Lab exercises for CMT304, 27 January 2020.

1. Write a filter that prints the number of unique file sizes in `/etc`.
2. Using a single command-line, create a set of directories of the form "YYYY-MM-DD" for each month in the years 2012—2018.
3. Write a Bash script that creates a set of files names N.txt, where N is a number ranging from 0 to 100.

   Hint: First line of a Bash script:
   ```
   #!/bin/bash
   ```
4. Write a shell script that renames the files from 3) to the form N.TXT.
5. Write a password generator in Perl. The password should consist of randomly-chosen letters, numbers, and special characters. The length of the password should be given on the command-line. The password should be printed to the screen.

   Hints:

   a) First line of a Perl program:
   ```
   #!/usr/bin/perl
   ```
   b) Random-number generation in Perl:
   `rand(max)` – creates a random number between 0 and (max-1).

Some further hints:

1. Create a directory specifically for this exercise.

2. Useful tools: `ls, sort, uniq, awk, wc, mkdir, mv`

3. Making a script executable: `chmod u+x SCRIPT`

4. Running a script in the current directory: `./SCRIPT`

5. `man <command>` is often quite helpful

Lab exercises for CMT304, 29 January 2018.

Solutions

1. Write a filter that prints the number of unique file sizes in `/etc`.

```
ls -l /etc | grep -v ^total | awk ` { print $5; }` | sort -n | uniq | wc -l
```

The filter first obtains a list of files (and directories – since in Unix all directories are also files, this is consistent with the task formulation) in long format, which includes the size. As the first line in the output lists a total, `grep -v` (in**v**erted pattern) is used to only extract the lines that do not start with "total" (the caret ^ denotes the beginning of the line). The `awk` program then extracts the 5th column of each line. `sort -n` sorts these columns numerically, and `uniq` reduces any duplicate lines to one. Finally, `wc -l` counts the lines in the output.

The following alternative solution considers all files in `/etc` recursively and only considers regular files (not directories, etc.):

```
du -b `find /etc -type f` | sort -k1 -n | uniq | wc -l
```

Here, the `du` command is used to show the disk usage of the files provided to it. Command expansion runs the `find` command to list all regular files (i.e. not directories) in `/etc`, recursively. The output of `du` consists of the file size and file name. This output is sorted using the `sort` command. The parameter `-k1` specifies that the first column of each line should be used as the key.

Note that the second solution may fail if there is a very large number of files in /`etc`, because then the number of files may exceed the maximum length of a command line. In that case, the `xargs` command might come in handy.

2. Using a single command-line, create a set of directories of the form "YYYY-MM-DD" for each month in the years 2012—2018.

```
mkdir {2012..2018}-0{1..9}-DD {2012..2018}-{10..12}-DD
```

The solution simply uses brace expansion to create the patterns, and `mkdir` to create the directories. Note that the task formulation asked for the ending -`DD` to each directory name.

Write a Bash script that creates a set of files named N.`txt`, where N is a number ranging from 0 to 100.

```
#!/bin/bash
touch {0..100}.txt
```

Again, the solution uses brace expansion. The touch command updates a file's access timestamp. More importantly, though, it creates the file if it does not already exist.

Write a shell script that renames the files from 3) to the form `N.TXT`.

```
#!/bin/bash
for f in {0..100}.txt; do
        mv $f ${f%txt}TXT
done
```

The script first creates the list of filename using brace expansion. Then `f` loops over this list. The `mv` command moves the file named in the first argument to the file or location named in the second argument. Here, the second argument uses suffix removal to remove "`txt`". This means that instead of e.g. the value "`43.txt`" the value "`43.`" is replaced. With `TXT` added, this results in "`43.TXT`". The `mv` command line for f=43.txt thus is "`mv 43.txt 43.TXT`".

3. Write a password generator in Perl. The password should consist of randomly-chosen letters, numbers, and special characters. The length of the password should be given on the command-line. The password should be printed to the screen.

```
 1   #!/usr/bin/perl
 2   #
 3   # This script generates a random password. The length of the
 4   # password must be given on the command line.
 5   #

 6   # Notes:
 7   #  - @ARGV contains the parameters given on the command line.
 8   #  - die($message) immediately exits the script (printing out
 9   #  - the optional $message).

10   $length = $ARGV[0];
11   die "Please provide a non-zero, non-negative length.\n"
         if $length < 1;

12   # Creates an array containing all the characters that can be used
13   @chars = (a..z, A..Z, 0..9, '/', '_', '-', '+', '?', '$', '%',
              '(', ')');

14   # Repeat this loop $length times.
15   for $i (1..$length) {

16      # Append a random character from the @chars array to $password.
17      $password .= $chars[rand($#chars + 1)];

18   }

19   # Print $password, terminated with a newline.
20   print "$password\n";
```

Note: An alternative, and very common and elegant way of parsing command-line arguments (lines 10..12) uses `shift` and `or`:

```
$length = shift @ARGV
```

```
      or die "Please provide a non-zero length.\n";
```

`or` evaluates its right-hand side expression only if the left-hand side evaluates to 0. Be aware that this does not test for negative numbers.