KIT104 ICT Architecture and Operating Systems
Tutorial Week 11

1. UNIX Shell Scripting – The `helpme.sh`

A	The shell script, shown below, is from a file called `helpme.sh`. When executed the script asks the user to specify a topic (a keyword) on which the help is required. Internally it uses a standard UNIX command to list each manual page that has the specified keyword occurring in it. The command **`man -k`** is an alternative to the command **`apropos`**. Recall that the `apropos` command is used to search all manual pages for the keyword specified. It performs a keyword look-up to locate commands. For example, if you wonder what the command to copy a file is, the following could be typed: `$ apropos "copy file"`

B	
	`#!/bin/sh`
	`echo -e "What do you need help on:\c"`
	`read topic`
	`man -k $topic | tee file1`

C	Login to your account on alacritas. Change into `kit104` directory, and make a new directory named `wk11`. Change into `wk11`. Enter the above script in a file called `helpme.sh`. Use the command `chmod` to assign execute permission to you. Use the command `ls -l` to verify the permissions.
D	To run the script, do this **`$`** `./helpme.sh` While it is run, enter the following keywords, respectively: *chat* *jpeg* *login* *remote* chat Date::Manip::TZ::askamc00 (3pm) - Support for the Asia/Kamchatka time zone Date::Manip::TZ::pachat00 (3pm) - Support for the Pacific/Chatham time zone chat (8) - Automated conversational script with a modem chattr (1) - change file attributes on a Linux file system Also try other keywords (made up by you) on which you might require help.
E	What does the command `tee` do in this script? grep and put into file1 file
F	Experience shows that the list generated for most key topics can be long and unmanageable. Fortunately, `apropos` as well as `man -k` lists each command with a 1-line description of the command. We can refine the list by searching for lines containing (or not containing) secondary keywords and clues.

	So we will try to seek one more keyword to reduce the size of the list. Append the following script to the `helpme.sh` file (add after the last statement). Take note of the use of the `case` construct.
G	

```
echo –e "\n\n Is the list of man pages too long (y/n)? \c"
read YN
case $YN in
[yY]*) echo –e "Please suggest another keyword:\c"
     read topic
     cat file1 | grep $topic | tee file2
     rm file1
     mv file2 file1
     ;;
[nN]*) echo –e "Good. Bye for now.\n"
     ;;
esac
```

H	Make sure that you understand the purpose of each line in this script. Run this script by first entering the keyword *remote* and then the keyword *login*. Enter `y` to answer the question "Is the list of man pages too long?" Run this script again to work on the following input pairs: *jpeg* (first keyword), *compress* (second keyword) *memory* (first keyword), *statistics* (second keyword) *cpu* (first keyword), *time* (second keyword)
I	Next, we will try to repeat the list shortening step two times. If the list is not adequately short by the end of the second step, the user will be asked to remove some man pages based on a keyword that they do NOT want to read about. The script below is a major re-write of the `helpme.sh` file. Create a shell script named `helpme2.sh`, with the following content.
J	

```
#!/bin/sh
echo –e "What do you need help on: \c"
read topic
man -k $topic | tee file1
#
RepeatsLeft="2"
while [ $RepeatsLeft -ne "0" ]
do
    RepeatsLeft=`expr $RepeatsLeft - 1`
    echo –e "\n\n Is the list of man pages too long (y/n)? \c"
    read YN
    #
    case $YN in
    [yY]*) echo –e "Please suggest another keyword: \c"
```

```
            read topic
            cat file1 | grep -i $topic | tee file2
            rm file1
            mv file2 file1
            ;;
        [nN]*) echo -e "Good. Bye for now.\n"
             exit 0
            ;;
        esac
done

echo -e "\nProvide a keyword that you wish to exclude: \c"
read AntiKey
cat file1 | grep -iv $AntiKey > file2
echo
more file2
```

K

Explain the functionality of the following statements in the above code (5 marks each):

`RepeatsLeft=`expr $RepeatsLeft - 1`` cut down the times for loop

`cat file1 | grep -iv $AntiKey > file2`

L

Run the script by entering the following keywords (Enter `y` to answer the question "Is the list of man pages too long?"):

-i, --ignore-case
-v, --invert-match, to select non-matching lines
exclue the unwanted lines and output to file2

file (first keyword)
compress (second keyword)
zip (third keyword)
bzip2 (last keyword)

Have you seen a shorter list after each extra keyword has been entered?

2. Introducing Shell Functions

A	A shell function executes a group of statements enclosed within curly braces. It optionally returns a value with the `return` statement. Unlike in C, a shell function definition uses a null argument list, but requires ( ):
	
	`function_name() {`
	`  statements`
	`  return value` (this is optional)
	`}`
	
	The function can be invoked by its name (*without* the parentheses), optionally followed by its arguments. The value returned is numeric and represents the success or failure of the function.

B	Enter the following shell function named `info` onto your shell (the $ is the shell prompt, the > is automatically generated):
	
	`$ info() {`
	`> echo –e "The current directory is: \c"; pwd`
	`> echo –e "The current users are: \c"; users`
	`> echo –e "Today is `date`"`
	`> }`

C	To run this function simply use the function name. The following is a sample output:
	
	`$ info`
	`The current directory is: /u/staff/sxu/kit104/demo`
	`The current users are: cmcgee dherbert sxu`
	`Today is Fri Aug  28 15:25:33 EST 2016`

D	Shell functions can be created at the command prompt (like above). Shell functions can also be defined at the beginning of a shell script using them or at least preceding the function calls. This is because shell statements are executed in the interpretive mode.
E	The following shell script is from a previous tutorial. It stores the last modified C or Java program (under the current directory) in the variable `file`. It then displays a message. `#! /bin/sh` `if [ $# -eq 1 ] ; then` `  if [ $1 = "j" ] ; then` `  file=`ls -t *.java | head -1``  `  echo –e "The last modified Java file is $file\n"` `  elif [ $1 = "c" ] ; then` `  file=`ls -t *.c | head -1`` `  echo –e "The last modified C file is $file\n"` `  else echo "Invalid file type"` `  fi`

```
else echo –e "Usage: $0 file_type\nValid file types are c
and j"
fi
```

F

We could define two functions within the script to make the procedure clearer:

```
#! /bin/sh

javafile()
{
        file=`ls -t *.java | head -1`
        echo –e "The last modified Java file is $file\n"
}

cfile()
{
        file=`ls -t *.c | head -1`
        echo –e "The last modified C file is $file\n"
}

if [ $# -eq 1 ] ; then
        if [ $1 = "j" ] ; then
                javafile
        elif [ $1 = "c" ] ; then
                cfile
        else echo "Invalid file type"
        fi
else echo -e "Usage: $0 file_type\nValid file types are c
and j"
fi
```

G

Save the above shell script as content into a script named `JorC.sh`. Assign execute permission to you for the file.

Create the following small C program named `hello.c`.

```
#include<stdio.h>
main()
{
        printf("Hello World!\n");
}
```

Copy `hello.c` into `hello2.java`. Run the script `JorC.sh` to test the defined functions:

```
$ ./JorC.sh c
$ ./JorC.sh j
```

(The End)