**Chapter 10: 12, 13, 14, 17, 30**

**10.12.**

Frame regular expressions to match these patterns.

1. jefferies jeffery jeffreys
   1. “^jeff[er][er][iy]e\*s\*
2. hitchen hitchin hitching
   1. “^hitch[ei]n\*g\*”
3. Heard herd Hird
   1. [Hh][ei]a\*rd$”
4. dix dick dicks dickson dixon
   1. “^dix\*c\*k\*s\*o\*n\*”
5. Mcgee mcghee magee
   1. “[Mm][ca]gh\*ee$”
6. wood woodcock woodhouse
   1. “^wood....e\*”

**10.13.**

How do these expressions differ?

1. [0-9]\* and [0-9][0-9]\*
   1. [0-9]\* will search for repetitions of characters of 0 or more entries. So this could include nothing at all, or it could include 1, 12, 121, 1234, 12345 as an example
   2. [0-9][0-9]\* will search for a pattern that has at least one digit to begin, and then any number of digits following.
2. ^[^^] and ^^^
   1. ^[^^] will search for a pattern where at the beginning of the line the single character is not ^
   2. ^^^ however will search for a pattern of ^^ at the beginning of the line

**10.14**

Frame a command sequence that looks at romeo's mailbox to tell him either that he has received a message from henry or that the Subject: line contains the word urgent or immediate in lower- or uppercase

grep -Ei 'henry|urgent|immediate'

**10.17**

How do you locate all lines containing printf, but not sprintf and fprintf, anywhere but at the end of the line.

You would have to use the regular expression “^printf[.]” That way at least one character is required at the end of the line, and it signifies that the line needs to start with printf.

**10.30**

Specify the command sequence needed to remove the directory /usr/local/bin from the PATH defined in $HOME/.profile

You would use the expression

sed 's/\(PATH\)\(/usr/local/bin\)\(.\*\)/\1\3' $HOME/.profile

**Chapter 7: 1, 2, 3, 4, 6, 8**

**7.1**

Explain the significance of the two parameters, $$ and $!. Explain the differing behaviour of the command echo $$ when run from the shell prompt and inside a shell script

The parameter $$ contains the process ID of the shell that you are currently in, whereas $! will display the process ID of the last background process that has begun. The command echo $$ will differ between the prompt and a shell script since these are two different shells to begin with. The command prompt forks a new shell to execute the script, therefore the echo $$ in the script will produce a different process ID.

**7.2**

Describe the similarities between processes and files.

Processes, like files, are organized in a hierarchial structure. The first process at the top is similar to the root directory. As well processes and files both have their own data structures as tables. The process table is similar to the inode. The process table contains the process ID's as well as a pointer to the process control block for each process. The code for a process and a file may be the same as well, as the a process is simply a file in execution, so the file's code get's copied over to the process when it is executed

**7.3**

If two users execute the same program are the memory requirements doubled.

Yes they are because in this case when each program is executed a process in memory is created which then will copy across the data and code of the file each time.

**7.4**

What are two options available to a parent after it has spawned a child? How can the shell be made to behave in both ways.

After a parent has spawned a child it has either the option to wait for the child to finish executing it's code, or it has the option to continue executing itself. In order for the parent to wait for the child to finish executing, the child must be run in the foreground. The parent process will then wait for the signal from the child indicating that it is done. If the child is run in the background control returns to the parent process.

**7.6**

Explain the attributes of daemon processes using three examples. How do you display and identify them?

One major attribute of daemon processes is that they have a PPID of 1, meaning that they were created by the init process. These processes also have a d at the end of their name as well as having a ? under the TTY column when listed. Meaning that they are run by the init process rather than any particular terminal.

In order to see them you could use the ps aux command

**7.8**

Unlike the build-in commands pwd and echo, which also exist as separate disk files, why is there no file named cd on any UNIX system.

Cd is an environment variable, when the command is run the shell looks to $CDPATH to replace the command in execution