Lab 4, 5 Shell Programming Supplement

1 Exercise 1

1. Make a directory ~/bin in your home directory:

```
$ cd
$ mkdir bin
```

2. Copy the following files from the network directory /home/nfs/ossi-part-time/scripts to your ~/bin directory:

```
$ cp -a /home/nfs/ossi-part-time/scripts/* ~/bin
```

3. Run and understand each of demo files:
   - `var1` (see script 1 on page 3), `var2` (see script 2 on page 3), `var3` (see script 3 on page 4)
   - `if1` (see script 4 on page 4), `if2` (see script 5 on page 4)
   - `arith` (see script 6 on page 5)
   - `loop` (see script 7 on page 5), `loop2` (see script 8 on page 5)
   - `case` (see script 9 on page 6)
   - `file1` (see script 10 on page 6), `file2` (see script 11 on page 7)

2 Exercise 2

1. Write a script which displays “`Good morning`”, “`Good afternoon`” or “`Good evening`”, on the monitor, depending on the time of running the script.

   Have a look at the `date` command: do `man date`. You will see that there are options to output the time in any format; for example, `date +%j` will print what day of the year today is, i.e., from 1 to 366.

   You will probably want to use *command substitution*, i.e., use *backticks*, `‘. . . ‘` that is the single quotes on the left of your keyboard. There is an example on Joe’s slide “shell variable (2), slide 13 of the latest PowerPoint slides on shell scripting, and also in script 2 on page 3, and script 7 on page 5, where it is used with the external `expr` command to do arithmetic, as an alternative to the (easier) built in `let` command.

   Of course, you will also need to use an `if`. . . `elif`. . . `else`. . . `fi` statement; see script 5 on page 4.

3 Exercise 3

Write a script which:

1. prompts to read your name and age:

   ```
   Enter your name: Joe
   Enter your age: 25
   ```

   Use the built in `read` command to read input from the user. Use `echo -n` to print the prompt without a newline at the end. See script 5 on page 4 and script 7 on page 5.

2. Write the read name to the first line of a file named `my_info`.

   Use file redirection (`>`) to do this.

3. Change your age to age $\times 3 - 3$ and write the value to the second line of `my_info`.

   You can use `let` to do the arithemtic. Use file redirection with appending (`>>`) to do this.

4. Content of `my_info`:

   ```
   my name is Joe
   my modified age is 72
   ```

4 Exercise 4

1. Write a script which reads a number in units of seconds and converts it to the units hours:minutes:seconds and prints the result to standard output.

2. Your script must prompt for re-input if a negative value is input

   ```
   Enter number of seconds: 12345
   Result:
   12345 seconds is 3:25:45
   ```

   This is an exercise in arithmetic evaluation. The best bet is to use the built in `let` command, but you could also use the external `eval` command with command substitution (i.e., backticks `‘...‘`). You will also need a loop to prompt for re-input.

5 Exercise 5

1. Write a script `calculate`, which accepts 4 arguments a, b, c, d and prints the value of $a \times 20 - b \times 2 + c \div d$ to standard output.

2. An example of executing the script:

```
$ calculate 2 12 5 2
result of "2*20 - 12*2 + 5/2" is 18
```

This is an exercise in arithmetic evaluation. The best bet is to use the built in `let` command.

A The Shell Script Examples

Script 1 The script `var1` shows how variable settings disappear after a script stops running.

```
#! /bin/sh
# This script shows how changes to variables made by a script
# disappear after execution of the script.

echo "Before re-assignment by this script, environment variable \$HOME is $HOME"
HOME=/root
echo "After re-assignment by this script, environment variable \$HOME is $HOME"
echo "After execution of this script, echo \$HOME"
```

Script 2 `var2` shows how variables are assigned and read.

```
#! /bin/sh
# This script shows how to use variables.
# Variable declaration and assignment:
VAR1=abc
VAR2="this is a variable"
VAR3="today is `date` and VAR1 is $VAR1"
VAR4="\$VAR1"

# display content of variables:
echo "VAR1 is ${VAR1}"
echo "VAR2 is ${VAR2}"
echo "VAR3 is ${VAR3}"
echo "VAR4 is ${VAR4}"
echo "dollar sign is \$, backquote is \`, backslash is \\"
```

Script 3 `var3` shows handling of command line parameters.

```
#!/bin/sh

# This script shows the use of positional parameters
# Please run with 4 parameters, like this:
# var3 first_parameter second third fourth

echo "command name  is $0"
echo "1st parameter is $1"
echo "2nd parameter is $2"
echo "3rd parameter is $3"
echo "4th parameter is $4"
echo "total number of parameters is $#"
```

Script 4 `if1 shows a basic application of an if...else statement.`

```
#!/bin/sh
echo -n "input an integer (1 to 10): "
read INPUT_VAR

if [ $INPUT_VAR -le 5 ]
then
    echo "input less than or equal to 5"
else
    echo "input is greater than 5"
fi
```

Script 5 `if2` shows the use of `if...elif...fi`, and also shows one use of the "`&&`" operator.

```
#!/bin/sh
echo -n "input an integer (1 to 10): "
read INPUT_VAR

if [ $INPUT_VAR -le 2 ]
then
    echo "input less than or equal to 2"
elif [ $INPUT_VAR -lt 4 ] && [ $INPUT_VAR -gt 2 ]
then
    echo "2 < input < 4"
else
    echo "input >= 4 "
fi
```

Script 6 `arith` shows how to use `let` to assign arithmetic experssions to variables.

```
#!/bin/sh
# let works with + - *  /  %
#note that there are no spaces on either the left or right side
#  of +, - , *, / , %, unless you quote the expression.
# let SUM=1 + 2  (this is wrong)
let "SUM = 1 + 2" # Okay, since quoted the spaces
let SUM=1+2 # okay, no spaces, no need for quotes
let SUB=5-1
let MUL=2*5
let DIV=10/3
let REMAINDER=10%3
let 'EQUATION = 1 + 2 - 5 * 4 / 10'

echo "SUM is ${SUM}"
echo "SUB is ${SUB}"
echo "MUL is ${MUL}"
echo "DIV is ${DIV}"
echo "REMAINDER is ${REMAINDER}"
echo "EQUATION  is ${EQUATION}"
```

Script 7 `loop` shows the use of a `while` loop, and also shows the use of the external program `expr` as an alternative to using `let`. Note that `expr` is more portable, but `let` is easier to use.

```
#!/bin/sh
echo "input an integer less than 4"
read VAR
n=0
while [ $VAR -lt 10 ]
do
    VAR=`expr $VAR + 1`
    n=`expr $n + 1`
    echo  "loop $n: VAR is currently equal to $VAR"
done
```

Script 8 `loop2` is the same as script prg:loop1.sh, but uses `let` instead of `expr`.

```
#!/bin/sh
echo "input an integer less than 4"
read VAR
n=0
while [ $VAR -lt 10 ]
do
    let "VAR = VAR + 1"
    let "n = n + 1"
    echo "loop $n: VAR is currently equal to $VAR"
done
```

Script 9 `case` shows how to use the `case` statement.

```
#!/bin/sh
#case control flow

echo -n "input any integer: "
read VAR

case $VAR in
    "1")  echo "input is 1"  ;;
    "2")  echo "input is 2"  ;;
    "3")  echo "input is 3"  ;;
    "4")  echo "input is 4"  ;;
    "5")  echo "input is 5"  ;;
    "6")  echo "input is 6"  ;;
    "7")  echo "input is 7"  ;;
    "8")  echo "input is 8"  ;;
    "9")  echo "input is 9"  ;;
    "10") echo "input is 10" ;;

    *)    echo "input is greater than 10" ;;   #default case
esac
```

Script 10 `file1` shows techniques you can use to write to more than one file at a time.

```
#!/bin/sh

# first create a file (test) for writing
# then appending to this file

# open a file (test) for writing and assign a file descriptor 3 to this file
# file descriptor is used later for identifying  this file
# if this file does not exist, it will be created:
exec  3>test

#redirect all std output of any command to this opened file (test)
echo   "message redirect from std out to file (test)"  1>&3

# close opened file:
exec  3>&-

# open a file (test) for appending and assign a file descriptor 4 to this file
# file descriptor is used later for identifying  this file
exec  4>>test

#redirect all std output of any command to this opened file (test)
echo   "appended message redirect from std out to file (test)"  1>&4

# close opened file:
exec  4>&-
```

Script 11 This script, `file2`, combines the use of `read` with a `while` loop, and some fancy file redirection.

```
#!/bin/sh

#read content of a file line by line and echo read line to monitor

# Open an existed file (test) for reading and
# assign a file descriptor 3 to this file.
# File descriptor is used later for identifying  this file
exec 3<test

LINE_NUM=1

# Redirect all stdin of any command to read from opened file (test) of descriptor 3
while read LINE 0<&3
do
    echo  "Line ${LINE_NUM} of file (test) is: ${LINE}"
    let LINE_NUM=${LINE_NUM}+1
done

# close opened file:
exec  3>&-
```