1 -ge expression2 True if expression1 is greater than or equal to
expression2
expression1 -lt expression2 True if expression1 is less than expression2
expression1 -le expression2 True if expression1 is less than or equal to
expression2
! expression True if the expression is false, and vice versa.
File Conditional Result
-----------------------
-d file True if the file is a directory
-e file True if the file exists. Note that historically the -e option has not been portable, so -f is usually used.
-f file True if the file is a regular file
-g file True if set-group-id is set on file
-r file True if the file is readable
```

-s file True if the file has nonzero size
-u file True if set-user-id is set on file
-w file True if the file is writable
-x file True if the file is executable

5. Shell Control Statements: [control.sh]

control.sh: control statements.

```
while condition
do
statements
done
```

While execute the statements until the condition FAILS.
until execute the statements until the condition SUCCESS.

6. Shell ERROR Handling: [script-error.sh]

script-error.sh: Proper ERROR handling in shell.

7. Shell Debugging: [script7.sh]

script7.sh: debugging in shell

```
$ sh -x ./<script_name>
```

8. Functions in Shell: [Example: functions.sh]

You can define functions in the shell.

To define a shell function, simply write its name followed by empty parentheses and enclose the statements in braces:

```
function_name ()
{
echo $1
statements
}
function_name 12
```

9. Export user defined environment variables: [Example: export1.sh, export2.sh]

By default, variables created in a shell are not available in further (sub)shells invoked from that shell.

The export command creates an environment variable from its parameter that can be seen by other scripts and programs invoked from the current program.

The commands set -a or set -allexport will export all variables thereafter.

10. Usage of Shell scripting:

1. ping.sh: This shell shows Internet status.
2. server-monitor.sh: network monitor application show the below information

```
a OS name
b. Architecture version
c. Kernel version
d. Internet status
e. IP Address
f. Memory Usage
g. Disk file systems usage
h. system uptime
3. km-bbb-kernel-build.sh: Environment Setup, Configure,Compilation of kernel source code.
4. km-bbb-kernel-install.sh: Installation of kernel source code.
## 11. Background Process vs foreground Process:
### 11.1. What is Process:
Process is an execution of a program (or) is an instance of a program.
### 11.2. Create a process:
Command line interface [CLI] or Graphical interface [GUI].
### 11.3. List of running process:
$ ps -eaf
### 11.4. Process Execution:
Foreground process [ Interactive Process ] and Background Process [ Non-Interactive Process / Daemon Process]
Create a process in Foreground:
$ gnome-calculator
Create a process in background [Daemon]
$ gnome-calculator &
List of background process:
$ jobs
Move process from bg -> fg:
$ fg
Move Particular process from bg -> fg:
$ fg <job_id>
<!-- vim-markdown-toc GFM -->
<!-- vim-markdown-toc -->
```

examples/conditions.sh

```
#!/bin/sh
# if control statment example
echo "enter two numbers"
read var1
read var2
if [ $var1 -eq $var2 ]
then
echo "Numbers are equal"
else
echo "Numbers are not equal"
fi
```

examples/control.sh

```
#!/bin/sh
# control statments in shell
#echo $PWD
#cd /home/kishore
#echo $PWD
# usage of if statment
#if [ -w abc ] ; then
# echo "abc file have write permission"
#else
# echo "abc file doen't have write permission"
#fi
#for var1 in 0 1 2 3 4 5
#do
# echo "\$var1:$var1"
#done
#usage of for statement
#for i in *
#do
# if grep -rq abc $i
# then
# echo $i
# fi
```

```
#done
#i=1
#k=5
#while [ $i -le $k ]
#do
# echo $i
# i=$(expr $i + 1)
#done
#usage of while statement
echo "Enter password"
read trythis
while [ "$trythis" != "secret" ]
do
echo "sorry, try again"
read trythis
done
echo "password authentiacation successfully!";
```

examples/error-hand.sh

```
#!/bin/sh
# This shell example shows that how to use Proper ERROR handling in shell
#echo kernel > 123
ping -c 1 www.google > /dev/null 2> /dev/null
#echo "\$?:$?"
if [ $? -eq 0 ]
then
echo "Internet is Working"
else
echo "Internet is not Working"
exit 4
fi
echo "program starts here"
exit 0
#ls xyz
#echo "ls xyz command status: $?"
```

```
#echo kernel > 123
#echo "ls 123 command status: $?"
```

examples/export1.sh

```
#!/bin/sh
#echo "user name:$USER"
# the below command exports all the variables thereafter
# set -a
var1=12
var2=34
./export2.sh
```

examples/export2.sh

```
#!/bin/sh
echo "$var1"
echo "$var2"
```

examples/for.sh

```
#!/bin/sh
# for control statment example
for i in 0 1 2
do
echo $i
done
```

examples/functions.sh

```
#!/bin/sh
# This example shows how to use functions in shell.
# function_name() {
# statements
# }
status()
{
echo "inside status function"
echo $1
echo $2
```

```
}
echo "script starts here"
status 12 13
echo "script ends here"
```

examples/hello.sh

```
#!/bin/sh
# shell command line arguments example script program
echo "read command start here"
read var
echo $var
echo "read command end here"
#echo "No. of arguments: $#"
#echo "Shell script name:$0"
#echo "Shell name:$SHELL"
#echo "1st arg:$1"
#echo "2nd arg:$2"
#echo "All arg:$@"
```

examples/if.sh

```
#!/bin/sh
# if control statment example
echo "enter two numbers"
read var1
read var2
if [ $var1 -eq $var2 ]
then
echo "Numbers are equal"
else
echo "Numbers are not equal"
fi
```

examples/parameters.sh

```
#!/bin/sh
# Shell parameters and command line arguments
IFS='_'
echo "Shell name:\$0=$0"
echo "shell 1st argument:\$1=$1"
echo "shell 2nd argument:\$2=$2"
echo "shell no. of arguemnts:\$#=$#"
echo "shell pid no:\$$=$$"
echo "list all paramters:\$*=$*"
echo "list all paramters:\$@=$@"
```

examples/ping.sh

```
#!/bin/sh
# This shell example shows that how to use Proper ERROR handling in shell
#echo kernel > 123
ping -c 1 www.google > /dev/null 2> /dev/null
#echo "\$?:$?"
if [ $? -eq 0 ]
then
echo "Internet is Working"
else
echo "Internet is not Working"
exit 4
fi
echo "program starts here"
exit 0
#ls xyz
#echo "ls xyz command status: $?"
#echo kernel > 123
#echo "ls 123 command status: $?"
```

examples/quotations.sh

```
#!/bin/bash
#shell variables examples
echo "user name:$USER"
myvar=kernel
echo $myvar
echo "$myvar"
echo '$myvar'
echo \$myvar
echo \$myvar:$myvar
#echo Enter some text
#read myvar
#echo '$myvar' now equals $myvar
exit 0
```

examples/script.sh

```
#!/bin/sh
#This shell shows Internet Status
clear
ping -c 3 www.google.com > /dev/null 2> /dev/null && echo "Internet is Working" || echo "Internet is not Working"
#if ping -c 3 www.google.com > /dev/null 2> /dev/null
#then
# echo "Internet is Working"
#else
# echo "Internet is not Working"
#fi
#ping -c 3 www.google.com > /dev/null 2> /dev/null
#if [ $? != 0 ]
#then
#echo "Internet is not working"
#exit 0
#fi
#echo "Kernel Version:`uname -r`"
#echo "Ubuntu Version:`cat /etc/issue`"
#echo "Host name:`hostname`"
```

examples/script7.sh

```
#!/bin/sh
# comparison of two numbers;
# command usage: ./script7.sh <first_number> <second_number>
echo "Shell Name:\$0:$0"
echo "Shell First Parameter:\$1:$1"
echo "Shell Second Parameter:\$2:$2"
if [ $# -lt 2 ] || [ $# -gt 2 ]
then
echo "Script Usage:$0 <first_number> <second_number>"
exit 1
fi
echo "user pass proper syntax"
if [ "$1" = "$2" ]
then
echo "Both Strings are EQUAL"
elif [ "$1" != "$2" ]; then
echo "Both Strings are NOT EQUAL"
fi
exit 0
```

examples/until.sh

```
#!/bin/sh
until who | grep "$1" > /dev/null
do
sleep 10
done
echo -e \\a
echo “**** $1 has just logged in ****”
exit 0
```