A shell is a command line interpreter which provides the user interface for terminal windows. It can also be used to run scripts, even in non-interactive sessions without a terminal window, as if the commands were being directly typed in. For example,

typing find . -name "\*.c" -ls at the command line accomplishes the same thing as executing a script file containing the lines:

#!/bin/bash

find . -name "\*.c" -ls

The first line of the script, which starts with #!, contains the full path of the command interpreter

(in this case /bin/bash) that is to be used on the file. As we have noted, you have quite a few choices

for the scripting language you can use, such as /usr/bin/perl, /bin/csh, /usr/bin/python, etc.

#! /usr/bin/python3

|  |  |
| --- | --- |
| **Character** | **Description** |
| **#** | Used to add a comment, except when used as **\#**, or as **#!** when starting a script |
| **\** | Used at the end of a line to indicate continuation on to the next line |
| **;** | Used to interpret what follows as a new command to be executed next |
| **$** | Indicates what follows is an environment variable |
| **>** | Redirect output |
| **>>** | Append output |
| **<** | Redirect input |
| **|** | Used to pipe the result into the next command |

Putting Multiple Commands on a Single Line

Users sometimes need to combine several commands and statements and even conditionally execute them based on the behavior of operators used in between them. This method is called chaining of commands.

There are several different ways to do this, depending on what you want to do. The **;** (semicolon) character is used to separate these commands and execute them sequentially, as if they had been typed on separate lines. Each ensuing command is executed whether or not the preceding one succeeded.

Thus, the three commands in the following example will all execute, even if the ones preceding them fail:  
  
**$ make ; make install ; make clean**

However, you may want to abort subsequent commands when an earlier one fails. You can do this using the **&&** (and) operator as in:

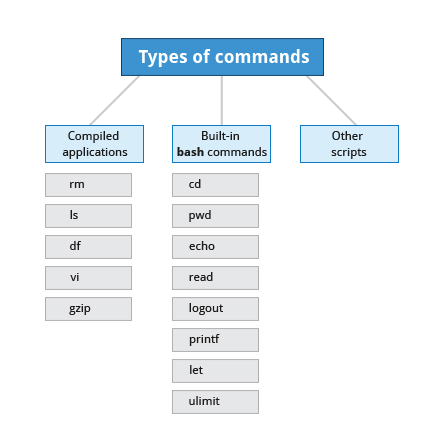
**$ make && make install && make clean**

If the first command fails, the second one will never be executed. A final refinement is to use the **||** (or) operator, as in:

**$ cat file1 || cat file2 || cat file3**

In this case, you proceed until something succeeds and then you stop executing any further steps.

Chaining commands is not the same as piping them; in the later case succeeding commands begin operating on data streams produced by earlier ones before they complete, while in chaining each step exits before the next one starts.



## Script parameters

|  |  |
| --- | --- |
| **Parameter** | **Meaning** |
| **$0** | Script name |
| **$1** | First parameter |
| **$2**, **$3**, etc. | Second, third parameter, etc. |
| **$\*** | All parameters |
| **$#** | Number of arguments |

Write a script which:

* + 1. Prompts the user for a directory name and then creates it with**mkdir**.
    2. Changes to the new directory and prints out where it is using **pwd**.
    3. Using**touch**, creates several empty files and runs **ls** on them to verify they are empty.
    4. Puts some content in them using **echo** and redirection.
    5. Displays their content using **cat**.
    6. Says goodbye to the user and cleans up after itself.

Click the link below to view a solution to the Lab exercise.