



## 第七章

# 設定你的 Shell 提示訊息

### 7.01 為什麼要改變提示訊息？

在許多系統上，C Shell 預設的提示訊息是一個百分比符號（%），沒什麼大用處，不是嗎？這個提示訊息唯一告訴你的只不過是你已經登入了。

如果你很擅於記憶目前的目錄之名稱、所登入的電腦、你目前的登入名稱... 等等的事；而且，假設你絕對不會離開終端機很久的時間；或許這樣的提示訊息對你已經足夠了。

但我時常忘記這些東西，我可能會在很多地方登入主機，而且也時常因某事而中斷目前的工作，如果提示訊息無法提供更多的資訊，那麼我就必須找出我現在到底在哪？ - 利用 `pwd` 或 `whoami`。

我將提示訊息做了一些修改，讓它能提示我所需要的資訊，雖然它沒辦法幫我做到所有的事（至少在 C Shell 不能），但它確實可以讓事情簡單一些。

除此之外，玩玩你的提示訊息是件很有趣的事，它是最流行的 UNIX 遊戲之一，特別是對初學者而言。

看完這一章應該讓你知道如何開始動手。前面幾篇文章涵蓋了基礎的知識，之後的文章介紹了幾個不同的提示訊息以及如何產生這些提示訊息，你可以試試各種不同的提示訊息，看看哪個對你最有幫助。

-- JP

## 7.02 設定提示訊息的基礎知識

Shell 所顯示的提示訊息是存放在一個 Shell 變數 (6.08) 中，在 C Shell 這個變數是 `prompt`，在 Bourne Shell 則是 `PS1`，當然，它也可以是任何其它的 Shell 變數。〔這兩個變數在 `bash` 和 `tcsh` 還有許多其它特性，後面的文章有相關的範例。-- JP〕

舉個例子來說，如果我想在 C Shell 的提示訊息中加入我的登錄名稱，那麼我可以把下面這個指令放到 `.cshrc` 內：

```
set prompt="tim % "
```

在提示訊息的尾端留下 `%` 是個不錯的作法，這可以讓你能確定目前所用的是 C Shell，`%` 後面的空格可以讓你所輸入的指令和提示訊息保持一段距離。

如果我想在提示訊息中加入目前所登入的系統的名稱，可以這麼做：

```
set prompt="\`uname -n` % "
```

'...' 9.16  
uname -n 50.07

如果我想加入每個指令的歷程記錄號碼 (11.01)，可以用：

```
set prompt="\! % "
```

如果我三個資料都想要：

```
set prompt="tim@`uname -n` \!% "
```

提示訊息就會變成類似下面的形式：

```
tim@isla 43%
-- TOR
```

## 7.03 C Shell 提示訊息在 vi、rsh 所產生的問題

[ 錯誤的提示訊息可能對會啟動非交談式 Shell 的指令產生問題，這個問題已在許多版本的 C Shell 解決了，但 Chris 所談到的加速 .cshrc 的方法仍然可能面臨這些問題。 -- JP ]

如果當你設定 .cshrc 時，沒有小心檢查是否已設定了 prompt (2.02)，有許多版本的 C Shell 會將提示訊息列印到 vi 所使用的管線，以展開團塊 (glob) 字元 [ 檔案名稱萬用字元 (\*, ?, []) (1.16) 及 波浪符號 (~) (14.11) -- JP ]。

當你輸入 :r abc\* 時，vi 會開啟一個到 C Shell 的管線，並將指令 echo abc\* 寫入管線中，然後讀取回應，如果此回應中包含空格或跳行字元，vi 將會因此而感到困惑，如果你在 .cshrc 中將提示訊息設為 (n) [ 也就是說你在括號內顯示 history 號碼當作提示訊息 -- TOR ]，則 vi 會由 C Shell 取得：

```
(1) abc.file (2)
```

而不只是 abc.file。

解決方法就是把你的 .cshrc 改成這樣：

```
if 47.03      if ($?prompt) then
$?prompt 47.04 # 對交談式 Shell 的做法,像是:
                set prompt='\!' '
                endif
```

這個方法能夠行得通，是因為非交談性的 Shell 並沒有啟始的提示訊息，而交談性的 Shell 則會設為 %。

如果你的 .cshrc 非常大，當程式利用 `csh -c 'command'` 來執行其它程式時，而且如果你所有指令放在這個測試中，這種方法可以加快一些速度。

-- CT 於西元 1984 年 4 月 22 日在 `net.unix-wizards` 所發表的文章

## 7.04 利用內建指令加速提示訊息的設定

想要設定提示訊息，你必須執行一個指令（在絕大多數的 Shell，這個指令會設定一個 Shell 變數），在設定提示訊息之前，你或許必須執行其它的指令來獲取你所需要的資訊，例如目前目錄的名稱。Shell 可以執行兩種指令：內建指令及外部指令 (1.10)，內建指令的執行速度通常比外部指令快，在一些速度比較慢的電腦上，這種速度的差別就非常重要 - 為了重新設定提示訊息而必須等待幾秒鐘，這可會令人感到非常不耐煩。如果你能用更具有創造性的方式來使用內建指令，就能夠解決這個問題，讓我們來看看一些例子。

1. `pwd` 是一個外部指令，它會搜尋檔案系統 (14.04) 以找出目前目錄的名稱，（`pwd` 內建於某些系統，但那些版本並不會搜尋檔案系統。）不過，某些 Shell 可以利用一個變數來提供目前目錄的名稱，這個變數通常是 `$cwd` 或 `$PWD`，在較慢的電腦上，下面第一行指令可能會耗用較多的時間：

```
'...' 9.16      set prompt=" 'pwd' % "  
                set prompt=" ${cwd}% "
```

你必須做一個取捨；因為 Shell 的內建指令可能無法 (14.13) 提供一個正確的答案，而且在 C Shell 中，每當你跳到另一個新的目錄時，你必須執行一個新的 `set prompt` 指令，不過你可以用一個像 `setprompt` (7.05) 這樣的別名來幫你自動完成。

2. 如果你將目前的目錄放到提示訊息中，而或許你想要的其是路徑名稱的結尾部份（最後一個斜線後的名稱）。要怎麼編輯路徑名稱呢？大部分的人都是用 `basename` (4.5.18) 或 `sed` (3.4.24)，並由 `$cwd` 取得目前目錄的名稱，他們或許會輸入：

```
set prompt="`basename $cwd`% "
```

最快的方法就是 C Shell 內建的尾端運算子 `:t`：

```
{ } 6.08 set prompt="${cwd:t}% "
```

如果目前的目錄是 `/usr/users/banna/projects`，上面兩個提示訊息看起來都會像這樣（在百分比符號後面有一個空格）：

```
project%
```

C Shell 有好幾個像 `:t` 這樣的字串運算子 (9.06)，Korn Shell 和 `bash` 有功能更強大的字串處理運算子 (9.07)。

3. Korn Shell 和 `bash` 可以將其它 Shell 變數目前的值加到提示訊息中，因此，把 `$PWD` (6.03) 加到提示訊息字串（Shell 變數 `PS1`）中，那麼提示訊息就會顯示目前所處的目錄名稱。或使用其它的變數，任何時候只要它被改變了，那麼提示訊息也同時會改變。重要的是你必須以單引號（`'`）來儲存提示訊息，而不能用雙引號（`"`），這樣直到提示訊息被真正列印在螢幕時，提示訊息內的變數名稱才會被評估 (8.14 8.05)。

舉例來說，我將目前的目錄以及一個叫 `PSX` 的變數的值加到提示訊息內，當我改變了兩者其一，提示訊息也將同時改變：

```
$ PSX=foo
$ PS1=`$PWD $PSX`$ `
/home/jerry foo$ PSA=bar
/home/jerry bar$ cd ..
/home bar$
```

4. `tcsh` 和 `bash` 也有特殊的提示訊息字串格式，讓你可以加入主機名稱、使用者名稱、時間以及其它的資訊，你並不需要外部指令，也不類似 `steprompt` 這樣的別名來重設提示訊息字串。

舉例來說，想讓提示訊息顯示目前的目錄、一個跳行字元（在兩行的提示訊息 (7.05) 中跳到下一行）、目前的時間，最後以 \$ 或 % 結尾：

```
PS1=' $PWD\n\t \\\$ '                ...bash
set prompt \ '%~\\'                  ...tcsh
%p%
```

如果你想得到更多關於這方面的資料，請參考 *Using csh & tcsh* 以及 *Learning the bash Shell* 這兩本書，或是 Shell 的線上使用手冊。

因此，如果你的提示訊息花了很長的時間進行設定，改用內建指令可以節省不少時間。再舉個例子，7.11 介紹如何在 Shell 提示訊息中使用 `dirs`。

## 7.05 多行的 Shell 提示訊息

很多人喜歡在提示訊息中加入大量的資訊：主機名稱、目錄名稱、歷程記錄號碼、或許還有使用者名稱，但也有很多人花費了很多時間，試著把提示訊息變得短到足以放入螢幕內，以便輸入比 `ls` 還長的指令：

```
<elaineq@applefarm> [/usr/elaineq/projects/april/week4] 23 % ls
```

即使提示訊息非常短，如果你在執行了一些指令之後再看螢幕，想再由提示訊息得到資料是有點困難的（真正的終端機並不會把使用者的輸入變成粗體，因此我在下面的範例中也沒這麼做）：

```
<elaineq@applefarm> [~] 56% cd beta
<elaineq@applefarm> [~/beta] 57% which prog
/usr/tst/applefarm/bin/beta/prog
<elaineq@applefarm> [~/beta] 58% prog
61,102 units inventoried; 3142 to do
<elaineq@applefarm> [~/beta] 59%
```

有個不錯的解決方法，就是將提示訊息變成很多行而不是僅有一行，下面 `.cshrc` 的一部分，它是用來設定三行的提示訊息：一行是空白列、一行是主機名稱及目前所在的目錄，第三行則是歷程記錄號碼以及一個百分比符號：

```
uname -n 50.07      set hostname=`uname -n`
                    alias setprompt `set prompt=" \\  
{..} 6.08         ${hostname}:${cwd} \\  
                    \! % "  
                    alias cd `chdir \!* && setprompt`  
                    setprompt          # to set the initial prompt
```

經過這樣的設定，提示訊息看起來就像這樣：

```
applefarm:/usr/elaineq/projects/april/week4  
23 % prog | tee /dev/tty | mail -s "prog results" bigboss@corpoffice  
61,102 units inventoried; 3142 to do  
  
applefarm:/usr/elaineq/projects/april/week4  
24 % cd ~/beta  
  
applefarm:/usr/elaineq/beta  
25 % prog | mail joanne
```

中間的空白列可用分隔每個指令 - 不過你可能會因為想節省空間而省略這個空白列。舉個例子來說，Mike Sierra (目前任職於 O'Reilly & Associates) 用的是一行星號 (\*)：

```
***** 23 *** <mike@mymac> *** /home/mike/calendar *****  
% cd September  
***** 24 *** <mike@mymac> *** /home/mike/calendar/September *****  
%
```

並不一定要設定多行的提示訊息才能將跳行字元加到 `bash` 的提示訊息內，你可以在任何你想斷行的地方加入一個 `\n` (表示一個跳行字元) 就可以了。

我最喜歡多行提示訊息的原因，在於它可以提供你許多的資訊，但仍保留整個螢幕的寬度讓我輸入指令。當然，你也可以在提示訊息中加入不同的資訊，但最重要的概念是：如果你需要更多的資訊，而且也需要輸入指令的空間，就試試多行的提示訊息吧。

```
-- JP
```

## 7.06 給需求簡單的使用者的“選單式提示訊息”

有些人不喜歡在提示訊息中放在目前的目錄或主機名稱之類的資訊，這看起來凌亂不堪。告訴你另一個方法，如果你的終端機或視窗系統有個狀態列或標題列，你可以把所需要的資訊放在這兒，這是個好法子；因為你在執行程式時還能看到這些資訊，但缺點是如果你執行一個連到其它主機或目錄的指令，但並未更新狀態列內的資料，那麼你由狀態列所得到的資訊可能已經過時了。

當你執行指令 `cd` 時，有個使用 `echo` 指令的別名會將特殊的跳脫序列 (5.08) (終端機指令) 寫到終端機或視窗。

下面是個 `cd` 的別名以及其它讓你放在 `.cshrc` 的指令，如果我登入 `www.jpeek.com` 的 `/home/jpeek`，這個別名將會存放：

```
www:/home/jpeek
```

在狀態列，當然，你可以改變狀態列的格式，改變把下面的指令字串 `${host:h}:${cwd}`，換成你需要的資料。

```
`echo...033 45.35      set e="`echo -n x | tr x `033`" # Make an ESCape character
                        set host=`uname -n`
                        # Puts $host and $cwd in VT102 status line. Escape sequences are:
                        # ${e}7 = save cursor position, ${e}[25;1f = go to start of status
                        # line (line 25), ${e}[0K = erase line, ${e}8 = restore cursor
&& 44.09      alias cd 'chdir `!* & & ` \
                        echo -n "${e}7${e}[25;1f${e}[0K   ${host:h}:${cwd}${e}8''
```

如果你都是使用 VT102 型的終端機（大多數人都是如此），這個別名就可以正常運作，如果你使用其它類型的終端機，請參考它的使用手冊或是 termcap/terminfo 內的資料項，以找出適用於此終端機的跳脫序列。

使用一種類型以上的終端機（而且並不全都與 VT102 相容）的人，可以加入一個 case (44.05) 或 switch (47.06) 來測試終端機的型態，並使用專為此終端機所撰寫的 cd 別名。（在沒有狀態列的終端機上，這個別名也可以將狀態的資訊放到 Shell 的提示訊息裡。）但你可能遇到一些麻煩：如果這個別名定義在你的 .cshrc 裡，但終端機型態卻在 .login 中設定，那麼在別名被讀取之前，終端機型態還未被設定。

如果你使用遠端登入 (1.12)、子 Shell (38.04) 或其它的東西，那麼狀態列所包含的資料可能與事實並不吻合，原因可能是在狀態列放入一個新的提示訊息時，卻沒有在應該重新設定原始的提示訊息時進行重新設定，而最簡單的解決方法就是使用下面這個指令，將目錄換到目前的目錄並重新設定狀態列。

```
% cd .  
-- JP
```

## 7.07 給初學者使用的“選單型的提示訊息”

有些人並不喜歡面對 UNIX 的 % 或 \$ 之類的 Shell 提示訊息，如果你通常只執行一些特定的 UNIX 指令，你可以把這些指令的名稱放到 Shell 提示訊息裡，下面簡單的單行 Bourne Shell 提示訊息，用於 .profile：

```
PS1='Type "rn", "mailx", "wp", or "logout": '
```

接下來是一個用於 C Shell .cshrc 的多行提示訊息：

```

($?prompt) 2.09      if ($?prompt) then
                    set prompt='\\
                    Type "rn" to read the news,\\
                    type "mailx" to read and send mail,\\
                    type "wp" for word processing, or\\
                    type "logout" to log out.\\
                    YES, MASTER? '
                    endif

```



(7.03)

看過這兩個例子，相信你應該有些概念了吧！

-- JP

## 7.08 特效顯示 Shell 的提示訊息

如果在提示訊息中有某些你想凸顯的資訊；或你希望提示訊息能與螢幕上其它的文字有所區別；你可以用特效字元來達到目的。如果你的終端機擁有可用來改變字元顯示方式的特殊跳脫串列（escape sequence，絕大部分的終端機都有，5.08），你可以用它們來凸顯部分或全部的提示訊息。

舉個例子來說，如果你希望以 root（也就是超級使用者）身份登入的人能夠注意到他們目前的身份，可以讓 root 提示訊息的某個部份不停閃爍，也就是把下面這幾行放在 root 的 .cshrc：

```

# Put ESCape character in $e. Use to start blinking mode (${e}[5m)
# and go back to normal mode (${e}[0m) on VT100-series terminals:
set e="`echo x | tr x \\ 033`"
echo...033 45.35      set prompt=" ${e}[5mroot${e}[0m@`uname -n`# "
uname -n 50.07

```

提示訊息看起來就像這樣，其中 root 這個字會不停閃爍：

```
root@sys.ora.com#
```

這個提示訊息設定在雙引號 ( " ) 之內，因此 `uname -n` 指令只會在 PS1 字串第一次儲存時執行，在某些 Shell (像是 `bash` 和 `pdksh`)，你可以在 PS1 字串外加上一個單引號 ( ' )，這就會在字串內放一個反單引號 ( ` )，而當 Shell 在印出每個提示訊息之前，就會先對它們進行解譯。(在上面這個例子中這種作法是無用的，因為 `uname -n` 的輸出永遠不會改變，但如果你希望常常更新提示訊息所包含的資訊，那這是個簡便的方法。) 8.14 提供了更詳細的資料。

因為同一個跳脫串列並不能適用所有的終端機，如果終端機型態屬於 DEC VT100 系列，最好加上一個 `if` 測試 (47.03)，而且只設定提示訊息。表 7-1 列出了 VT100 及相容終端機的一些跳脫序列，每個序列的 ESC 表示一個跳脫字元。

表 7-1 VT100 用於控制特效顯示的跳脫序列

序列	功用
ESC[1m	粗體字，加強前景
ESC[4m	底線
ESC[5m	閃爍
ESC[7m	反白 (黑底白字)
ESC[0m	關閉所有的特性

當然，如果你的終端機需要的話，你也可以使用不同的跳脫序列，比較好的方法是用 `tput` 或 `tcap` (41.10) 這類的程式，在 `termino` 或 `termcap` 資料庫中找到關於你的終端機的跳脫序列，然後將此跳脫序列儲存在一個 Shell 變數中 (6.08)。

`bash` 會對提示訊息中的八進位字串進行解譯，因此你可以把上面那兩個指令簡化為下面這個版本，如果你願意的話，也可以把反單引號 ( ` ) 換成 `$( 和 )` (9.16)：

```
PS1="\033[5mroot\033[0m@\`uname -n`# "
```

tcsh 的 Eight-bit-clean 版可以將陰影效果 (standout)、粗體、底線 - 以及任何的終端機跳脫序列 - 放到 Shell 提示訊息內，例如說，%S 可以啟動 standout 模式，%s 則會結束此模式。如果你想獲得最完整的資料，請參考 tcsh 的線上使用手冊。

舉例來說，如果你想做一個和上面一樣的提示訊息，並將 root 這個字設為 standout 模式 (tcsh 把主機名稱放在 %m 內)：

```
set prompt = '%Sroot%s@%m# '  
  
.. JP
```

## 7.09 利用 \$SHLVL 顯示 subshell 的深度

如果你像我一樣，當啟動一個 shell escape (30.26) 或任何 subshell (38.04) 時，常常會忘記目前所在的並不是登入 Shell，Shell 的歷程記錄 (11.01) 也可能會感到困惑、Shell 變數 (6.08) 可能也還沒有設定、而且也可能發生其它的問題。tcsh (8.03) 和 bash (8.02) 有個內建的 SHLVL 環境變數，可以讓你追蹤你目前所在的 Shell 是屬於第幾層的 subshell，這篇文章說明如何在 C Shell 下設定 SHLVL 變數。在 ksh 下，如果設定了環境變數 ENV (6.03)，你也可以使用類似的步驟。tcsh (以及 csh) 也有一個 Shell 變數 shlvl 可提供相同的資訊。

如果你在最上層的 Shell，\$shlvl 的值就是 1，在第一個子 Shell，這個值是 2，在子 Shell 的子 Shell，這個值是 3，依此類推。你可以用這個變數來控制 Shell 的啟始檔案 - 例如，在 .cshrc 中有些指令只有當你第一次登入 (\$shlvl 是 1) 才執行，而在其它的子 Shell 下都不執行，你也可以把 %shlvl 放在提示訊息 (7.01)？(但只要放在子 Shell？ - 可以提醒你目前並不在最上層的 Shell)。你可以把最上層 Shell 的提示訊息設為 mike%，第一層子 Shell 設為 (1) mikee%，第二層子 Shell 設為 (2) mikee%，依此類推。下面是一個 .cshrc 的提示訊息設定範例：

```
# If this is a subshell, put shell level in prompt:
if ($SHLVL == 1) then
    set prompt=" ${USER}% "
else
    set prompt=" ($SHLVL) ${USER}% "
endif
```

bash 並不需要 if，因為登入 Shell 會讀取 .bash\_profile (或 .profile) - 而子 Shell 會讀取你的 .bashrc。下面這些指令可用來設定我前面所談到的提示訊息：

```
PS1=' \u\$ ' ...for the .bash_profile
ZPS1=' ($SHLVL) \u\$ ' ...for the .bashrc
```

bash 和 tcsh 都使用相同的環境變數，因此你可以由一個 Shell 啟動其它的 Shell，而深度的設定仍然是正確的。下面是如何讓 csh 與 tcsh 和 bash 一起工作（如果你並不使用其它 Shell，也可以用來和自己工作）的方法，把下面這幾行程式碼放在你的 .cshrc？：（註）

```
# Stuff for top-level logins and rsh's...
if (! $?SHLVL) then
    # This section read by both interactive and non-interactive shells.
    #
    # $SHLVL has never been set. Set it to show that this is
    # is a top-level shell; increment it to 1 below:
    setenv SHLVL 0
    ...
endif

# Set shell level (depth of nested shells).
# (Note: csh can't do arithmetic on environment variables.)
set shlvl = $SHLVL
@...+ 47.04 @ shlvl++
setenv SHLVL $shlvl
```

註 你同時使用 csh 和 tcsh，而且這兩個 Shell 使用相同的 .cshrc 嗎？tcsh 不應該執行由 set shlvl 開頭的那三行，它已經設定了那些變數，因此在那三行外加上了 if (! \$tcsh) 和 endif。

```
 $?prompt 2.09      if ($?prompt) then
                    # This section only read by interactive shells:
                    ...put prompt-setting code (from above) here
                    endif
```

在你的帳號下，會由最上層的 Shell 啟始檔案（像是 `.login`）啟動一個視窗系統嗎？是的話，下面那段範例程式（這段程式碼是給 `.login` 用的）會重新設定 `SHLVL`，讓視窗中的 `SHLVL` 從 1 開始 - 並扮演最上層 Shell 的角色。這段程式碼假設你的第一個登入 Shell 啟動於稱為 `/dev/console` 的 `tty`，而所開啟的視窗將不會有個叫 `/dev/console` 的 `tty`（如果你不確定，請用 `who` (51.04) 檢查），你或許必須適應它，這個技讓是在你啟動視窗系統之前，先將 `SHLVL` 設為 0，當視窗的 Shell 啟動時，它們將會把 `SHLVL` 增為 1：

```
 # If on workstation console, bury this shell and run X right away:
 if ( "`/bin/tty`" == /dev/console) then
     setenv SHLVL 0
     xinit          # Start X window system
 endif
```

讓這段程式能在所有情況（`rsh` (1.33)、`su` (22.22)、`shell escapes` (30.26) - 交談式及非交談式、子 Shell、視窗系統、`at` (40.03) 等等）都能正常運作是很有挑戰性的工作 (2.07)，這需要一點事先的計劃，先坐下來，想想你所有啟動子 Shell 的方法 ... 哪些子 Shell 是交談式的，哪些不是 ... 它們是否由父行程取得 `SHLVL`（如果你不能確定，可以用 `env` 或 `printenv` 指令 (6.01) 來做測試），然後安排哪些 Shell 需要哪種 `SHLVL` 的設定方式，如果這樣太複雜，讓它能在絕大部份的情況正常運作就可以了！如果你使用許多子 Shell，這個系統會方便到讓你忽略它的存在。

-- JP

## 7.10 空的 Shell 提示訊息有什麼用？

假使你常用滑鼠在視窗間複製及貼上指令，這個技巧將十分有用。

某些我曾用過的終端機（像是舊型的 Hewlett-Packard 和 Tektronix 終端機）擁有區域編輯的功能，你可以向上移動游標以取得前一個指令列，做一些編輯，然後再按下 SEND LINE 鍵，將這一個指令列重新送到主機，這和某些 Shell 所擁有的複雜的指令列編輯 (11.13) 不同，或許你的終端機也有這種功能。

但問題是除非我刪除螢幕上的 Shell 提示訊息（%），否則它不會送回 Shell，而且會出現錯誤訊息“%: Command not found.”，因此我把 Shell 提示訊息設為：

```
set prompt='    '
```

沒錯，四個空格。絕大部份的 UNIX 指令都將輸出放在第一行，因此我的提示訊息就很容易找到，因為它們就在凹下去的地方，而且 Shell 並不在乎我在指令列前面加上了四個空格，因此在我使用新終機但沒有 SEND LINE 鍵之前，一切都能運作得很好。

如果你希望在提示訊息中加入某些資訊，可以使用多行的提示訊息 (7.05)，並在最後一行加入四個空格。

```
-- JP
```

## 7.11 提示訊息？的 dirs：比 \$cwd 好的選擇

C Shell 在 \$cwd (14.13) 內存放了目前目錄的絕對路徑名稱，大部分的人在提示訊息中所用的都是這個變數，但如果你使用 pushd 及 popd (14.06) 指令，你可能很難記得目前到底是目錄中的哪一層（至少我是如此），而且，你想要把使用者目錄的絕對路徑名稱縮短成一個 tilde (~)，而在提示訊息中節省一些空間？告訴你該怎麼做：執行 dirs 指令，並把它的輸出結果用於提示訊息。下面是一個給 cd 使用者的簡單別名：

```
alias cd 'chdir `!` && set prompt=" `dirs`% "'
```

而提示訊息就像這樣：

```
/work/project % cd
~ % cd bin
~/bin %
```

如果你想製造一個多行的提示訊息 (7.05)，並顯示目錄堆疊，只要把下面這段程式碼放在 `.cshrc` 就可以了：

```

# PUT hostname.domain.name IN $hostname AND hostname IN $HOST:
set hostname=`uname -n`
setenv HOST `expr $hostname : '\([^.*]*\)'.*'`
alias setprompt 'set prompt="\\"
${USER}@${HOST} `dirs`\"
\! % "'
alias cd 'chdir \!* & & setprompt'
alias pushd 'pushd \!* & & setprompt'
alias popd 'popd \!* & & setprompt'
setprompt # SET THE INITIAL PROMPT

```

因為 `bash` 可以在每次設定提示訊息時執行一個指令，而且它有內建的提示訊息運算子 (7.04)，因此以上所談的在 `bash` 只要一行就搞定了：

```
$(...) 9.16 PS1=' \n\u@\h $(dirs)\n! \${ '

```

這會在每個提示訊息前加上一個空白列：如果你不想要這個空白列，可以把 `setprompt` 別名的第一行和第二行接在一起，或是把第一個 `\n` 刪掉。現在讓我們把一些目錄推入堆疊中，看看會出現什麼提示訊息：

```

jerry@ora ~
1 % pushd /work/src/perl
/work/src/perl ~

jerry@ora /work/src/perl ~
2 % cd ../cnews

```

```

jerry@ora /work/src/cnews ~
3 % pushd ~/bin
~/bin /work/src/cnews ~

jerry@ora ~/bin /work/src/cnews ~
4 %

```



當然，這個提示訊息看起來是有點累贅，因為每個 `pushd` 指令也都會顯示 `dirs` 的輸出。不過，在提示訊息中顯示目錄堆疊，對稍後會介紹的一些指令是很有用的。如果你的目錄堆疊有很多資料項，提示訊息的第一行會比螢幕還寬，如果遇到這種情況，你可以把 `dirs` 的輸出存到一個 Shell 陣列 (47.05)，然後用一個類似 `sed` 的指令，或是內建的 `cs` 字串編輯 (9.06) 來編輯它。

舉例來說，如果只想顯示 `dirs` 輸出中每個路徑的最後一個部分，可以使用下面這個別名，C Shell 的運算子 `:gt` 會編輯所有的單字，直到每個路徑名稱的最後面：

```

alias setprompt 'set dirs=(`dirs`); set prompt="\
${USER}@${HOST} $dirs:gt\
\! % "'

```

看看這個提示訊息，如果你忘了在提示訊息中這些名稱的意義，只要輸入 `dirs` 就可以了：

```

jerry@ora bin cnews jerry
5 % pushd ~/tmp/test
~/tmp/test ~/bin /work/src/cnews ~
...
jerry@ora test bin cnews jerry
12 % dirs
~/tmp/test ~/bin /work/src/cnews ~

```

47.05 有一個相關的技巧：將目錄堆疊儲存在一個陣列變數中。

-- JP

## 7.12 外部指令送出信號來設定變數

Bourne Shell 的 `trap` (44.12) 可以在 Shell 得到一個信號 (38.08) 時 (通常是由 `kill` 指令所發出的), 執行一個或多個指令, Shell 可以執行任何程式, 包括設定 Shell 變數的指令, 例如, Shell 可以重新讀取設定檔, (38.11 有關於這方面的內容), 或是設定一個新的 PS1 提示訊息, 讓它隨時執行每一個指令 (例如另一個 shell script 或一個 cron 的工作 (40.12)) 以送出一個信號給 Shell, 還有許多可以做的事。

這個技巧使用信號 5, 這個信號通常沒有被使用, 當 Shell 收到信號 5 時, `trap` 將執行一個指令以取得日期及時間, 然後重新設定提示訊息, 一個在背景 (1.27) 的工作每一分鐘發出一個 `trap`, 因此每隔一分鐘, 當你輸入任何指令之後, 提示訊息將會被改變。

你可以執行任何指令: 計算使用者的數目、顯示平均負載 (39.07) ... , 而且對較新的 Shell (例如 `bash`) 而言, 每次顯示 Shell, 都能在 backquotes (9.16) 執行一個指令 - 7.08 有個範例。但是, 要讓一個外部指令能在任何時間更新一個 Shell 變數, `trap` 這個技巧仍是最佳的選擇。

現在讓我們來看一個比較特殊的例子, 把日期和時間放在舊式 Bourne Shell 的提示訊息?, 如果你的系統的 `date` 指令並不瞭解 `date` 的格式 (像是 `+%a`), 那麼就去找一個辦得到的 - 像是本書所附 CD 的版本 (51.10), 然後把下面這幾行程式碼加到你的 `.profile` ? (或就直接在 Bourne Shell 的提示訊息下輸出) :

```
# Put date and time in prompt; update every 60 seconds:
trap 'PS1=`date "+%a %D %H:%M%n" `\'
$ \ 5
while :
do
    sleep 60
    kill -5 $$
done &
promptpid=$!
```

現在，每隔一分鐘當你輸入一個指令後，你的提示訊息就會改變：

```
Mon 02/17/92 08:59
$ cc bigprog.c
undefined symbol                first referenced in file
xputc                            bigprog.o
ld fatal: Symbol referencing errors.
Mon 02/17/92 08:59
$ ls
bigprog.c
bigprog.o
Mon 02/17/92 09:00
$
```

提示訊息的格式可隨你改變，這個範例中用的是兩行的提示訊息，並有一個反斜線（\）以保護 trap 不受跳行及格式的影響；如果是單行的提示訊息就容易設計得多了。date 的線上使用手冊列出了可以放入提示訊息的資料。

這個設定在背景啟動了一個 while 迴圈 (44.10)，promptpid 變數儲存此背景 Shell 的行程 ID 號碼 (38.03)，在你登出之前，你應該 kill (38.10) 這個迴圈，你可以在提示訊息後輸入這個指令：

```
kill $promptpid
```

或是放在一個當你登出時會執行 (3.02) 的檔案。

-- JP

## 7.13 bash：在提示訊息前執行的指令

bash 可以在印出提示訊息之前執行一個或多個指令，這個指令並不需要設定提示訊息，它只不過是在印出提示訊息之前被執行而已，這個指令可以做一些系統的檢查工作、重新設定 Shell 變數、或任何你可以在 Shell 提示訊息輸入的指令，如果這些指令執行起來很慢，你可以把這些指令儲存在 Shell 提示訊息 PROMPT\_COMMAND，不過這又會使提示訊息變慢。

下面是我用於 bash 設定檔案的一段程式碼，不過看起來有點愚蠢：

```
PROMPT_COMMAND='
# Save old $IFS; set IFS to tab:
IFS 35.21 OIFS="$IFS"; IFS=" "
# Put x in $1, face in $2, explanation[s] in $3[, $4, ...]:
set 44.19 set x `smiley`
smiley 51.12 # Put face into $face and explanation(s) into $explan:
face="$2"; shift 2; explan="$*"
# Restore shell environment:
shift $#; IFS="$OIFS"

# Prompt I use (includes the latest $face):
PS1='\u@\h $face '
```

第一個部分是一串儲存於 PROMPT\_COMMAND 的 Shell 指令，它們被一對單引號（' '）所包圍，一個在第一行（= 的後面），另一個在 IFS 被重新設定的後面，這串指令是在任何提示訊息之前執行，而且也在設定每個提示訊息之前，設定兩個 Shell 變數 - \$face 和 \$explan，提示訊息是在最後一行所設定的，它包含了 \$face 的值。

下面就是這個看起來有點可笑的設定方式所呈現的模樣，注意，提示訊息會在 PROMPT\_COMMAND 重新設定 \$face 和 \$explan 時立刻改變，如果我想知道那張臉是什麼意義，可以輸入 echo "\$explan"：

```
jerry@ruby :~() echo "$explan"
normal smiling face with a moustache
jerry@ruby +<||-) vi proj.cc
...
jerry@ruby :-0 echo "$explan"
Mr. Bill
    Wow!
    ohh, big mouth, Mick Jagger
    uh oh
jerry@ruby :-) < g++ -Wall proj.cc
...
```

(它看起來比 `psychoanalyze-pi nhead` (32.13) 還沒用，但讓它持續工作是件很有趣的事。) 現在我必須一本正經的告訴你：`PROMPT_COMMAND` 不一定要用於設定提示訊息，你可以用它來執行任何的指令，如果在 `PROMPT_COMMAND` 的指令將輸入寫入標準輸出或標準錯誤輸出，你將在看到提示訊息之前看到這段文字。

-- JP

