

Piscine 101
Shell 00

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

Contents

1	THSU uctions	4
II	Foreword	3
III	Exercise 00: 42	5
IV	Exercise 01 : Hello 42!	6
V	Exercise 02 : GiT commit	7
VI	Exercise 03 : clean	9
VII	Exercise 04 : count_files	10
VIII	Exercise 05: MAC	11
IX	Exercise 06 : Find string	12
X	Exercise 07 : FizzBuzz	13
XI	Exercise 08: How many Chars?	15
XII	Exercise 09: Create your own command	16
XIII	Exercise 10: Create a manual for your own command	18

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/bash.
- You <u>cannot</u> leave <u>any</u> additional file in your directory than those specified in the subject.
- 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...
- You should use guacamole.42tokyo.jp to validate exercises.

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,

Piscine 101 Shell 00

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 : ex00/

Files to turn in : 42

Allowed functions: *

• Create a empty file called 42.

?>1s

42

?>



https://bit.ly/3an5uiL



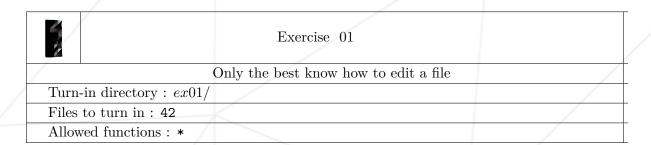
https://bit.ly/3pmfsqL



https://bit.ly/3qYROfh

Chapter IV

Exercise 01: Hello 42!



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

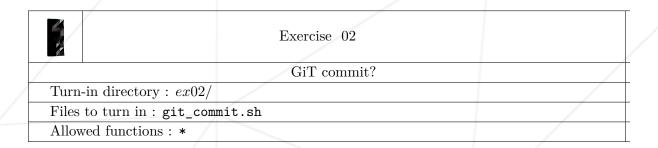
?>cat -e 42
Hello 42!\$
?>



https://bit.ly/3qZX1IV

Chapter V

Exercise 02: GiT commit



• 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$
%>
```

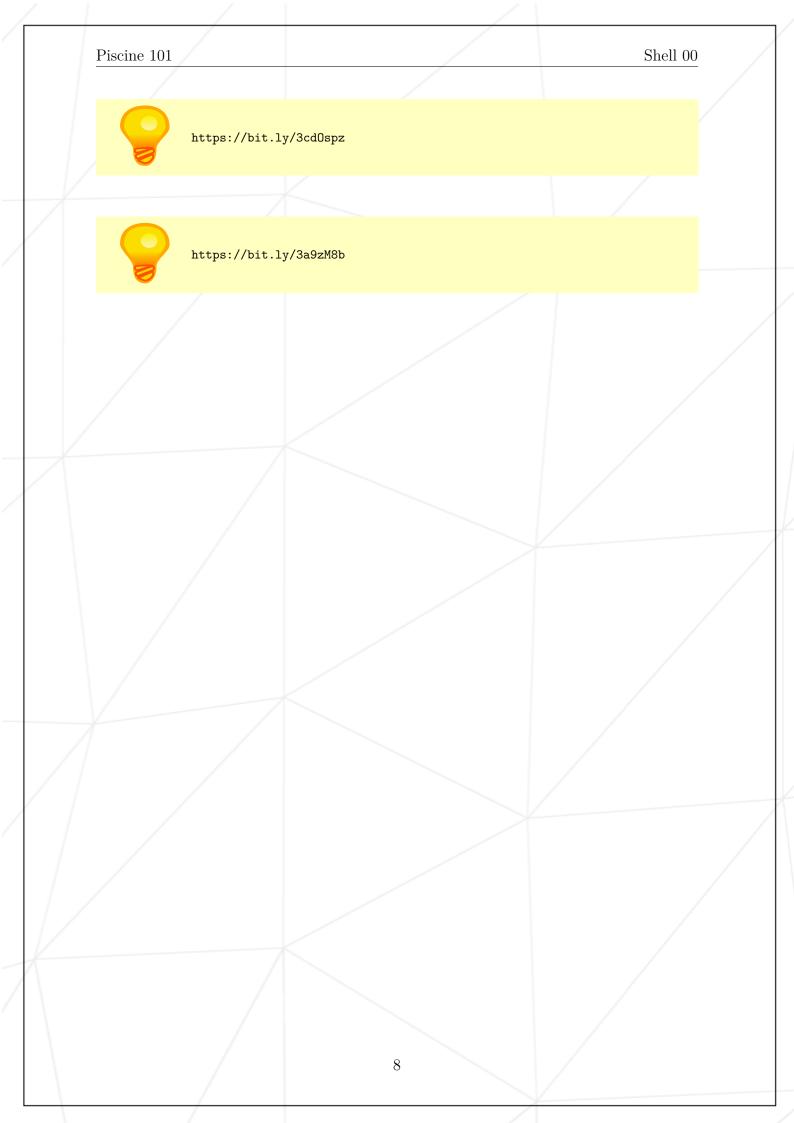
To test your script, we will use our own environment.



https://bit.ly/3pnst3o

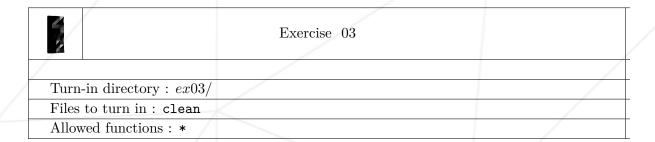


w https://bit.ly/2MuYWG9



Chapter VI

Exercise 03: clean



- ullet 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 \sim , 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.



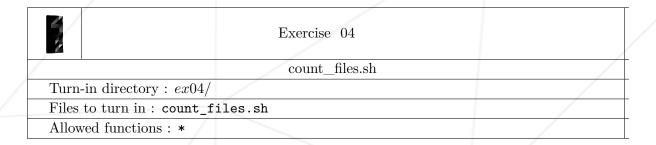
Don't delete the file which has # as a filename.



https://bit.ly/3aaGI5d

Chapter VII

Exercise 04: count_files

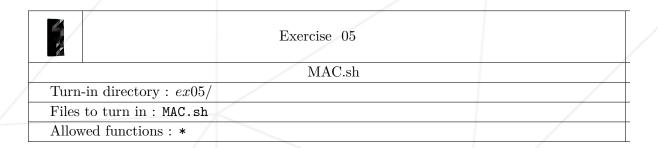


- Write a command line that counts and displays the number of regular files and directories in the current directory and all its sub-directories. It should print ".", but not "..".
- Example of output :

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

Chapter VIII

Exercise 05: MAC



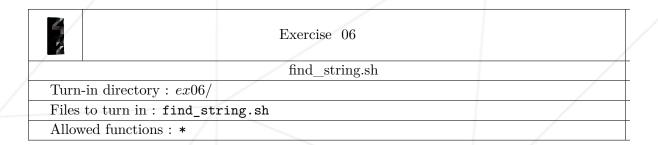
• Write a command line that displays your machine's MAC addresses for each network interface using ifconfig command. Each address must be followed by a line break.



https://bit.ly/2M7T6eb

Chapter IX

Exercise 06: Find string



- 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:

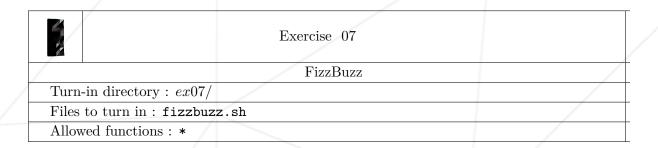
```
%> bash find_string.sh "42" | cat -e
./subject/test3.txt$
./test.txt$
./test2.txt$
%>
%> bash find_string.sh | cat -e
%>
%> bash find_string.sh "42" "24" | cat -e
%>
```



https://bit.ly/3ogt99i

Chapter X

Exercise 07: FizzBuzz



- 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.
- Command line argument's value passed to fizzbuzz.sh by testers(reviewers/Moulinette) will always be between 1 and 100. Other values will behave as undefined behavior.

Example:

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

Piscine 101 Shell 00

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



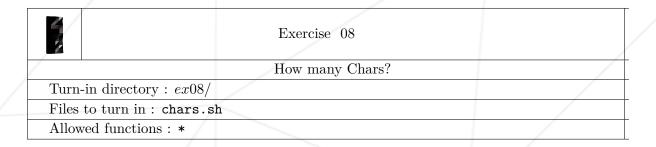
https://bit.ly/39kY93A



https://bit.ly/2MtIbew

Chapter XI

Exercise 08: How many Chars?



- 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 or directory$
/etc/passwd:6804$
%>
```



https://bit.ly/3cmF15U

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: *		

- Write a shell script that prints the file's content from a specified line to another specified line.
- Add a shebang at the top of the file. #!/bin/bash
- Check the example and follow the same format for the output of the command. Example:

```
%> ./file_viewer --help | cat -e
usage: file_viewer [-h] filename start end$
Display specific part of the file.$
positional arguments:$
  filename
              filename$
              starting line$
  start
               ending line$
optional arguments:$
  -h, --help show this help message and exit$
\mbox{\ensuremath{\%}{>}} ./file_viewer /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$
\mbox{\ensuremath{\%}{>}} ./file_viewer /etc/passwd 11 a | cat -e
file_viewer: illegal offset -- a$
%> ./file_viewer /etc/passwd a 13 | cat -e
file_viewer: illegal offset -- a$
    /file_viewer /etc/passwd | cat -e
```

Piscine 101 Shell 00

```
usage: file_viewer [-h] filename start end$
Display specific part of the file.$
positional arguments:$
             filename$
 filename
              starting line$ ending line$
 start
 end
optional arguments:$
 -h, --help show this help message and exit$
\mbox{\ensuremath{\mbox{\%}{>}}} ./file_viewer no
suchfile 11 13 | cat -e
file_viewer: nosuchfile: No such file$
%> ./file_viewer /etc/passwd 1000 1001
%> ./file_viewer /etc/passwd 10 1
%> ./file_viewer | cat -e
usage: file_viewer [-h] filename start end$
Display specific part of the file.$
positional arguments:$
 filename
             filename$
              starting line$
  start
              ending line$
 end
optional arguments:$
 -h, --help show this help message and exit$
```



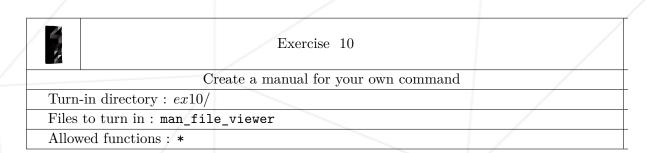
use space, not tabs.



https://bit.ly/2YkiOPg

Chapter XIII

Exercise 10: Create a manual for your own command



- Create a man page for file_viewer.
- Man section should be 1.
- \bullet Man should print out these sections: NAME, SYNOPSIS, DESCRIPTION, AUTHOR



man man