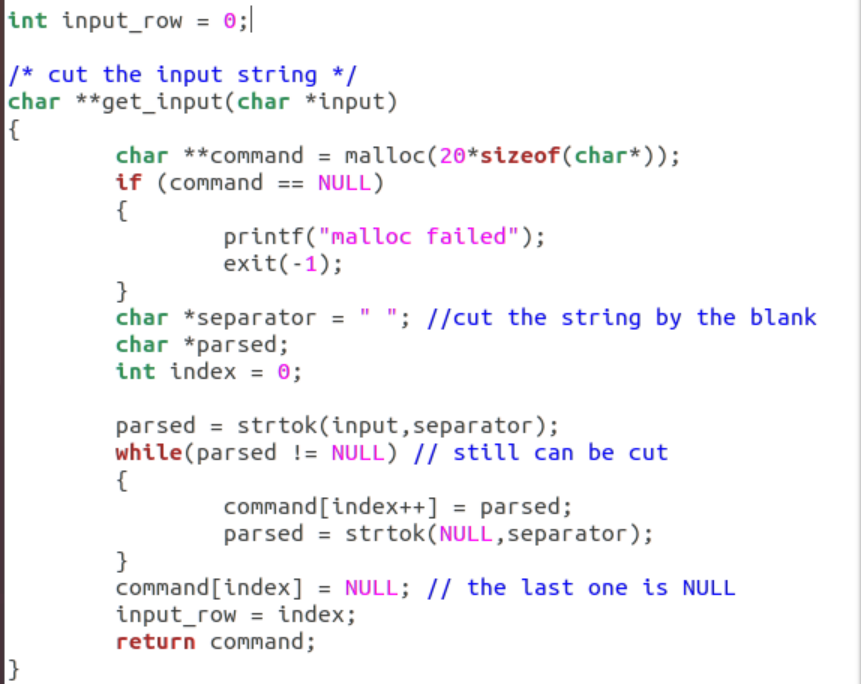
**程式碼**

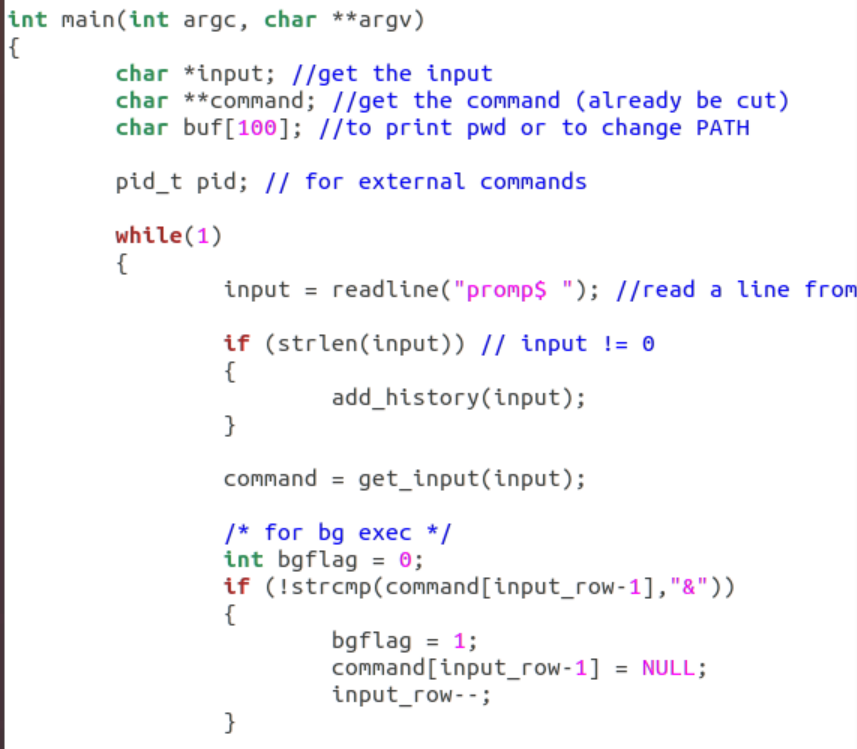
1. **全域變數及get\_input function**

****

input\_row是用來己算目前的command有幾個字串(如: ls -l 就是2)

get\_input function 用來剖析輸入的指令，會將輸入的整段指令以空白做切割，轉換成二維陣列後回傳。

1. **main function**

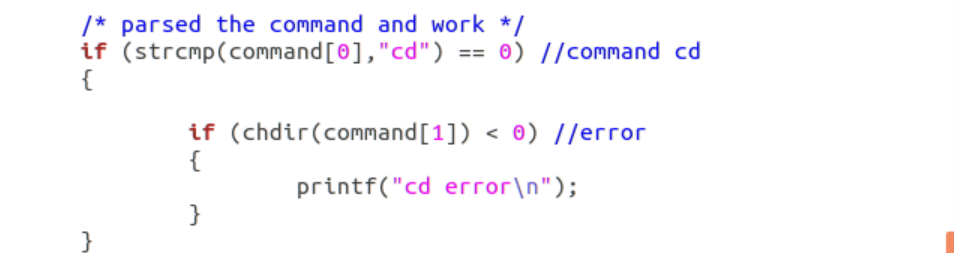


在main中會先用input 變數讀取整行輸入的指令，確定有讀到內容就會放入history中，這部分是在實作查詢歷史command的功能。

之後會判斷輸入指令的最後一個字串是否為&，如果是的話即代表這個指令是背景執行的，會設bgflag為1。(實測結果會在後面才統一截圖)

處理完這兩件事情後，下面會依序判斷其為內部指令或外部指令，並做相對應的工作

1. 內部指令 cd



利用chdir 來執行cd指令的工作，若回傳值小於0則代表有錯

1. 內部指令 pwd



會先判斷指令中是否需要輸出檔案，之後利用getcwd印出目前位置

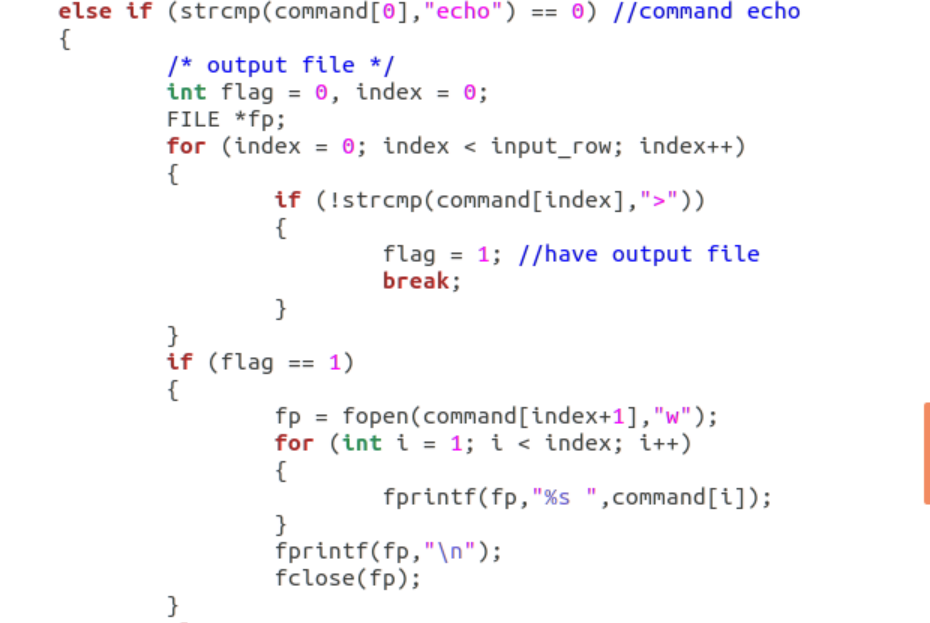
(若要輸出檔案，則會將位置印在檔案中)

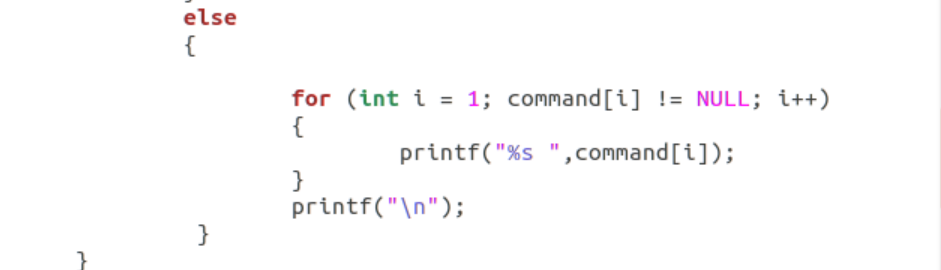
1. 內部指令 export



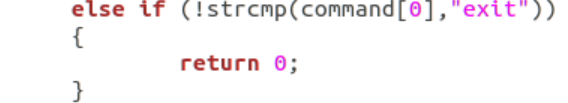
可以設定PATH的環境變數。會利用strncpy函式將新輸入的路徑存在buf中，再透過setenv重新設定環境變數，最後會印出目前的環境變數PATH為何

1. 內部指令 echo





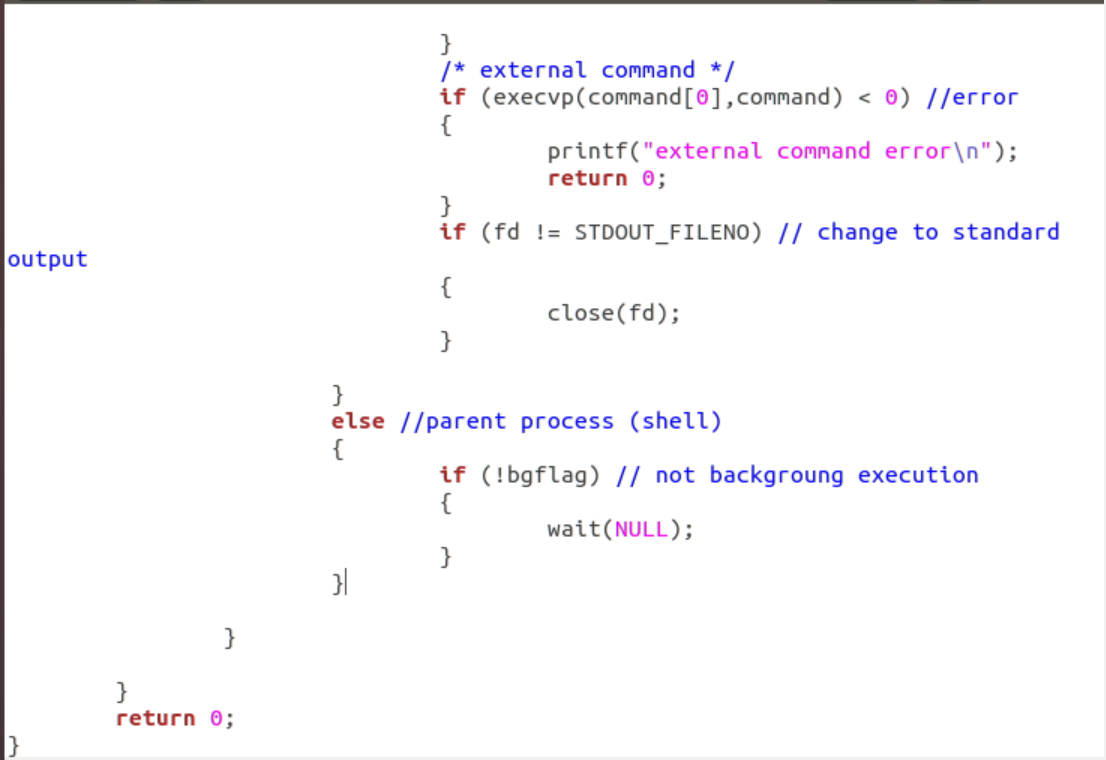
先判斷是否要輸出檔案，之後先input中希望echo出的內容印在螢幕或需要輸出的檔案中。



另外多寫了exit的內部指令，方便我們離開程式，不用按ctrl+C

1. 外部指令





建立child process，如果pid小於0代表error

如果等於0代表是child process，會先判斷是否需要輸出檔案，如果要的話就會將output的目的地從螢幕轉成輸出檔案

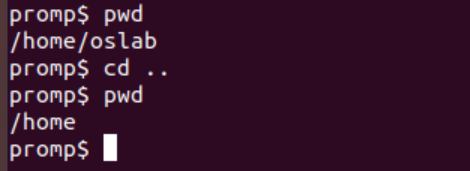
再來會利用execvp函式讓child process執行我們所輸入的指令，這邊如果回傳值小於0就代表有錯誤，會印出錯誤提醒。

執行完畢後，若原先是需要輸出到檔案的話(代表output的目的地被換地方了)，就會利用close(fd)將output換成預設的(螢幕)

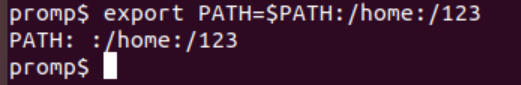
如果pid值不小於0，也不是0，就代表這是parent process，如果我們不用背景執行的話，就必須利用wait(NULL)讓他等待child process執行完畢，反之則不用，所以這邊用if的條件式來判斷。

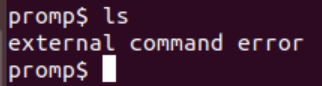
**功能實測**

1. cd and pwd

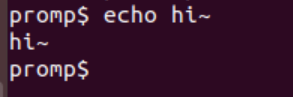


1. export

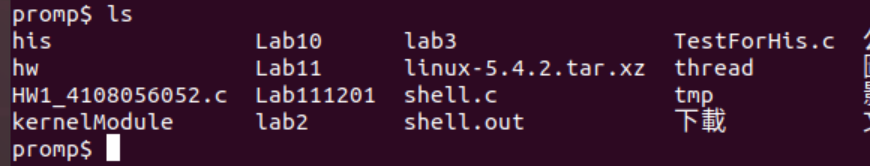




1. echo



1. 外部指令



1. 查詢歷史功能

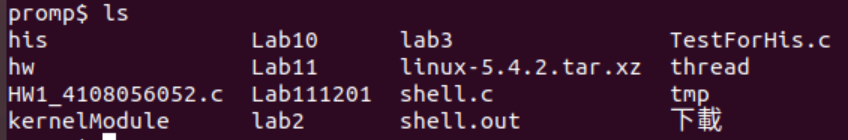
現在



按上鍵的話



按enter的話



1. 背景執行



沒有&，需等待五秒後才能重新輸入



有&，背景執行，按完後可以立即輸下一個指令

1. output redirection



