Piscine 101

#00

Summary: This document is the subject for the SHELL module 00 of the Piscine 101 @ 42Tokyo.

Contents

Chapter I

Instructions

- Only this page will serve as reference; do not trust rumors.

- Watch out! This document could potentially change up to an hour before submission.

- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We `will not` take into account a successfully completed harder exercise if an easier one is not perfectly functional.

- Make sure you have the appropriate permissions on your files and directories.

- You have to follow the submission procedures for every exercise.

- Your exercises will be checked and graded by your fellow classmates.

- On top of that, your exercises will be checked and graded by a program called Moulinette.

- Moulinette is very meticulous and strict in its evaluation of your work. It is entirely automated and there is no way to negotiate with it. So if you want to avoid bad surprises, be as thorough as possible.

- Exercises in Shell must be executable with /bin/sh.

- You cannot leave any additional file in your directory than those specified in the subject.

- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.

- Your reference guide is called `Google / man / the Internet / ....`

- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...

Chapter II

Foreword

A shell script is a computer program designed to be run by the Unix shell,
a command-line interpreter.
[1] The various dialects of shell scripts
are considered to be scripting languages.
Typical operations performed
by shell scripts include file manipulation,
program execution, and printing text.
A script which sets up the environment,
runs the program, and does any necessary cleanup,
logging, etc. is called a wrapper.

The term is also used more generally to mean
the automated mode of running an operating system shell;
in specific operating systems they are called
other things such as batch files (MSDos-Win95 stream, OS/2),
command procedures (VMS), and shell scripts
(Windows NT stream and third-party derivatives
like 4NT-article is at cmd.exe),
and mainframe operating systems are
associated with a number of terms.

The typical Unix/Linux/POSIX-compliant
installation includes the KornShell (ksh)
in several possible versions such as ksh88,
Korn Shell '93 and others.
The oldest shell still in common use is the Bourne shell (sh);
Unix systems invariably also include the C shell (csh),
Bash (bash), a Remote Shell (rsh),
a Secure Shell (ssh) for SSL telnet connections,
and a shell which is a main component of the Tcl/Tk
installation usually called tclsh; wish is a GUI-based Tcl/Tk shell.
The C and Tcl shells have syntax quite similar
to that of said programming languages,
and the Korn shells and Bash are
developments of the Bourne shell,

which is based on the ALGOL language with
elements of a number of others added as well.

[2] On the other hand, the various shells
plus tools like awk, sed, grep, and BASIC, Lisp,
C and so forth contributed to the
Perl programming language.[3]

Other shells available on a machine or available
for download and/or purchase include Almquist shell (ash),
PowerShell (msh), Z shell (zsh, a particularly common enhanced KornShell),
the Tenex C Shell (tcsh), a Perl-like shell (psh). Related programs
such as shells based on Python, Ruby, C, Java, Perl, Pascal,
Rexx &c in various forms are also widely available.
Another somewhat common shell is osh,
whose manual page states it "is an enhanced,
backward-compatible port of the standard command interpreter
from Sixth Edition UNIX."[4]

Windows-Unix interoperability software such as the MKS Toolkit,
Cygwin, UWIN, Interix and others make the above shells
and Unix programming available on Windows systems,
providing functionality all the way down to signals
and other inter-process communication,
system calls and APIs.
The Hamilton C shell is a Windows shell that is
very similar to the Unix C Shell.
Microsoft distributed Windows Services for UNIX for use with
its NT-based operating systems in particular,
which have a POSIX environmental subsystem.

https://en.wikipedia.org/wiki/Shell_script

Chapter III

Exercise 00 : 42

	Exercise 00	
	Only the best know how to create a file	
Turn-in directory : *ex*00/		
Files to turn in : `42`		
Allowed functions : `None`		

- Create a file called `42`

```
?>ls -l
42
?>
```

https://bit.ly/3an5uiL

https://bit.ly/3pmfsqL

https://bit.ly/3qYR0fh

Chapter IV

Exercise 01 : Hello 42!

	Exercise 01
	Only the best know how to edit a file
Turn-in directory : *ex01/*	
Files to turn in : `42`	
Allowed functions : `None`	

- Create a file called `42`. The file should contain "Hello 42!" with a newline at the end.

```
?>cat 42
Hello 42!
?>
```

https://bit.ly/3qZX1IV

Chapter V

Exercise 02 : GiT commit

	Exercise 02
	GiT commit?
Turn-in directory : *ex02/*	
Files to turn in : `git_commit.sh`	
Allowed functions : `None`	

- Create a shell script that displays the ids of the last 5 commits of your git repository.

```
%> bash git_commit.sh | cat -e
baa23b54f0adb7bf42623d6d0a6ed4587e11412a$
2f52d74b1387fa80eea844969e8dc5483b531ac1$
905f53d98656771334f53f59bb984fc29774701f$
5ddc8474f4f15b3fcb72d08fcb333e19c3a27078$
e94d0b448c03ec633f16d84d63beaef9ae7e7be8$
%>
```

To test your script, we will use our own environment.

> https://bit.ly/3pnst3o

> w https://bit.ly/2MuYWG9

> https://bit.ly/3cdOspz

https://bit.ly/3a9zM8b

Chapter VI

Exercise 03 : clean

	Exercise 03
Turn-in directory : *ex03/*	
Files to turn in : `clean`	
Allowed functions : `None`	

- In a file called `clean` place the command line that will search for all files - in the current directory as well as in its sub-directories - with a name ending by ~, or a name that start and end by `#`

- The command line will show and erase all files found.

- Only one command is allowed: no ';' or '&&' or other shenanigans.

`https://bit.ly/3aaGI5d`

Chapter VII

Exercise 04 : count_files

	Exercise 04
	count_files.sh
Turn-in directory : *ex04/*	
Files to turn in : `count_files.sh`	
Allowed functions : `None`	

- Write a command line that counts and displays the number of regular files and directories in the current directory and all its sub-directories. It should include ".", the starting directory.

- Example of output :

```
$> bash count_files.sh | cat -e
42$
$>
```

Chapter VIII

Exercise 05 : MAC

	Exercise 05
	MAC.sh
Turn-in directory : *ex05/*	
Files to turn in : `MAC.sh`	
Allowed functions : `None`	

- Write a command line that displays your machine's MAC addresses. Each address must be followed by a line break.

https://bit.ly/2M7T6eb

Chapter IX

Exercise 06 : Find string

	Exercise 06
	find_string.sh
Turn-in directory : *ex06/*	
Files to turn in : `find_string.sh`	
Allowed functions : `None`	

- Write a shell script that displays the filename which contain the string given from the commandline argument in the current directory and all its sub-directories.

- filename should only appear once.

- use `sort` command to sort the output in alphabetic order.

- If there is no command line argument or more than 1 argument, do nothing.

Example:

```
%> bash find_string.sh "test" | cat -e
./test.txt$
./test2.txt$
./subject/test3.txt$
%>
%> bash find_string.sh | cat -e
%>
%> bash find_string.sh "test" "test2" | cat -e
%>
```

`https://bit.ly/3ogt99i`

Chapter X

Exercise 07 : FizzBuzz

	Exercise 07
	FizzBuzz
Turn-in directory : *ex07/*	
Files to turn in : `fizzbuzz.sh`	
Allowed functions : `None`	

- Write a shell script that prints the numbers from 1 to the number given from command line argument, each separated by a newline.

- If the number is a multiple of 3, it prints 'fizz' instead.

- If the number is a multiple of 5, it prints 'buzz' instead.

- If the number is both a multiple of 3 and a multiple of 5, it prints 'fizzbuzz' instead.

- If there is no command line argument or more than one command line argument, do nothing.

- If there is one command line argument, the value will always be between 1 and 100.

  Example:

```
%> bash fizzbuzz.sh 16 | cat -e
1$
2$
fizz$
4$
buzz$
fizz$
7$
8$
fizz$
buzz$
11$
fizz$
13$
14$
fizzbuzz$
16$
%>
%> bash fizzbuzz.sh | cat -e
```

```
%>
%> bash fizzbuzz.sh 16 100 | cat -e
%>
```

https://bit.ly/39kY93A

https://bit.ly/2MtIbew

Chapter XI

Exercise 08 : How many Chars?

	Exercise 08
	How many Chars?
Turn-in directory : *ex08/*	
Files to turn in : `chars.sh`	
Allowed functions : `None`	

- Write a shell script that prints the filename and the numbers of character inside that specific file.

- filename are given from command line argument.

- If the file given from command line argument doesn't exist, it should print `No such file or directory` after the filename.

- Check the example and follow the same format for the output of the command.

  Example:

```
%> bash chars.sh chars.sh nosuchfile /etc/passwd | cat -e
chars.sh:139$
nosuchfile:No such file$
/etc/passwd:6804$
%>
```

https://bit.ly/3cmFl5U

Chapter XII

Exercise 09 : Create your own command

	Exercise 09
	Create your own command

Turn-in directory : *ex09/*
Files to turn in : `file_viewer`
Allowed functions : `None`

- Write a shell script that prints the file's content from a specified line to another specified line.

- Check the example and follow the same format for the output of the command.

- Add a shebang inside the file.

  Example:

```
%> ./file_viewer --help | cat -e
file_viewer$
Display specific part of the file.$
$
Usage:$
    file_viewer.sh [-h] [-f filename] start end$
$
Optional arguments:$
    -h, --help          show this help message and exit$
$
Mandatory arguments:$
    -f, --filename      specify the file$
    start               starting line$
    end                 ending line$
%>
%> ./file_viewer -f /etc/passwd 11 13 | cat -e
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false$
root:*:0:0:System Administrator:/var/root:/bin/sh$
daemon:*:1:1:System Services:/var/root:/usr/bin/false$
%>
%> ./file_viewer --filename /etc/passwd 11 13 | cat -e
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false$
root:*:0:0:System Administrator:/var/root:/bin/sh$
daemon:*:1:1:System Services:/var/root:/usr/bin/false$
%>
```

```
%> ./file_viewer -f /etc/passwd 11 a | cat -e
file_viewer: illegal offset -- a$
%>
%> ./file_viewer -f /etc/passwd a 13 | cat -e
file_viewer: illegal offset -- a$
%>
%> ./file_viewer -f /etc/passwd | cat -e
file_viewer$
Display specific part of the file.$
$
Usage:$
    file_viewer.sh [-h] [-f filename] start end$
$
Optional arguments:$
    -h, --help            show this help message and exit$
$
Mandatory arguments:$
    -f, --filename        specify the file$
    start                 starting line$
    end                   ending line$
%>
%> bash file_viewer.sh -f nosuchfile 11 13 | cat -e
file_viewer: nosuchfile: No such file$
%>
%> bash file_viewer.sh -f /etc/passwd 1000 1001
%>
%> bash file_viewer.sh -f /etc/passwd 10 1
%>
%> bash file_viewer.sh | cat -e
file_viewer$
Display specific part of the file.$
$
Usage:$
    file_viewer.sh [-h] [-f filename] start end$
$
Optional arguments:$
    -h, --help            show this help message and exit$
$
Mandatory arguments:$
    -f, --filename        specify the file$
    start                 starting line$
    end                   ending line$
%>
```

https://bit.ly/2YkiOPg

17

Chapter XIII

Exercise 10 : Create a manual for your own command

	Exercise 10
	Create a manual for your own command
Turn-in directory : *ex10/*	
Files to turn in : `man_file_viewer`	
Allowed functions : `None`	

- Create a man page for file_viewer.

- Man section should be 1.

- Man should print out these sections: NAME, SYNOPSIS, DESCRIPTION, AU-THOR

  `man man`