1. **shell 十三問?**

|  |
| --- |
| 8) $(( )) 與 $( ) 還有${ } 差在哪？   我們上一章介紹了 ( ) 與 { } 的不同，這次讓我們擴展一下，看看更多的變化：$( ) 與 ${ } 又是啥玩意兒呢？  在 bash shell 中，$( ) 與 ` ` (反引號) 都是用來做命令替換用(command substitution)的。 所謂的命令替換與我們第五章學過的變量替換差不多，都是用來重組命令行： \* 完成引號裡的命令行，然後將其結果替換出來，再重組命令行。 例如：   1. $ echo the last sunday is $(date -d "last sunday" +%Y-%m-%d)   *复制代码*  如此便可方便得到上一星期天的日期了... ^\_^  在操作上，用 $( ) 或 ` ` 都無所謂，只是我"個人"比較喜歡用 $( ) ，理由是：  1,  ` ` 很容易與 ' ' ( 單引號)搞混亂，尤其對初學者來說。 有時在一些奇怪的字形顯示中，兩種符號是一模一樣的(直豎兩點)。 當然了，有經驗的朋友還是一眼就能分辯兩者。只是，若能更好的避免混亂，又何樂不為呢？ ^\_^  2, 在多層次的復合替換中，` ` 須要額外的跳脫( \` )處理，而 $( ) 則比較直觀。例如： 這是錯的：   1. command1 `command2 `command3` `   *复制代码*  原本的意圖是要在 command2 `command3` 先將 command3 提換出來給 command 2 處理， 然後再將結果傳給 command1 `command2 ...` 來處理。 然而，真正的結果在命令行中卻是分成了 `command2 ` 與 `` 兩段。 正確的輸入應該如下：   1. command1 `command2 \`command3\` `   *复制代码*  要不然，換成 $( ) 就沒問題了：   1. command1 $(command2 $(command3))   *复制代码*  只要你喜歡，做多少層的替換都沒問題啦~~~  ^\_^  不過，$( ) 並不是沒有斃端的... 首先，` ` 基本上可用在全部的 unix shell 中使用，若寫成 shell script ，其移植性比較高。 而 $( ) 並不見的每一種 shell 都能使用，我只能跟你說，若你用 bash2 的話，肯定沒問題...  ^\_^  接下來，再讓我們看 ${ } 吧... 它其實就是用來作變量替換用的啦。 一般情況下，$var 與 ${var} 並沒有啥不一樣。 但是用 ${ } 會比較精確的界定變量名稱的範圍，比方說：   1. $ A=B 2. $ echo $AB   *复制代码*  原本是打算先將 $A 的結果替換出來，然後再補一個 B 字母於其後， 但在命令行上，真正的結果卻是只會提換變量名稱為 AB 的值出來... 若使用 ${ } 就沒問題了：   1. $ echo ${A}B 2. BB   *复制代码*  不過，假如你只看到 ${ } 只能用來界定變量名稱的話，那你就實在太小看 bash 了﹗ 有興趣的話，你可先參考一下 cu 本版的精華文章： <http://www.chinaunix.net/forum/viewtopic.php?t=201843>  為了完整起見，我這裡再用一些例子加以說明 ${ } 的一些特異功能： 假設我們定義了一個變量為： file=/dir1/dir2/dir3/my.file.txt 我們可以用 ${ } 分別替換獲得不同的值： ${file#\*/}：拿掉第一條 / 及其左邊的字串：dir1/dir2/dir3/my.file.txt ${file##\*/}：拿掉最後一條 / 及其左邊的字串：my.file.txt ${file#\*.}：拿掉第一個 .  及其左邊的字串：file.txt ${file##\*.}：拿掉最後一個 .  及其左邊的字串：txt ${file%/\*}：拿掉最後條 / 及其右邊的字串：/dir1/dir2/dir3 ${file%%/\*}：拿掉第一條 / 及其右邊的字串：(空值) ${file%.\*}：拿掉最後一個 .  及其右邊的字串：/dir1/dir2/dir3/my.file ${file%%.\*}：拿掉第一個 .  及其右邊的字串：/dir1/dir2/dir3/my 記憶的方法為：  # 是去掉左邊(在鑑盤上 # 在 $ 之左邊) % 是去掉右邊(在鑑盤上 % 在 $ 之右邊) 單一符號是最小匹配﹔兩個符號是最大匹配。  ${file:0:5}：提取最左邊的 5 個字節：/dir1 ${file:5:5}：提取第 5 個字節右邊的連續 5 個字節：/dir2  我們也可以對變量值裡的字串作替換： ${file/dir/path}：將第一個 dir 提換為 path：/path1/dir2/dir3/my.file.txt ${file//dir/path}：將全部 dir 提換為 path：/path1/path2/path3/my.file.txt  利用 ${ } 還可針對不同的變數狀態賦值(沒設定、空值、非空值)：  ${file-my.file.txt} ：假如 $file 沒有設定，則使用 my.file.txt 作傳回值。(空值及非空值時不作處理)  ${file:-my.file.txt} ：假如 $file 沒有設定或為空值，則使用 my.file.txt 作傳回值。 (非空值時不作處理) ${file+my.file.txt} ：假如 $file 設為空值或非空值，均使用 my.file.txt 作傳回值。(沒設定時不作處理) ${file:+my.file.txt} ：若 $file 為非空值，則使用 my.file.txt 作傳回值。 (沒設定及空值時不作處理) ${file=my.file.txt} ：若 $file 沒設定，則使用 my.file.txt 作傳回值，同時將 $file 賦值為 my.file.txt 。 (空值及非空值時不作處理) ${file:=my.file.txt} ：若 $file 沒設定或為空值，則使用 my.file.txt 作傳回值，同時將 $file 賦值為 my.file.txt 。 (非空值時不作處理) ${file?my.file.txt} ：若 $file 沒設定，則將 my.file.txt 輸出至 STDERR。 (空值及非空值時不作處理) ${file:?my.file.txt} ：若 $file 沒設定或為空值，則將 my.file.txt 輸出至 STDERR。 (非空值時不作處理)  tips: 以上的理解在於, 你一定要分清楚 unset 與 null 及 non-null 這三種賦值狀態. 一般而言, : 與 null 有關, 若不帶 : 的話, null 不受影響, 若帶 : 則連 null 也受影響.   還有哦，${#var} 可計算出變量值的長度： ${#file} 可得到 27 ，因為 /dir1/dir2/dir3/my.file.txt 剛好是 27 個字節...  接下來，再為大家介稍一下 bash 的組數(array)處理方法。 一般而言，A="a b c def" 這樣的變量只是將 $A 替換為一個單一的字串， 但是改為 A=(a b c def) ，則是將 $A 定義為組數... bash 的組數替換方法可參考如下方法：   1. ${A[@]} 或 ${A[\*]}   *复制代码*  可得到 a b c def (全部組數)   1. ${A[0]}   *复制代码*  可得到 a (第一個組數)，${A[1]} 則為第二個組數...   1. ${#A[@]} 或 ${#A[\*]}   *复制代码*  可得到 4 (全部組數數量)   1. ${#A[0]}   *复制代码*  可得到 1 (即第一個組數(a)的長度)，${#A[3]} 可得到 3 (第四個組數(def)的長度)   1. A[3]=xyz   *复制代码*  則是將第四個組數重新定義為 xyz ...  諸如此類的....  能夠善用 bash 的 $( ) 與 ${ } 可大大提高及簡化 shell 在變量上的處理能力哦~~~  ^\_^  好了，最後為大家介紹 $(( )) 的用途吧：它是用來作整數運算的。 在 bash 中，$(( )) 的整數運算符號大致有這些： + - \* / ：分別為 "加、減、乘、除"。 % ：餘數運算 & | ^ !：分別為 "AND、OR、XOR、NOT" 運算。  例：   1. $ a=5; b=7; c=2 2. $ echo $(( a+b\*c )) 3. 19 4. $ echo $(( (a+b)/c )) 5. 6 6. $ echo $(( (a\*b)%c)) 7. 1   *复制代码*  在 $(( )) 中的變量名稱，可於其前面加 $ 符號來替換，也可以不用，如： $(( $a + $b \* $c)) 也可得到 19 的結果  此外，$(( )) 還可作不同進位(如二進位、八進位、十六進位)作運算呢，只是，輸出結果皆為十進位而已： echo $((16#2a)) 結果為 42 (16進位轉十進位) 以一個實用的例子來看看吧： 假如當前的  umask 是 022 ，那麼新建文件的權限即為：   1. $ umask 022 2. $ echo "obase=8;$(( 8#666 & (8#777 ^ 8#$(umask)) ))" | bc 3. 644   *复制代码*  事實上，單純用 (( )) 也可重定義變量值，或作 testing： a=5; ((a++)) 可將 $a 重定義為 6  a=5; ((a--)) 則為 a=4 a=5; b=7; ((a < b)) 會得到  0 (true) 的返回值。 常見的用於 (( )) 的測試符號有如下這些：  <：小於 >：大於 <=：小於或等於 >=：大於或等於 ==：等於 !=：不等於  不過，使用 (( )) 作整數測試時，請不要跟 [ ] 的整數測試搞混亂了。(更多的測試我將於第十章為大家介紹)  怎樣？好玩吧..  ^\_^  okay，這次暫時說這麼多... 上面的介紹，並沒有詳列每一種可用的狀態，更多的，就請讀者參考手冊文件囉... |