KIT213 UNIX Shell Scripting Assignment (2023)

Due Date: 3:00pm Monday the 16th of October 2023 - Week 14
Weight: 30% of your KIT213 final result

1. Introduction

This assignment requires you to apply what you have learned from the unit to write scripts to implement tasks specified below. You must complete this assignment individually and ensure the work you submit is your own.

2. Working environment

On the teaching server, **`ictteach.its.utas.edu.au`**, you must make a directory named **`kit213script`** in your home directory and use this directory as your *working directory* for all scripting assignment development. *Assignment scripts must be edited in this directory as your* ***`ictteach`*** *login history will demonstrate your personal ongoing development of the scripts* [1]

1. The first time you start working on the assignment, after logging on, type the following commands (**$** is the prompt, do not type that):

   ```
   $ mkdir kit213script
   $ cd kit213script
   ```

2. Every other time you log on, simply do the following and then continue working:

   ```
   $ cd kit213script
   ```

3. Assessment Process

All student script submissions will be marked on **`ictteach`** using *bash* according to a marking scheme. You will lose considerable marks if your scripts do not run correctly on our teaching server, so it is imperative (and required) that you develop and test your scripts on **`ictteach`** before you submit.

---

[1] The sudden appearance of a nearly-complete assignment script in your personal ictteach directory without any prior evidence of it being developed there is a possible sign of academic misconduct (plagiarism).

4. Scripting Tasks

Task A

Write a shell script (to run on the Bourne shell) that can be used to remove some old C programs you no longer wish to keep. When the script is run with no arguments supplied, it picks up each C program from the current directory and lists the first 10 lines (hint: research the `head` command). It then prompts for deletion of the file (the user can then choose to delete or not to delete the file). When the user supplies arguments (C program file names) with the script, it works on those files only.

Your script for this task must be named **`delC.sh`**. In designing your script you should consider the following scenarios:

- There is no C program file stored under the current directory;
- There is at least one C program file stored under the current directory;
- The user-supplied arguments list contains names of existing C programs stored under the current directory. We can assume that all the C programs have been correctly named as *.c, and that there is no special character (such as space) in any of these filenames;
- The user-supplied arguments list contains both existing and missing names of C programs stored under the current directory. Missing names refer to the files which no longer exist under the current directory.

Make sure your script is user-friendly and follows common sense (for example, when there is no C program stored under the current directory your script should display a message and then exit). The following is a sample output of the script. The $ is the shell prompt.

```
$ ./delC.sh
This script removes C files which you no longer want to keep.
Here are the C file(s) under the current directory:
No C files found.

$ ./delC.sh
This script removes C files which you no longer want to keep.
Here are the C file(s) under the current directory:
f1.c  f2.c

Displaying first 10 lines of f1.c:

/* A C program for handling arrays
   Written by John Jones, July 2009
   Updated April 2010
*/

#include <stdio.h>
```

```
int main()
{
  int x;

Delete file f1.c? (y/n):n (user input)
File f1.c NOT deleted.

Displaying first 10 lines of f2.c:

/* Another C program for handling arrays
   Written by John Jones, August 2009
   Updated May 2010
*/

#include <stdio.h>

int main()
{
  int x, y, z;

Delete file f2.c? (y/n): y (user input)
File f2.c deleted.
```

$ ./delC.sh f1.c

```
This script removes C files which you no longer want to keep.
The file(s) you want to delete is/are:
f1.c

Displaying first 10 lines of f1.c:

/* A C program for handling arrays
   Written by John Jones, July 2009
   Updated April 2010
*/

#include <stdio.h>

int main()
{
  int x;

Delete file f1.c? (y/n):y (user input)
File f1.c deleted.
```

$ ./delC.sh f2.c f3.c

```
This script removes C files which you no longer want to keep.
The file(s) you want to delete is/are:
f2.c f3.c
```

```
Displaying first 10 lines of f2.c:

/* Another C program for handling arrays
   Written by John Jones, August 2009
   Updated May 2010
*/

#include <stdio.h>

int main()
{
  int x, y, z;

Delete file f2.c? (y/n): y (user input)
File f2.c deleted.

Displaying first 10 lines of f3.c:

File f3.c does not exist.

$ ./delC.sh f3.c f4.c
This script removes C files which you no longer want to keep.
The file(s) you want to delete is/are:
f3.c f4.c

Displaying first 10 lines of f3.c:

File f3.c does not exist.

Displaying first 10 lines of f4.c:

File f4.c does not exist.

$
```

(Continued next page)

Task B

Write a shell script (to run on the Bourne shell) that runs an infinite loop to monitor the creation and removal of `.pdf` or `.PDF` files under the current directory. Every 3 seconds it should display a list of those filenames created or removed after the previous display. Without loss of practical significance of this utility, we can assume that the time interval between creation of a `.PDF` or `.pdf` file and removal of a `.PDF` or `.pdf` file is over 3 seconds, which means that it's unnecessary for your script to handle the situation where creation of a file is followed by immediate removal of a file. (Hint: research the `cmp` and `comm` commands.)

Your script for this task must be named **`pdf.sh`**. The following is a sample output of the script (It is OK that the script leaves behind a temporary file when it is finally interrupted). The $ is the shell prompt.

```
$ ./pdf.sh

No pdf files have been created or removed in the last 3 seconds.

No pdf files have been created or removed in the last 3 seconds.

The following pdf file(s) have been created in the last 3 seconds:
a3.pdf
a4.pdf
a5.pdf

The following pdf file(s) have been removed in the last 3 seconds:
a4.pdf
a5.pdf

No pdf files have been created or removed in the last 3 seconds.

No pdf files have been created or removed in the last 3 seconds.

… … … … …
```

(Continued next page)

Task C

Write a shell script (to run on the Bourne shell) that allows a user to view, add, or delete a setting in a configuration file (`config.txt`) that contains settings in the form `variable=value`. The following is an example of such configuration file:

```
HOME=/u/soc/abc
HOST=lawson
HOSTTYPE=sun4
LOGNAME=abc
OSTYPE=solaris
PATH=/usr/dt/bin:/usr/openwin/bin:/bin:.
PS1=$
PS2=>
SHELL=/usr/bin/tcsh
TZ=Australia/Tasmania
USER=abc
VENDOR=sun
EDITOR=joe
```

Your script for this task must be named **`setting.sh`**. For ease of use, your script must present a menu of operations that a user may choose from. After the user makes a selection and that the selected operation has been completed, the menu must be displayed again so that the user can make another selection. Validation check on user inputs is required (see the following sample output about this). In the beginning of your script you need to check to see whether the required configuration file (`config.txt`) actually exists under the current directory (if not, your script displays a message and then exits).

Here is a sample output of your script. The $ is the shell prompt. The items in italics are not part of the sample output. They are hints indicating how your script should behave.

```
$ ./setting.sh

*** MENU ***
1. Add a Setting
2. Delete a Setting
3. View a Setting
4. View All Settings
Q - Quit

CHOICE: 1 (user input)
Enter setting (format: ABCD=abcd): (user simply presses the Enter/Return key)
New setting not entered

Enter setting (format: ABCD=abcd): EDITOR (user input)
Invalid setting (A valid setting needs to contain a "=" sign)
```

```
Enter setting (format: ABCD=abcd): EDITOR= (user input)
The variable name of the setting is: EDITOR
The variable value of the setting is:
Invalid setting.
```

(Hint: To retrieve a variable name before the "=" sign, research the `expr` command's ability in handling strings)

```
Enter setting (format: ABCD=abcd): =vi (user input)
The variable name of the setting is:
The variable value of the setting is: vi
Invalid setting.

Enter setting (format: ABCD=abcd): 1EDITOR=vi (user input)
The variable name of the setting is: 1EDITOR
The variable value of the setting is: vi
Invalid setting. The first character of a variable name cannot be a digit.

Enter setting (format: ABCD=abcd): EDITOR=vi (user input)
The variable name of the setting is: EDITOR
The variable value of the setting is: vi

New setting added.

*** MENU ***
1. Add a Setting
2. Delete a Setting
3. View a Setting
4. View All Settings
Q - Quit
CHOICE: 1 (user input)

Enter setting (format: ABCD=abcd): USER=jchen (user input)
The variable name of the setting is: USER
The variable value of the setting is: jchen
Variable exists. Changing the values of existing variables is not allowed.

*** MENU ***
1. Add a Setting
2. Delete a Setting
3. View a Setting
4. View All Settings
Q - Quit
CHOICE: 2 (user input)

Enter variable name: EDTOR (user input)
Variable does not exist. (Your script needs to check whether a variable exists or not)
```

```
*** MENU ***
1. Add a Setting
2. Delete a Setting
3. View a Setting
4. View All Settings
Q - Quit
CHOICE: 2 (user input)

Enter variable name: EDITOR (user input)
EDITOR=vi
Delete this setting (y/n)? y (user input)
Setting deleted (However, if user's answer is n here, then the setting stays)

*** MENU ***
1. Add a Setting
2. Delete a Setting
3. View a Setting
4. View All Settings
Q - Quit
CHOICE: 3

Enter variable name:USER1 (user input)
Variable does not exist. (Your script needs to check whether a variable exists or not)

*** MENU ***
1. Add a Setting
2. Delete a Setting
3. View a Setting
4. View All Settings
Q - Quit
CHOICE: 3 (user input)

Enter variable name: USER (user input)
USER=abc
Requested setting displayed above.

*** MENU ***
1. Add a Setting
2. Delete a Setting
3. View a Setting
4. View All Settings
Q - Quit
CHOICE: 4 (user input)

HOME=/u/soc/abc
HOST=lawson
HOSTTYPE=sun4
LOGNAME=abc
OSTYPE=solaris
```

```
PATH=/usr/dt/bin:/usr/openwin/bin:/bin:.
PS1=$
PS2=>
SHELL=/usr/bin/tcsh
TZ=Australia/Tasmania
USER=abc
VENDOR=sun

*** MENU ***
1. Add a Setting
2. Delete a Setting
3. View a Setting
4. View All Settings
Q - Quit
CHOICE: 5 (user input)
Invalid choice.

*** MENU ***
1. Add a Setting
2. Delete a Setting
3. View a Setting
4. View All Settings
Q - quit
CHOICE: q (user input)
(The running script is terminated. The shell prompt is displayed)
```

(Continued next page)

For All Your Scripts

You must

- Include **your name**, **student ID**, and a **brief introduction** of what the script does in all your shell scripts, as a comment in the beginning of each script.
- Make your scripts run on the Bourne shell, regardless of which shell the user of your scripts is currently on.
- Add in-line comments to help other people understand your scripts.
- Use "\n" where appropriate to make the output of your scripts more readable.
- Note that your script structure and layout are also important as they will be marked as part of the assessment process.

5. Submitting Your Assignment

> - The instructions that follow assume you have created a directory called **kit213script** in your **ictteach** home directory, and that your script files (`delC.sh, pdf.sh, setting.sh, config.txt`) are located inside the **kit213script** directory.
> - These instructions also assume you are either using a lab macOS-based computer, or you are using your personal computer. If you are off-campus, you must be running GlobalProtect VPN to be able to connect to **ictteach** via SSH and SCP.

All assignment submissions are through MyLO. Submitting your assignment to MyLO requires you to first make a compressed copy of your script files on **ictteach**, copy that compressed file from **ictteach** to your local computer, and then upload the compressed file from your local computer to the MyLO submission area. Please read and follow the instructions below carefully – ask your lecturer if you have any difficulty, and *we suggest you try the instructions well-ahead of the due date so you do not encounter any last minute problems*.

5.1 Start on ictteach

You must first create a compressed version (a copy) of your assignment scripts on **ictteach**, and then copy this version to your local computer (MyLO does not permit .sh files to be directly submitted):

On **ictteach**, run the following commands (**$** is the prompt):

```
$ cd ~/kit213script
$ tar cvf kit213assignment.tar delC.sh pdf.sh setting.sh config.txt
$ gzip -c kit213assignment.tar > kit213assignment.tar.gz
```

You then need to copy the **kit213assignment.tar.gz** file to your local computer – instructions on how to do this differ depending on what local operating system your local

computer is using – see below. The instructions for each operating system use a command called **scp** (secure copy), which uses the SSH protocol behind the scenes to copy files.

5.2 Apple macOS (using Terminal)

If you are using an Apple macOS-based computer to submit, copy your compressed script file to the local computer via:

a) From a new terminal window (i.e. not a terminal session already connected to **ictteach**), type the following (**%** is the prompt, do not type that)

```
% scp  username@ictteach.its.utas.edu.au:kit213script/kit213assignment.tar.gz  ~/Desktop
```

(*replace* ***username*** *with your own UTAS username (this is NOT your email address!)*)

b) You should then be prompted for your **ictteach** password:

```
username@ictteach.its.utas.edu.au's password:
```

c) After typing your password (it will not be shown onscreen), if you have correctly authenticated, you should get a notification line like the following to indicate the file has been downloaded (to your Desktop in this instance – the file size and data transfer rate shown will differ to your file size and data transfer rate)

```
kit213assignment.tar.gz          100% 3727     2.6MB/s   00:00
```

5.3 Microsoft Windows (using putty)

If you are using your own Microsoft Windows-based computer to submit, we assume you have previously installed the **putty** program to access **ictteach**. Putty also includes some other programs when you install it – **pscp.exe** is one of them and it is the program needed. If you are not using putty but have been using some other access method, you will need to investigate how to use the **scp** command for that method.

Copy your compressed script file to the local computer via:

a) Start a new *command prompt* window (select the Windows start menu icon and then type **cmd.exe**).

b) In the command prompt window, use the following command to navigate to the putty local installation directory (in the example, putty is installed in a putty subdirectory of the standard location, **c:\Program Files** – your local putty directory may differ). (**C:\Users\username>** is the prompt, do not type that):

```
Microsoft Windows [Version 10.0.19042.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\username> cd c:\program files\putty
```

c) If successful, the current directory should now be the putty install directory. **pscp.exe** is putty's version of the scp command, and it should be installed here. Run the following command (**C:\Program files\PuTTY>** is the prompt, do not type that):

```
C:\Program files\PuTTY> .\pscp.exe
username@ictteach.its.utas.edu.au:kit213script/kit213assignment.tar.gz
"%userprofile%"
```

(*Replace* **username** *with your UTAS username (this is NOT your email address))*

> ➢ Note that there are only two spaces in the above command – between **pscp.exe** and your ***username***, and between **.gz** and "**%userprofile%**". All parts are on the same line, it just appears word-wrapped over lines in this document.

d) You will be prompted for your **ictteach** password (the password will not appear on screen):

```
username@ictteach.its.utas.edu.au's password:
```

If you have typed the commands correctly, your compressed script file, **kit213assignment.tar.gz** should now be in your Windows profile home location (usually c:\users**username**) but your path may differ.

5.4 MyLO submission (all students):

If you have successfully copied your compressed **kit213assignment.tar.gz** script file from **ictteach** to your local computer, it should now be on your local system. You can now submit this file to MyLO as usual:

a) Go to KIT213 MyLO site.
b) Navigate through the MyLO site's top menu to the submission link:
Assessments → Assignments → Unix Shell Scripting
c) Scroll down and select the **Add a File** button.
d) Chose the **My Computer** option, and then choose the **Upload** button. Navigate to your local location and select the **kit213assignment.tar.gz** file that you copied earlier and finally choose **Open**.
e) Back in the **Add a File** dialog window, choose the **Add** button at the bottom.
f) Then, if everything appears to be ok (verify you can see your file listed) choose the **Submit** button.

5.4.1 Revised submission?

If it is before the due date and time, follow all the submission steps again to make a compressed copy of your scripts, download the copy to your computer and then upload your copy to MyLO. We will only assess the latest submission.

5.4.2 Late submission?

If your assignment is late then you should submit your compressed script file to MyLO as above – the University's assessment policy means you will lose 5% of your score for every day the submission is late – assignments will not be accepted after 10 days past the scheduled submission date.

5.4.3 Need Help?

You are encouraged to seek assistance from your lecturer by email or during consultation **after you have seriously thought about the assignment**. Please note that we can provide general advice – but you are expected to write and debug (correct) your own code.

When writing your scripts, think about what steps you need to do, don't try to write the code all at once. **You should adopt the incremental development approach**: Implement a small part first, test that it is working correctly, then implement a bit more and test the added bit – if there are problems then the problems must come from the added bit. Repeating the process of thinking, writing (coding), testing and refining/fixing, **one bit at a time**. Do not leave the development of your script too late – you will run out of time!

6. Plagiarism

Plagiarism is a very serious matter, and ignorance of this is not an excuse. If you submit work claiming it to be your own, and it is not original work of your own, this is cheating. This means for example that you cannot submit sections of code that have been written by other students or sourced from the Internet ***or other services*** and claim you wrote the code yourself. Plagiarism can result in academic penalties both in relation to your assignment, and also on your permanent university record. Academic misconduct also applies to someone who shares their code with others.

As an example, you cannot take code written by someone else and change variable names, comments and spacing to make it appear you wrote the code – this would still be considered plagiarism.

(Go to next page to see the marking scheme)

KIT213 Unix Shell Scripting Assignment Marking Scheme

Script `delC.sh` (for each item there are only 3 possible marks: 100% or 50% or 0%)

	Mark	Out of
Execution of the shell script Script runs as expected (**4**). Script runs but not as expected (**2**). Script does not run (**0**)		**4**
Display a message then exit when there is no C file under current directory		**4**
Pick up each C file from current directory		**4**
List the first 10 lines then prompt for deletion		**4**
When user supplies C file names as arguments work on those files only		**4**
Can handle invalid file names contained in user-supplied arguments list		**4**

Script `pdf.sh` (for each item there are only 3 possible marks: 100% or 50% or 0%)

	Mark	Out of
Execution of the shell script Script runs as expected (**4**). Script runs but not as expected (**2**). Script does not run (**0**)		**4**
Correctly set up infinite loop		**4**
Display every 3 seconds "No pdf files have been created or removed … …" when this is true		**4**
Correctly display names of the files that have been created		**4**
Correctly display names of the files that have been removed		**4**

Script `setting.sh` (for each item there are only 3 possible marks: 100% or 50% or 0%)

	Mark	Out of
Execution of the shell script Script runs as expected (**4**). Script runs but not as expected (**2**). Script does not run (**0**)		**4**
Correctly allow user to remove settings		**4**
Correctly allow user to view a setting		**4**
Correctly allow user to view all settings		**4**
Checking the existence of the configuration file		**4**
Checking the existence of the variable name for deleting or viewing a setting		**4**
Correctly allow user to add new settings		**4**
Add new settings - Changing the values of existing variables not allowed		**4**
Add new settings - verify user input contains a "=" sign		**4**
Add new settings - retrieve variable name and variable value of a new setting		**4**
Add new settings – first character of a variable name cannot be a digit		**4**

Others (for each item there are only 3 possible marks: 100% or 50% or 0%)

	Mark	Out of
Shell scripts structure and layout Clear and tidy (**4**). Somewhat messy but understandable (**2**). Messy (**0**)		**4**
Appropriate comments		**4**
Include name, ID, and brief introduction in all scripts		**4**

Assignment Total: /100