Done

Shell scripting

Introduction

The purpose of this practical is to practice using the shell and shell scripting. It may be useful to refer to some of the commands described in the previous practical.

Don't forget that you are required to submit the indicated exercise through myLearn for assessment.

Standard input and output

By default, UNIX systems read input from the controlling terminal, and print output and errors to that same terminal. However, you can redirect input and output to different sources/destinations. For example, you could redirect standard output to a file. If not given any file to copy, the cat command will read from standard input. You can redirect programs to output to a file using the > operator. Thus:

```
cat > newfile
```

will read whatever you type on the keyboard and copy that output to a file named *newfile* (Press +d (i.e. ctrl and d together) to send an end of file character to exit `cat`).

Similarly, you can redirect contents of a file to standard input using <. So

```
cat < newfile
```

will read its input from *newfile* (and output it to the screen).

You can combine multiple redirects, so

```
cat < newfile > newfile2
```

will copy the contents of *newfile* into a file called *newfile2*.

Note that redirecting output with > will replace the contents of the file you are redirecting to. If you wish to append to the file instead, use >>

Pipes

Pipes can be used to connect the standard output from one program to the standard input of another. For example

```
cat report | sort | uniq
```

will copy the contents of the file *report* to the standard input of `sort`, which will sort the lines in the input into alphabetical order. The output of `sort` will be piped to the standard input of `uniq`, which will remove duplicate adjacent lines. The overall result is that the contents of report will be sorted, adjacent duplicate lines removed, and the final result output to standard output (i.e. the terminal). You can combine redirects and any number of pipes to create powerful combinations of commands.

tee

`tee` is a utility that outputs its standard input to a file as well as to standard output. This can be very useful when piping because you can store the input for the command to the right of `tee` into a file. For example:

```
cat report | sort | tee sorted.txt | uniq
```

will output the sorted contents of the file *report* to the file *sorted.txt* (including duplicate lines) and display only unique lines to standard output.

sleep

Sometimes it is useful to have the computer pause for a period of time. Try running the following command (and wait 10 seconds):

```
sleep 10
```

bash scripting

While all of the commands we have covered so far are useful, and pipes and redirections make them even more powerful, it is difficult to type long commands and would be much better if we could store combinations of commands in a file and execute that file to run the entire combination. Shell scripting allows us to do that. For example, if we edit a file called *script.sh* and add the following content:

```
#!/bin/bash
# A simple shell script
echo Starting
date
echo Finished
```

and then make it executable using:

```
chmod u+x script.sh
```

then we can run the script using:

```
./script.sh
```

A shell is a sophisticated programming language with variables, input/output functions, arithmetic operations, conditional expressions, selection structures, and iteration structures, so scripts need not be as simple as the one above. The most common shell for programming is the Bourne shell, and we'll be using bash, an enhanced version of Bourne shell.

Exercises

Use material from this week's lectures to perform the following exercises:

1. Create a subdirectory of your home directory called *somejunk*
2. Create a file named *junk* in *somejunk* with the following contents:

   ```
   This file contains no useful information
   ```

3. Make *somejunk* the current working directory
4. What access rights do you have to the file *junk*?
5. What is the group associated with the file *junk*, and what permissions do users in that group have for this file?
6. What access rights do all other users have to the file *junk*?
7. Use `chmod` to set the access permissions on *junk* so no user (including you) can read it
8. Try to list the contents of the file *junk*
9. Change the access permissions so only you can read the file *junk*. Confirm that you can list its contents
10. Change the access permissions so all users can read the file *junk*
11. Remove *junk* and *somejunk* from your home directory
12. Use wildcards to write a command to list all the files/directories with the prefix *prac* in your home directory, with one entry per line (you may need to create some directories to test this)
13. Use output redirection to redirect the output of the previous command to a file named *pracDirs*
14. Use a pipe from the `ls` command to the `wc` command to count the number of files/directories with the prefix *prac* in your home directory
15. Write a single command using pipes that does everything required for the last three exercises (hint: use the `tee` command)
16. Write a bash program called *hello.sh* which prints "Hello World" to standard output. Use an editor to write your script, and don't forget to make the file executable before you run it:

    ```
    nano hello.sh
    chmod u+x hello.sh
    ./hello.sh
    ```

17. **[Advanced]** Write a program called *sumInt.sh* which reads two integers from standard input, calculates the sum, and prints the result to standard output
18. **[Advanced]** Modify *sumInt.sh* so that, if the user enters a negative number, the script loops to ask the user to enter the value again
19. You may wish to try some of the exercises at https://cmdchallenge.com for more practice. Or, for a more detailed of bash commands, have a look at Bash Notes for Professionals.
20. **[This exercise should be submitted through the Assessments section of myLearn for Tutorial Exercise 1]** This exercise is designed to introduce you to the Python Automarker tool. You should click on the Tutorial Exercise 1 Submission link in the Assessment section of COSC110's myLearn page to launch the tool. Familiarise yourself with the tool. For this exercise, the code you require is already provided (there is no need for you to understand it yet), so ensure you can correctly submit your grade after running all the tests successfully.

Last modified: Thursday, 1 February 2024, 10:52 AM