Lab 05 - The Shell

1. *How many variables are currently defined in your shell? (Tip: use `wc -l`)*

```
[student@UWS ~]$ printenv | wc -l
14
```

2. What are the values for the shell variables `DISPLAY`, `LOGNAME`, and `PATH`?

```
[student@UWS ~]$ printenv | grep "DISPLAY\|LOGNAME\|PATH"
LOGNAME=student
PATH=/usr/local/bin:/bin:/usr/bin
```

`DISPLAY` does not appear to be set at this time.

3. *To which directory does the `cd` command set you now?*

```
student@UWS ~]$ cd
[student@UWS ~]$ pwd
/home/student
[student@UWS ~]$ HOME=/etc
[student@UWS student]$ export HOME
[student@UWS student]$ cd
[student@UWS ~]$ pwd
/etc
```

`cd` with no arguments changed the present working directory to `/etc`

4. *In which directories are the `fsck`, `which`, and `whereis` commands found?*

```
[student@UWS ~]$ whereis fsck which whereis cp
fsck: /sbin/fsck.ext4 /sbin/fsck /sbin/fsck.ext2 /sbin/fsck.ext3
/usr/share/man/man8/fsck.8.gz
which: /usr/bin/which /usr/share/man/man1/which.1.gz
whereis: /usr/bin/whereis /usr/share/man/man1/whereis.1.gz
cp: /bin/cp /usr/share/man/man1/cp.1.gz
```

5. *How many directories are searched for the occurrence of a program or command?*

`whereis` will search through the `$PATH` and `$MANPATH` variables for programmes, commands, and manual pages.

```
[student@UWS ~]$ echo $PATH
/usr/local/bin:/bin:/usr/bin
```

6. *What is the output if you try to run `whereis cp`?*

```
[student@UWS ~]$ whereis cp
cp: /bin/cp /usr/share/man/man1/cp.1.gz
```

7. *What is the `whereis` argument used to search only manuals?*

`-m`

8. *Which binary directories are omitted for student as opposed to root? Explain why this might be the case.*

```
/usr/local/bin:/bin:/usr/bin
[student@UWS ~]$ su -
[root@UWS ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
```

Any directories containing system binaries are omitted from the student user's path given that programmes in these binaries run themselves as root.

9. *What is the output if you try to run `ipmine`?*

```
[root@UWS ~]# cp /sbin/ip /root/ipmine
[root@UWS ~]# ipmine
-sh: ipmine: command not found
```

`/root` is not present in `$PATH` which is why it doesn't run.

10. *Explain the effect of the two commands `PATH=$PATH:.` and `export PATH` in your own words.*

`PATH=$PATH:.` assigns the current value of `$PATH` along with a colon and the present working directory to the `PATH` variable.
`export PATH` defines a shell environment variable which allows functions and parameters to be called in the current session.

11. *Does `ipmine` run now? If 'yes' explain why.*

Yes. the path environment variable has been updated to include the present working directory. We invoke the command and the shell searches for an appropriate binary within the path variable. It was found, so it was run.

12. *What command provides the same output as `echo *` ?*

`ls`

13. *Explain the two command lines above in respect to the fact that the shell expands wildcards*

`*` will match any group of zero or more characters. The shell will expand a single `*` to match any string as an argument. In the case of `*.dat` it will expand to only include string which match the given pattern.

14. *Explain the output produced by the FOUR command lines in detail.*

Each of the commands uses `?` which, in unix, will match a single character. In the case of `echo ???.dat` the only files which will be returned to standard out are those which contain three characters before the `.dat` suffix. The same is true for four, five, and six character filenames in the remaining three commands.

15. *What does the shell force the echo command 'to echo' if the expansion of wildcards does not produce any results, as in the case of the last command line?*

The shell will yield zero matching results and defer to standard `echo` functionality: feeding the programme a string to echo back to standard out.

16. *Do you know what type of files are found by the above command? Explain the function of the 3 metacharacters and the wildcard given within:* `[K][0-9][0-9]*`

- `[K]` will match the letter `K` exactly,
- `[0-9]` will match any digit in that range exactly, the same holds for the following `[0-9]`,
- `*` will match zero or more preceding characters.

17. *What actually happened to the existing file test.dat? What was the 1st task the shell performed once the above command line was issued?*

The shell would have looked through `$PATH` to find the `echa` application.

18. *Reflect on some potential dangers of this default behaviour*

I'm not sure I understand the question but I suppose a user might attempt to execute some command, programme a script, or automate some function which might attempt to run either a non-existent programme or the incorrect programme altogether which would result in unexpected or unintended behaviour. The shell assumes the user knows what they're doing.

19. *Explain the differences in the output of the above command lines*

`echo "$HOME"` will cause the shell to interpret and evaluate the value between the quotation marks which will result in the value of `$HOME` being presented to the use via echo. Inverted commas will simply have the shell return the string literal.

20. *Explain the output of the above command lines, if necessary use the man pages to understand the different output.*

```
[root@UWS ~]# echo `echo $HOME`
/root
[root@UWS ~]# echo `hostname`
UWS
[root@UWS ~]# echo `hostname -i`
127.0.1.1
```

In each statement the text within the backticks will be evaluated by the shell and it will replace the text contained by the backticks in the original statement. The result of `echo $HOME` in this instance is `/root`, which is then echoed by the first `echo` command.

21. *What is the function of the semicolon* `;` *in the above command line?*

The semicolon indicates the end of a complete statement. Anything following a semicolon will be interpreted by the shell as a new, independent statement.

22. *Would the script `envdisplay` run without a change of the permission settings? Note the current permission settings in octal representation.

No, given the lack of entry in the permission bit for the current user. The octal value is `644`

23. *Note the new permission settings of `envdisplay`, after the `chmod` command was issued, in octal representation. Why is the script executable now?*

The new permissions octal value is `755`. Permission bits have been granted to the file and can now be executed.

24. *Explain the output of `envdisplay`.*

The content of `envdisplay` is a series of commands which will print the content of several environment variables to standard out.

25. *Explain why you should only see the line `I did something wrong`, if your script does not perform as intended.*

`||` between two commands ensures that the second command will only execute if the first command fails to execute appropriately.

26. *Explain the output of `envdisplay`. In particular, why does the first invocation of `envdisplay` not produce the string test, yet the second one does?*

The first command is simply evaluated as a string and passed as an argument to the script to evaluate, which it will, as a string. Prefacing the variable name with a `$` symbol indicates a variable whose associated value should be evaluated and passed into the script.

27. *What is the value of `$HOME` after the script has finished running? What has happened?*

When a script is run the shell evaluates the shebang if one exists and spawns a new instance of that application, in our example, `/bin/bash/`. This new instance of bash runs and executes the script, and its environment variable is changed. When the script closes so too does that instance of bash, and the calling instance of bash remains untouched.

28. *What is the value of `$HOME` now? What happened?*

The current shell was used to execute the script which means the `$HOME` environment variable has been set to `/etc/`.

29. *Besides `/insecure` and `/home/crazy` are there any other unprotected directories? Try with at least one other insecure permission (`766, 733` or `722`).*

```
[root@UWS /]# find ./ -type d -perm 777 -print
./
./dev/shm
./home/crazy
./insecure
```

30. *What does the `-perm` in the find qualifier do?*

It searches for files and directories by a given set of permissions.

31. *What would be the find command line to change the directories from `777` to `755`?*

```
[student@UWS /]$ find ./ -type d -perm 777 -exec chmod 755 {} \;
```

32. *Does your shell script work?*

Yes, though there's no output.

33. *Comment on the different output.*

The only difference was the speed in execution which, I imagine, is due to the drastically reduced search parameter.

34. *Check whether your shell-script works by running it twice in a row (the 2nd time no directories should be found) or by checking the permissions using the ls -l command on the directories that were found to have the permission 777. What is the output of the script after the 1st and 2nd run?*

The directories which were found to have insecure permissions octal values had theri values changed to reflect the `-exec` argument and its subsequent command. The second run yielded no results given the lack of candidate files.

35. *Given a PID and nice number will `renice` a process with the given PID to the given nice number. This should be a one line shell-script using two input variables `$1` and `$2`, representing the nice number and the PID of the process. Try it out with your cruncher script.*

```
renice $1 $2
```

36. *Try to get the script to find the required `PID` if provided with only a process name only and a nice number.*

```
renice $1 `pidof $2`
```