附錄D

Shell 參考表

在這個附錄中，我們將展示四種shell的功能，即Bourne shell(**sh**)，C shell(**csh**)，Korn shell(**ksh**)和**bash**(**bash**)。**sh**是標準的程式語言，而**csh**因為它的一些受爭議之歷史指令取代的進階功能部分，所以仍然被包含進來。**ksh** 在這部分支援指令行的編輯功能，而且它也提供了許多進階和有用的功能。**bash** 提供了它們之中最好的部分，包括**sh** 的程式語言及**csh**的交談式功能。這些相關功能都已經在第八、十、十七、十八和十九章分別介紹過了；所以它們在這裡會用聯合的、總結和比較的方式展示。

指令執行的方式

指令可以用許多方式被執行，亦即循序的、條件式或使用來自另一指令的輸入。手稿可以執行在相同或不同的sub-shell。您也可以跳過別名及函數並且執行外部指令或內建指令。請注意nohup 和command 並沒有內建到**sh** 中，但是它們常以外部指令方式出現（像在 Solaris 中）。此處的*cmd*、*cmd1*和*cmd2*都代表指令。

指令	那些Shell 支援	指令執行方式
cmd1 ; *cmd2*	All	*cmd2* 在 *cmd1* 之後
cmd1 | *cmd2*	All	*cmd2* 與輸入來自 *cmd1*的標準輸出
cmd1 && *cmd2*	All	如果 *cmd1*執行成功才會執行 *cmd2*
cmd1|| *cmd2*	All	如果 *cmd1*執行失敗才會執行 *cmd2*
cmd1 `*cmd2*`	All	*cmd1* 與來自 *cmd2* 標準輸出的參數
cmd1 $(*cmd2*)	**ksh**, **bash**	同上
(*cmd1* ; *cmd2*)	All	先*cmd1* 然後 *cmd2*，此兩者在sub-shell 中
{ *cmd1* ; *cmd2* ; }	**sh**, **ksh**, **bash**	先*cmd1* 然後 cmd2，此兩者在目前shell 中
cmd &	All	cmd 在背景執行（**csh**可允許登出）
nohup *cmd&*	**ksh**, **csh**, **bash**	*cmd* 在背景執行（可允許登出）
. *script*	**sh**, **ksh**, **bash**	*script*執行在目前 shell（**bash**也可使用**source**）
source *script*	**csh**, **bash**	*script*執行在目前 shell
cmd	All	外部指令或內建指令會忽略同名稱的別名
command *cmd*	**ksh**, **csh**, **bash**	外部指令或內建指令會忽略同名稱的函數（也應用在**csh**, **bash**的別名）

轉 向

shell 使用三個主要的資料流來代表輸入、輸出和錯誤訊息。您可以防止檔案被覆寫，也可以將輸出和錯誤訊息資料流合併在一起以便可以同時收集它們。

sh，ksh，bash	csh	意義
>*file*	> *file*	輸出轉向*file*
>>*file*	>> *file*	輸出附加在*file*之後
< *file*	< *file*	從*file*讀取
<< *mark*	<< *mark*	從以下的行讀入直到讀到字串*mark*（此地文件）
set -o noclobber	set noclobber	使用>和>>時不要覆寫檔案（sh不適用）
set +o noclobber	unset noclobber	反轉 noclobber的定義（sh不適用）
>| *file*	>! *file*	將輸出轉向*file*即使有設定noclobber（sh不適用）
2> *file*	—	將錯誤訊息轉向*file*
> *file*1 2> *file*2	—	輸出轉向file1而錯誤訊息轉向*file2*
> *file* 2>&1	>& *file*	同時將輸出和錯誤訊息轉向*file*
2 > *file* 1>&2	>& *file*	同上
>| *file* 2>&1	>&! *file*	同時將輸出和錯誤訊息轉向*file*即使有設定noclobber（sh不適用）
>& *n*	—	輸出轉向到*file*的描述器*n*

檔名的中介字元

檔名的中介字元被區分成兩個群組。在第一個群組的項目對所有shell都是通用，並且被使用在那些以一個或多個檔名當參數的指令上。它們也被用在case的條件式以比對 strings，或在for迴圈中比對目前目錄的檔名。請注意檔名最前面的點(.）必須被明白的比對。

sh，ksh，bash	csh	比對
*	*	任何數目的字元包括空的(null)
?	?	單一個字元
.???*	.???*	檔名由點起頭後面接著包含至少三個字元
[*ijk*]	[*ijk*]	單一個字元，即*i,j,k*其中之一（可以用a-z,0-9的區間）
[*!ijk*]	—	單一個字元但不是*i,j,k*（允許區間）
.[!oc*]	—	檔案的副檔名不是C程式也不是目地(object)檔

第二個群組的功能並不能應用在sh，而且有一些也不能用在csh。如果您使用bash，在您放置把否定運算元!放在樣式之前必須先執行 shopt -s extglob 設定（下面有三個例子)。

ksh, bash	csh	意義
{*p1,p2,p3*}	{*p1,p2,p3*}	符合檔案*p1,p2*或*p3*
calc.{sh,pl,awk}	calc.{sh,pl,awk}	符合檔案calc.sh, calc.pl, calc.awk
!(*pat*)	—	符合所有檔案除了pat之外
!(*p1*|*p2*|*p3*)	—	符合所有檔案除了*p1,p2*或*p3*之外
!(*.GIF|*.JPEG)	—	符合所有檔案除了以.GIF或.JPEG結尾
~	~	符合目前使用者的家目錄
vi ~/.alias	vi ~/.alias	編輯位於家目錄的.alias
cd ~user	cd ~user	切換到使用者 user的家目錄
~-	—	前一個目錄
vi ~-/.profile	—	編輯位於之前造訪過目錄的.profile
cd ~-	—	切換到前一個目錄
cd -	—	同上

處理變數和陣列

UNIX 變數並沒有類別(type)，這也是為什麼變數包含數字值會被當成字串和數字看待。這節展示了數學函數、字串處理、陣列和特別的參數。有一些應用在ksh之*ksh93*版本的功能也許不能在您的系統被找到。此處，*var* 代表變數。

變數的啟始

變數可以被區分成兩種類別，即本地或全區域的（環境的）。csh使用不同的敘述句來處理這些變數，而且能讓本地和全區域的變數使用同一個變數名稱。

sh, ksh, bash	csh	意義
var=value	set *var=value*	指定 *value* 給本地變數 *var*
var=value	setenv *var value*	指定 *value* 給環境變數 *var*
export *var*	setenv *var value*	將 *var* 的*value* 傳給 sub-shell
export	setenv	顯示匯出/環境的變數
unset *var*	unset *var*	刪除本地變數*var*
unset *var*	unsetenv var	刪除環境變數*var*
read *var*	set *var* = $<	將鍵盤輸入讀進 *var*

數值的運算

除了sh之外，其他的shell使用+，－，*，/等運算子來執行數學運算。它們也使用遞增和遞減的運算子（++和--）。sh的使用者必須靠expr來執行數學運算和一些字串處理功能。POSIX指定使用(())而非let，但是最後一個範例沒有使用$，此為不被POSIX建議的格式。這種格式只能被用在新版的bash或*ksh93*。

ksh, bash	csh	**意義**
let z=x+y	**@ z = x + y**	將變數**z**指定為**x**和 **y**的總合
let x=x+1	**@ x++**	將**x**加**1**
z=$((x+y))	**@ z = $x + $y**	將**z**指定為**x**和 **y**的總合
z=$((x/y))	**@ z = $x / $y**	將變數z指定為**$x/$y**的餘數
((x++))	**@ x++**	將**x**加**1**(bash, *ksh93*)

字串的操作

這也被區分成兩個群組。第一個群組以不同的方式計算一個變數*var*，看它是否被設成一個非空的字串。第二個群組顯示了進階的**ksh**和**bash**字串處理功能。這個群組的前六個項目從一個字串中置換或刪除一個樣式*pat*，其中*pat* 可以是一個萬用字元的表示式。

sh, ksh, bash	csh	**被計算成**
${*var:+pat*}	—	*pat* 如果*var* 被指定成非空(non-null)值
${*var:-pat*}	—	*pat*如果*var* 未被指定或是空值
${*var:=pat*}	—	同上，但將*var*設成 *pat*
${*var:?pat*}	—	$*var* 如果有設定，否則印出*pat*並且結束手稿
—	${?*var* }	0如果*var*未設定，否則為1

sh, ksh, bash	**被計算成**
${*var#pat*}	從開頭比對到*pat*的部分，再刪除其中最短的區段後剩下的*var*的區段
${*var##pat*}	同上但是刪除其中最長的區段
${*var%pat*}	從尾端比對到*pat*的部分，再刪除其中最短的區段後剩下的*var*的區段
${*var%%pat*}	同上但是刪除其中最長的區段
${*var / pat*}	在刪除第一個比對到*pat*之後剩下的*var*(bash, *ksh93*)
${*var*/*s1*/*s2*}	在置換第一個出現的*s1*成為*s2*之後的*var*(bash, *ksh93*)
${**#var**}	*var* 的長度(bash, *ksh93*)
${*var:x:y*}	從*var*的位置*x*開始取出長度為*y*的子字串

陣列的操作

在**ksh**與**bash**中陣列的索引都是從0開始，而**csh**卻是1。您可以在**ksh**和**bash**中單獨設定個別的陣列元素，但是**csh**允許您存取一個區間的元素或最後一個陣列元素卻不需要知道它的索引。

ksh, bash	csh	意義
var=(*val1 val2* ...)	set *var*=(*val1 val2* ...)	設定陣列變數*var*的值為*val1, val2*,等
unset *var*	unset *var*	刪除*var*
var[n]=*value*	—	指定第*n*個元素的值
${*var*[n]}	$*var*[*]	第*n*個元素
${*var*[*]}	$*var*	所有的元素
${*var*[@]}	$*var*	同上
${#*var*[@]}	$#*var*	有多少元素
—	$*var*[$#*var*]	最後一個元素
—	$*var*[m-n]	第*m*到*n*個元素（也可使用*m*- 或 *n*-）

特別的參數和變數

shell使用許多自動設定的特別參數。指令行傳進shell手稿的參數可被手稿內部的一些參數讀入。除了csh以外的shell可以使用set敘述句設定它們的值。其他的就是與程序相關的部分。csh 存取最後一個指令行參數的方式比其他shell都還直接。

ksh, bash	csh	意義
$1,$2, etc.	$argv[1],$argv[2],etc.	指令行的參數編號 1, 2, ...等
$*	$* 或 $argv[*]	所有指令行的參數
$#	$#argv	有多少指令行的參數
—	$argv[$#argv]	指令行參數的最後一個，其他shell 需要使用 *var*=`eval echo \\$$#`
$0	$0	被執行的指令名稱
“$@”	—	將引號括起來的多字組的參數當成一個參數
$?	$status	最後一個指令的結束狀態
$$	$$	目前shell 的PID
$!	—	最後一個背景工作的PID
—	$<	從鍵盤讀進的輸入

歷史指令

歷史指令在sh中並不支援，與csh和bash比較起來，這個功能在ksh中使用有一些限制。然而，ksh允許在指令行直接使用類似vi或 emacs的指令編輯。此處bash與csh的功能幾乎是一樣的，但是bash卻提供兩者中最好的因為它也提供線上(in-line)的編輯。這裡的*stg*，*stg1*，和*stg2*代表字串。

重複的事件

csh, bash	ksh	意義
history *n*	**history** *-n*	顯示最近執行過的指令或n個執行過的指令
!!	*r*	重做最後一個指令
!*n*	*r n*	重做事件號碼*n*
!*n*:p	—	顯示事件號碼*n*但不執行它
!*-n*	*r -n*	重做第*n*個之前的事件
!-2	*r* -2	重做前一個的再前一個指令
!*com*	*r com*	重做前一個由*com*起頭的指令
!*?stg?*	—	重做前一個包含*stg*字串的指令
!?mtime?:p	—	顯示前一個包含mtime字串的指令但不執行它

修改先前的指令行

csh, bash	ksh	執行
^stg1^stg2	**r** *stg1=stg2*	以*stg2*替換*stg1*後的先前指令
!*com*:s/*stg1*/*stg2*	**r** *com stg1=stg2*	以*stg2*替換第一個出現之*stg1*後的前一個由*com*起頭的指令
!com:gs/*stg1*/*stg2*	—	同上，不過是以全區域方式將*stg2*替換成*stg1*
!! *arg*	**r** *arg*	前一個指令並將參數*arg*附加到指令行
!cut | sort	**r cut | sort**	最後一個cut指令但之後將結果pipe給sort指令
cmd !$	*cmd $_*	*cmd* 與前一個指令的最後一個參數（由bash使用的$_也是一樣）

使用前一個指令個別的參數

這種多功能的指令和擷取參數的功能讓您可以執行一個新的，或前一個指令加上先前指令的任何參數。前一個指令的參數也可以當作命令來執行。參數可用一般的 **:***n* 功能來取出，其中*n* 是單一的數字或區間。**:^**（或**:1**）和 **:$**參考到第一個

csh, bash	執行
cmd !*	*cmd* 使用前一個指令的所有參數
cmd !***$**	*cmd* 使用前一個指令的最後一個參數
cmd **$_**	同上（只適用bash）
cmd !:*n*	*cmd* 使用前一個指令的第*n*個參數
gzip !:4	gzip 使用前一個指令的第4個參數
cmd !:*m-n*	*cmd* 使用前一個指令的第*m*個到第*n*個參數
compress !ls:3-$	compress 使用前一個ls的前兩個參數
cmd !:*m-*	*cmd* 使用前一個指令的第*m*個到最後一個參數
cmd !*m*:*n*	*cmd* 使用事件*m*的第*n*個參數
!*m1*:0 !*m2* :*n*	事件*m1*的指令使用事件*m2*的第*n*個參數
!!:*n*	前一個指令的第*n*個參數當作指令

及最後一個參數，但:0只獨立出命令名稱。這種功能不能使用在ksh，只能用在 csh 及 bash。

捨去路徑名稱

假如之前執行指令的參數是一個檔案名稱，csh 與 bash 能讓您擷取出檔案名稱中不同的組成元件。副檔名的部分也可從它們的基礎檔名中被分離出來而個別的被取出。csh 也允許使用變數在這些機制上，而bash只能使用前一個指令的參數來工作。

csh, bash	執行
cmd !$::h	*cmd* 使用前一個指令的最後一個參數之前端（移除路徑中的檔名部分）
cmd !$:t	*cmd* 使用前一個指令的最後一個參數之尾端（留下檔名部分）
cp !$!$:t.txt	cp 複製前一個指令的最後一個參數並附加.txt的副檔名
cmd !$:r	*cmd* 使用前一個指令的最後一個參數之根部（移除副檔名之後剩下的檔名）
echo !$:*e*	echo 使用前一個指令的最後一個參數的副檔名部分（沒有點的部分）
echo $*var*:*h*	echo 來顯示儲存在變數var中路徑的前端（只適用csh）

線上(in-line)指令編輯

ksh 及 bash允許之前的執行過的指令可以被顯示在螢幕上。然後您就能使用內建的類似vi與emacs的編輯功能來修改它們。一次只有一種模式可以被啟動，並藉由使用set -o vi 或 set -o emac 的指令來設定。您可以在這些模式下使用的基本指令被顯示在表 17.4。

環境變數

shell 的工作環境是由它啟始檔中的項目來決定，它的變數及模式也在這些檔案中被設定。ksh 及 bash 簡單的提供了它們之中所有最佳的工作環境。被這些shell（以及在登出前被使用的）使用的起始手稿顯示在表17.1的最後三個項目。

環境／內建的變數

這些變數傳統上都是由大寫字母來表示。然而，csh提供相同的變數同時有大寫和小寫的名稱。它提供小寫字母的變數給它自己使用，但是外部程式使用大寫字母的變數。它在這裡顯示的四個變數實際上是陣列。改變其中之一的設定通常會影響到其他的，但也不全然如此。ksh和 bash使用許多額外的變數。

sh, ksh, bash	csh	意義
CDPATH	cdpath=(*dir1* dir2 ..)	當使用cd *dir*時會查詢的目錄列表
HOME	home	使用者的家目錄
IFS	—	內部欄位的分隔符號（由set 來使用）
LOGNAME	user	使用者的登入名稱
MAIL	mail=(n files)	使用者的電子郵件檔案（在csh中的一個或多個*files*）
MAILCHECK	mail=(n files)	多久檢查一次電子郵件信箱（*n*是秒數）
MAILPATH	—	由一個或多個：分隔的電子郵件信箱檔案；也能包含要列被列印的訊息
PATH	path=(dir1 dir2 ..)	當指令要被執行時需搜尋的目錄列表
PS1	prompt	主要的命令提示字串
PS2	—	第二行的命令提示字串（在csh通常是?）
SHELL	shell	使用者的登入shell或從程式vi或mail跳脫時用的shell
TERM	term	終端機的類型
USER	user	使用者的登入名稱(bash)
BASH_ENV	—	由sub-shell讀入的環境檔案(bash)
EDITOR	—	在ksh中用來做指令行線上編輯的編輯器（在VISUAL之後被讀入，bash中較少用）
ENV	—	由sub-shell讀入的環境檔案 (ksh)
HISTFILE	—	歷史檔案（csh使用 .history）
HISTFILESIZE	savehist	有多少執行過的指令在登出後會儲存在歷史檔(ksh使用HISTSIZE)
HISTFILE	history	有多少執行過的指令會儲存在記憶體（ksh會放在歷史檔）
LINENO	—	在手稿程式或函數中的行號
OLDPWD	—	前一個工作目錄
PPID	—	目前shell父程序的PID
PWD	cwd	目前的工作目錄
REPLY	—	當使用read 指令又沒有提供參數時會被讀入的變數
VISUAL	—	在ksh中用來做指令行線上編輯的編輯器（會覆寫EDITOR；沒有被使用在bash）

shell 模式

在 ksh 與 bash 中的模式是由 set -o 加上一個關鍵字當參數（例如, set -o vi）來設定。csh只是簡單的使用set 加上一個關鍵字（不需要 =；例如, set filec）這些關鍵字被表列在下面。關閉這些設定要使用set +o *keyword*(ksh), unset *keyword*(csh)。許多的這些設定在ksh 和 bash 中也可使用 set加上單一的字元選項來關閉。

sh, ksh, bash	csh	意義
emacs	—	設定線上指令行編輯器為emacs
vi	—	設定線上指令行編輯器為 vi
ignoreeof	ignoreeof	忽略eof 字元做登出動作，只允許exit
noclobber	noclobber	當使用>或>>時，防止檔案被覆寫
—	filec	開啟檔名展開
noglob	noglob	不展開萬用字元，把它們當作一般字元
nolog	—	不要將函數定義儲存在歷史檔(ksh)
notify	notify	當工作完成時立即通知，而且不在下一個提示符號上（ksh不支援）
verbose	—	當手稿被執行時，顯示它的每一行
xtrace	—	當指令執行時顯示指令行，之前並附加+符號（相等於 set -x）

比較測試

這些 shell 提供字串比較，整數和許多測試檔案屬性的廣泛支援。它們被使用在 test 敘述句或它的對等符號[]中，而通常它們會與if，while和until的敘述句放在一起。以下就是如何使用它們的方式：

```
while [ $x -lt 10] ; do                                    sh, ksh, bash
if ( -e .profile ) then                                    csh
```

整數測試

sh, ksh, bash	csh	意義
-eq	==	等於
-ne	!=	不等於
-gt	>	大於
-ge	>=	大於或等於
-lt	<	小於
-le	<=	小於或等於

字串測試

最後四個範例明白顯示使用[[ ]]運算子。它們允許使用萬用字元作字串的比對，和以條件運算子&&與||作為它們之中多重條件式的計算。這些形式可以工作在Korn shell的*ksh93*版本及bash之較新版本（由SuSE Linux 6.4所提供）。

sh, ksh, bash	csh	**計算後為真假如...**
stg	*stg*	字串*stg*被指定且不為空值
-n *stg*	—	*stg*不為空字串
-z *stg*	—	*stg*為空字串
stg1 = *stg2*	*stg1* == *stg2*	*stg1*等於 *stg2*
stg1 != *stg2*	*stg1* != *stg2*	*stg1*不等於 *stg2*
[[*stg* == *exp*]]	*stg* =~*exp*	*stg* = *exp*, 此處*exp*可以是萬用樣式（sh 不支援）
[[*stg* != *exp*]]	*stg* !~*exp*	*stg* 不符合萬用字元表示式 *exp* （sh 不支援）
[[*stg1* < *stg2*]]	—	*stg1* 的ASCII 排列順序在 *stg2* 之前（sh 不支援）
[[*stg1* > *stg2*]]	—	*stg1* 的ASCII 排列順序在 *stg2* 之後（sh 不支援）

檔案測試

sh, ksh, bash	csh	**計算後為真假如...**
-e *file*	**-e** *file*	檔案存在（**sh** 不支援）
-f *file*	**-f** *file*	檔案是個一般檔案
-d *file*	**-d** *file*	檔案是個目錄
-b *file*	—	檔案是個區塊特殊檔案
-c *file*	—	檔案是個字元特殊檔案
-r *file*	**-r** *file*	檔案可被讀取
-w *file*	**-w** *file*	檔案可被寫入
-x *file*	**-x** *file*	檔案可被執行
-0 *file*	**-o** *file*	檔案被使用者擁有（**sh** 不支援）
-s *file*	**!** **-z** *file*	檔案的大小大於0
-L *file*	—	檔案是個符號連結檔（**ksh**, **bash**）
-u *file*	—	檔案有 SUID 位元設定
-k *file*	—	檔案有 sticky 位元設定
file1 -nt *file2*	—	檔案1比檔案2還新（**ksh**, **bash**）
file1 -ot *file2*	—	檔案1比檔案2還舊（**ksh**, **bash**）
file1 -ef *file2*	—	檔案1被連結到檔案2（**ksh**, **bash**）

內部指令

最後一節展示這些shell中的內部指令。多重選項是以|隔開顯示。有些結構像select, function和typeset並不屬於POSIX部分的規格所以被刪除。然而, 卻也有替換這些結構的相關指令。至於*cmd*, *var*, *exp*和*stg*等代表符號跟它們原有的意義相同。

指令fg, bg, kill, stop和wait使用 jobid, 它是一個%符號之後加上工作號碼（**%*n***）, 一個指令由一個字串**s**起頭（**%*s***）, 一個指令包含一個字串***s***（**%?*s***）, 或是

目前的工作（%%）。

指令	那些Shell 支援	意義
. *file*	sh, ksh, bash	在目前shell中執行file
[ ]	sh, ksh, bash	與 test敘述句同義
[[ ]]	ksh, bash	同上，但允許較複雜的測試
alias *name*	ksh, csh, bash	顯示所有別名的定義，或*name*的別名定義如果*name*有指定
alias *name=cmd*	ksh, bash	定義*name* 為*cmd*的別名
alias *name cmd*	csh	同上
bg *jobids*	ksh, csh, bash	將目前工作或*jobids*移到背景執行
break *n*	All	離開目前或第*n*個（csh 不支援） for, while, 或until所包圍的迴圈
case string in pat1)*commands*1;; ..... patn)*commandsn*;; esac	sh, ksh, bash	執行 *commandsn* 如果*string*有比對到樣式*patn*。 *patn*可以是萬用字元的表示式。在單一選項的多重樣式可用|隔開。在選項中的*代表之前沒有比對到的所有部分。
cd	All	切換到家目錄
cd *dir*	All	切換到目錄 *dir*
cd -	ksh, bash	切換到前一個工作目錄
cd *~/usr*	ksh, csh, bash	切換到使用者 usr的家目錄
command *cmd*	ksh, bash	忽略別名及函數直接執行*cmd*
continue *n*	All	從目前或第*n*個（csh除外） for, while,或until所包圍的迴圈中最前面的指令開始執行
echo *ops stg*	All	顯示訊息*stg*。csh並不支援跳脫程序，而bash需要 -e選項才能支援它們。
eval *cmd*	All	計算*cmd*兩次。當變數包含特殊字元時就會需要。
exec *cmd*	All	將目前的shell置換成*cmd*
exec *n> file*	sh, ksh, bash	將檔案描述器*n*轉向到*file*
export	sh, ksh, bash	顯示所有被匯出的變數
export *var*	sh, ksh, bash	將$*var*的值傳進sub-shell
exit *n*	All	終止目前的shell並傳回結束狀態為0或*n* 如果*n*有指定
fg *jobid*	ksh, csh, bash	將目前工作或jobids移到前景執行
for *var in list*; do	sh, ksh, bash	將list中的每一個值指定給變數*var*
commands		並執行do到done之間的*commands*。
done		請參考break 和 *continue*。
foreach *var*(*list*) commands end	csh	同上，但執行foreach到end之間的*commands*。請參考break 和 continue。
functions	ksh	顯示所有的函數定義

function_name() { *commands* }	sh, ksh, bash	將函數*function_name*定義成一組的*commands*。它接受位置參數**$1**, $2 等，並且可選擇利用 **return** 敘述句來傳回一整數值。
goto stg	csh	執行有字串*stg*起頭及附加：之行後面的指令。
history *-n*	ksh, bash	顯示立即的事件或最後*n*個事件的歷史指令列表，假如*n*有指定同上
history *n*	csh	
if condition ; **then** *commands* *options* **fi**	sh, ksh, bash	條件敘述句。假如*condition*為真，那麼就執行*commands*，options包含使用**elif**和 **else**之另外的測試條件式。請參考18.8節。
if (condition) **then** *commands* *options* endif	csh	同上，但*options* 使用else **if** 和 **else**敘述句。請參考附錄A
jobs *-l*	ksh, csh, bash	顯示所有工作，如果有指定 -l ，也會顯示PIDs
kill *-n jobids|pids*	ksh, csh, bash	使用*jobids*或*pids*移除（終止）一個或多個程序，並可選用信號*n*。-9 確定要將程序殺掉。
let *var=exp*	ksh, bash	將*exp*的計算結果指定給變數*var*。*exp*使用運算子+, -, *, / 和%。
logout	csh	結束登入shell
nice + n|-n *cmd*	All	預設使用較低的優先權來執行指令*cmd*，或由*n*的單位更改優先權，假如*n*有指定
nohup *cmd*	ksh, csh	即使登出後在背景繼續執行*cmd*
notify	csh	當背景工作完成時立即回報
onintr -|*label*	csh	處理中斷信號2。手稿程式會跳到label執行或忽略信號（-）。
printf *fmt values*	ksh, bash	使用類似C的格式字串*fmt*來顯示經過格式化的values
pwd	All	顯示目前工作目錄
repeat *n cmd*	csh	重複執行*cmd n*次
read	ksh, bash	將鍵盤輸入讀進手稿中的變數**REPLY**
read *var*	sh, ksh, bash	同上，但輸入的值指定給 *var*
set	All	顯示所有的（在csh中是本地）變數
set *-o mode*	ksh, bash	設定一個shell模式，請參考shell模式
set *exp*	sh, ksh, bash	設定位置參數 **$1**, **$2**等的值成為*exp*中的字組
set *var*	csh	設定一個shell模式，請參考shell模式
set *var* = *$<*	csh	讀進鍵盤輸入到手稿中的變數*var*
set *var = value*	csh	指定本地變數*var*的值
setenv	csh	顯示所有的環境變數
setenv *var value*	csh	指定環境變數*var*的值為*value*
shift	All	將位置參數往左移；**$2** 變成 **$1**，以此類推。
shift n	sh, ksh, bash	同上，但是往左移*n*個位置
shift *var*	csh	將陣列*var*的元素向左移

source *file*	**csh**, **bash**	在目前shell中執行file
stop *jobids*	**ksh**, **csh**, **bash**	暫停有 *jobids* 的背景工作
suspend	**ksh**, **csh**, **bash**	暫停目前的前景工作（相等於 *[Ctrl-z]*）
switch(*string*) **case** *pat1*: *command1* **breaksw** ..... **case** *patn*: *commandn* **breaksw** **endsw**	**csh**	執行 *commandsn* 如果*string*有比對到樣式 *patn*。 **switch**會由上往下依序執行指令直到發現**breaksw**敘述句。最後一個選項可用 default 來取代，它能符合之前沒有比對到的所有部分
test	**sh**, **ksh**, **bash**	與 [ ] 同義
trap *cmds sigs*	**sh**, **ksh**, **bash**	信號的處理器，當它接收到信號*sigs*時會執行*cmds*。如果*cmds*是 '' 則手稿程式會忽略信號。信號0會在目前shell結束時執行 *cmds*。
type *cmd*	All	找出*cmd*是外部、內部指令，別名或函數
ulimit *ops*	All	設定或顯示一個資源的限制
umask *nnn*	All	顯示檔案建立時的遮罩(mask)或設定遮罩為八進位值*nnn*
unalias *name*	**ksh**, **csh**, **bash**	刪除別名的定義 *name*
unset *name*	All	刪除變數或陣列 *name*
unset -f *name*	**sh**, **ksh**, **bash**	刪除函數name
until *condition* ; **do** *commands* **done**	**sh**, **ksh**, **bash**	執行 **do**與 **done** 之間的 *commands*直到 *condition* 為真。請參考**break** 和 **continue**。
wait	All	暫停執行直到所有的背景工作完成
wait *pids*	**sh**, **ksh**, **bash**	暫停執行直到這些PID為*pids* 的工作完成
while *condition* ; **do** *commands* **done**	**sh**, **ksh**, **bash**	只要 *condition* 為真，即執行**while**與 **done** 之間的 *commands*。請參考**break** 和 **continue**。
while (*condition*) *commands* **end**	**csh**	只要 *condition* 為真，即執行**while**與**end**之間的 *commands*。請參考**break** 和 **continue**。