Power Shell：B

**目錄**

[Power Shell：指令操作 1](#_Toc206004370)

[1. 認識 Shell 與Shell Script 3](#_Toc206004371)

[1.1 Shell Script 3](#_Toc206004372)

[1.2 Linux 中常見的 Shell： 3](#_Toc206004373)

[2. 認識 Bash 3](#_Toc206004374)

[2.1 Bash 的環境變數 4](#_Toc206004375)

[2.2 常見環境變數 4](#_Toc206004376)

[3. 變數 4](#_Toc206004377)

[3.1 範例：初始化變數hello 4](#_Toc206004378)

[3.2 範例：變數沒有一定資料類型 4](#_Toc206004379)

[3.3 範例：空變數自動轉成整數 5](#_Toc206004380)

[3.4 區域變數 5](#_Toc206004381)

[3.5 參數變數 5](#_Toc206004382)

[3.6 接收傳遞 script 或函式的參數 (Receiving Arguments) 6](#_Toc206004383)

[3.7 取消定義變數 (Unsetting Variables) 7](#_Toc206004384)

[3.8 查詢變數是否被定義 (Checking if a Variable is Set) 7](#_Toc206004385)

[3.9 在變數未被定義時，使用預設值或顯示錯誤 7](#_Toc206004386)

[3.10 將檔案的內容寫入變數 (Assigning File Content to a Variable) 8](#_Toc206004387)

[3.11 將變數的值設為唯讀 (Read-only Variables) 8](#_Toc206004388)

[4. 運算子 8](#_Toc206004389)

[5. 判斷式 10](#_Toc206004390)

[5.1 if 判斷語法 10](#_Toc206004391)

[5.2 範例：密碼比對 10](#_Toc206004392)

[5.3 範例：成績判斷 (if-elif-else) 10](#_Toc206004393)

[6. 基本的數值運算 11](#_Toc206004394)

[6.1 算數擴展運用 11](#_Toc206004395)

[6.2 位元左移或右移 11](#_Toc206004396)

[6.3 and 與 or 的用法 11](#_Toc206004397)

[6.4 let 的用法 12](#_Toc206004398)

[6.5 產生亂數 12](#_Toc206004399)

[7. 控制流程 12](#_Toc206004400)

[7.1 控制流程基礎 12](#_Toc206004401)

[7.2 在 IF 中寫多條件 12](#_Toc206004402)

[7.3 在 IF 條件逆轉 13](#_Toc206004403)

[7.4 使用&&與||處理 13](#_Toc206004404)

[7.5 case 判斷語法 13](#_Toc206004405)

[7.6 範例：輸入 1~5 對應文字 13](#_Toc206004406)

[7.7 使用 case 根據字串模式進行判斷 14](#_Toc206004407)

[8. 迴圈 14](#_Toc206004408)

[8.1 for 迴圈 14](#_Toc206004409)

[8.2 while 迴圈 14](#_Toc206004410)

[8.3 在條件成立期間內，執行迴圈內容 15](#_Toc206004411)

[8.4 無窮迴圈的設計 15](#_Toc206004412)

[8.5 使用 while 計算平方和： 15](#_Toc206004413)

[8.6 跳到下一次的迴圈執行：continue 15](#_Toc206004414)

[8.7 中斷迴圈進 16](#_Toc206004415)

[8.8 行下一步處理：break 16](#_Toc206004416)

[8.9 對所有的參數進行重複處理 16](#_Toc206004417)

[8.10 對指令的輸出結果做重複處理 16](#_Toc206004418)

[8.11 對檔名做重複處理 17](#_Toc206004419)

[8.12 在背景進行重複處理 17](#_Toc206004420)

[8.13 將重複處理的結果存為檔案 17](#_Toc206004421)

[8.14 使用 select 顯示選單並做迴圈處理 17](#_Toc206004422)

[9. 函數 18](#_Toc206004423)

[9.1 定義函式 (Defining a Function) 18](#_Toc206004424)

[9.2 範例：使用函數 18](#_Toc206004425)

[9.3 取消定義函式 (Unsetting a Function) 18](#_Toc206004426)

[9.4 在呼叫函式時傳遞參數 (Passing Arguments to a Function) 19](#_Toc206004427)

[9.5 在函式內部取得結果 (Getting a Return Value from a Function) 19](#_Toc206004428)

[9.6 遞迴：呼叫函式本身 (Recursive Function) 20](#_Toc206004429)

[9.7 呼叫其他檔案中定義的函式 (Sourcing Functions from Other Files) 20](#_Toc206004430)

[10. 使用字串 (Using Strings) 21](#_Toc206004431)

[10.1 寫一個橫跨多行的字串 (Creating a Multi-line String) 21](#_Toc206004432)

[10.2 跳脫字元使用方法 (Using Escape Characters) 21](#_Toc206004433)

[10.3 字串切片 22](#_Toc206004434)

[11. 檔案內的字串處理 22](#_Toc206004435)

[11.1 查詢字串的長度 (Shell 內建功能) 22](#_Toc206004436)

[11.2 轉換字串的大小寫 (使用 tr) 22](#_Toc206004437)

[11.3 查詢檔案內的字串 (使用 grep) 22](#_Toc206004438)

[11.4 替換字串 (使用 sed) 23](#_Toc206004439)

[11.5 抽取出字串 (使用 awk ) 23](#_Toc206004440)

[11.6 抽取出字串 (使用cut) 23](#_Toc206004441)

[12. 檔案的欄位處理 (Field Processing in Files) 23](#_Toc206004442)

[12.1 更換逗點分隔的欄位順序 23](#_Toc206004443)

[12.2 增加一個固定的欄位 24](#_Toc206004444)

[12.3 變更欄位的分隔符號 24](#_Toc206004445)

[12.4 取出符合條件的內容 24](#_Toc206004446)

[13. 檔案操作技巧 24](#_Toc206004447)

[13.1 從檔名取得檔名或資料夾名稱 24](#_Toc206004448)

[13.2 產生內容為空的檔案 25](#_Toc206004449)

[13.3 變更檔案的更新日期 25](#_Toc206004450)

[13.4 使用暫存檔 25](#_Toc206004451)

[13.5 查詢指令所在的位置 25](#_Toc206004452)

[13.6 列出資料夾內的檔案 25](#_Toc206004453)

[13.7 搜尋檔名與檔案內容 26](#_Toc206004454)

[13.8 找出檔名與樣式符合的檔案 26](#_Toc206004455)

[13.9 讀入文字資料 26](#_Toc206004456)

[13.10 處理 CSV 格式的檔案 26](#_Toc206004457)

[14. 日期、時間與時區值 (Date, Time, and Timezone Values) 27](#_Toc206004458)

[14.1 日期、時間與時區值 (Date, Time, and Timezone Values) 27](#_Toc206004459)

[14.2 計算指定的日期與時間 27](#_Toc206004460)

# 使用者界面技巧

這個章節主要介紹如何透過 dialog 指令、參數處理和格式化輸出，來設計更具互動性、

## 抽取指令列的參數 1

#!/bin/bash

echo "argument of -a is $1"

echo "argument of -b is $2"

# bash test.sh arg1 arg2

# argument of -a is arg1

# argument of -b is arg2

## 抽取指令列的參數 2

當參數數量不固定或包含選項時，我們可以使用 shift 指令來處理。shift 會將所有參數向前移動一位（$2 變成 $1，$3 變成 $2），並移除第一個參數。

#!/bin/bash

while [ "$1" != "" ]; do

    case "$1" in

        -a)

            echo "option -a"

            ;;

        -b)

            echo "option -b"

            ;;

        \*)

            echo "other arg: $1"

            ;;

    esac

    shift

done

# bash test.sh -a -b arg1 -c arg2

## 將檔名帶入程式的判斷中

basename 指令可以將檔案路徑轉換為檔名，這在 Shell Script 中經常被用來根據腳本檔名來執行不同的邏輯。

## 將訊息顯示在畫面上

#!/bin/bash

printf "%5s %-10s\n" "ID" "NAME"

printf "%5d %-10s\n" 1 "Alice"

printf "%5d %-10s\n" 2 "Bob"

## 將訊息顯示在其他的終端機上

# 將訊息 "Hello, world!" 寫到使用者 jack 的終端機

echo "Hello, world!" | write jack

## 將錯誤訊息導向輸出

當指令執行失敗時，會產生錯誤訊息（標準錯誤）。我們可以將其導向 /dev/null，使其不顯示在畫面上。

#!/bin/bash

# 錯誤訊息會顯示在畫面上

ls no\_such\_file

# 將錯誤訊息導向 /dev/null，畫面上不會顯示任何東西

ls no\_such\_file 2>/dev/null

## 輸出訊息時不換行

echo 指令預設會換行。你可以使用 -n 參數來阻止換行

#!/bin/bash

echo -n "hello "

echo "world"

## 不將輸入的文字顯示在螢幕上

stty -echo 指令可以隱藏使用者的輸入，這在要求輸入密碼時非常有用。

#!/bin/bash

echo -n "password: "

stty -echo

read password

stty echo

echo

echo "Welcome to My House"

## 查詢程式正在執行的行數

$LINENO 是一個內建變數，它儲存著腳本目前正在執行的行數

#!/bin/bash

func() {

    echo "This is func(). LINENO: $LINENO"

}

func

echo "main entry. LINENO: $LINENO"

func

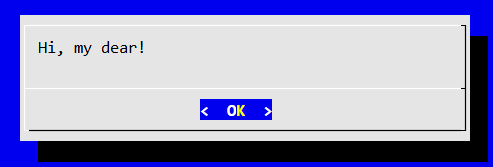
# 使用文字對話框的使用者介面

## 使用文字對話來顯示訊息

dialog --msgbox 用來顯示一條簡單的訊息，需要使用者按下「OK」來關閉。

#!/bin/bash

dialog --msgbox "Hi, my dear!" 6 50



## 從多個選項中選擇

#!/bin/bash

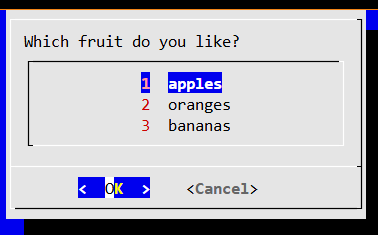
# 15 40 4：對話框的高度、寬度和選項數量

dialog --menu "Which fruit do you like?"  15 40 4 \

    1 "apples" \

    2 "oranges" \

    3 "bananas"



## 使用文字對話來詢問 yes/no

dialog --yesno 用來詢問使用者一個是非題，並根據使用者的選擇返回不同的結束狀態碼。

* 結束狀態碼 0：使用者選擇 Yes。
* 結束狀態碼 1：使用者選擇 No。

#!/bin/bash

dialog --yesno "Are you ready?" 8 70

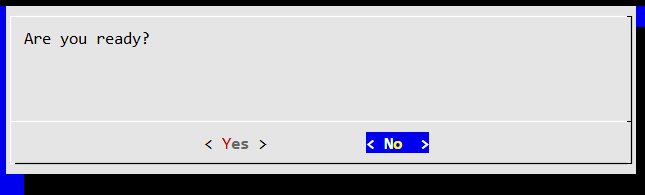
if [ $? -eq 0 ]; then

    echo "You are ready"

else

    echo "You are not ready"

fi



## 在文字對話框取得一行輸入

dialog --inputbox 會顯示一個對話框，讓使用者輸入一行文字，並將輸入的內容存到一個暫存檔中。

#!/bin/bash

tmpfile=$(mktemp)

dialog --inputbox "may I have your name, please?" 8 70 2> "$tmpfile"

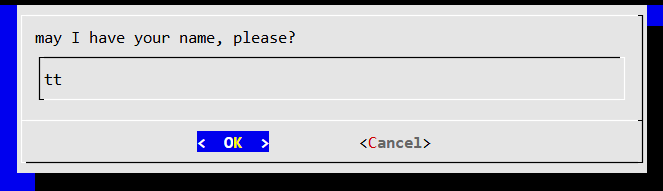
name=$(cat "$tmpfile")

echo "Your name is: $name"

rm "$tmpfile"

# 2> "$tmpfile"：將 dialog 的標準錯誤輸出（也就是使用者輸入的內容）重導向到一個暫存檔。

# cat "$tmpfile"：讀取暫存檔的內容，即使用者的輸入。



## 設計密碼輸入對話框

dialog --insecure 或 dialog --passwordbox 可以用來設計一個密碼輸入對話框，使用者輸入的文字會以 \* 顯示。

#!/bin/bash

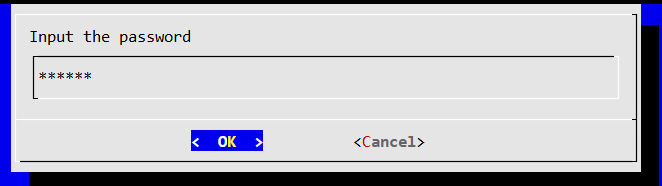
tmpfile=$(mktemp)

dialog --insecure --passwordbox "Input the password" 8 70 2> "$tmpfile"

password=$(cat "$tmpfile")

rm "$tmpfile"

echo "password is: $password"



## 設計進行進度盒

dialog --gauge 會顯示一個進度條，可以用來顯示程式執行的進度。

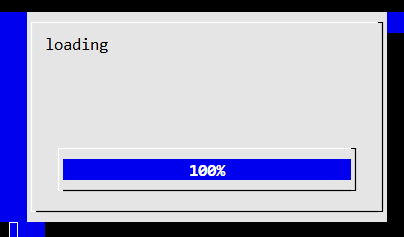
#!/bin/bash

for i in $(seq 10 10 100); do

    sleep 1

    echo "$i"

done | dialog --gauge "loading" 10 40 0



## 設計文字顯示盒

dialog --textbox 會顯示一個對話框，用來顯示一個文字檔案的內容。

#!/bin/bash

dialog --textbox /etc/passwd 20 50

## 設計資訊提示盒

dialog --infobox 類似於 msgbox，但它會在一段時間後自動消失，不需要使用者按 OK。

#!/bin/bash

dialog --infobox "This is a text" 8 50

sleep 3

## 設計時間控制盒

dialog --timebox 會顯示一個對話框，讓使用者輸入時間。

#!/bin/bash

dialog --timebox "Time" 8 30

## 設計倒數計時盒

dialog --pause 會顯示一個對話框，並進行倒數計時，直到計時結束

#!/bin/bash

dialog --pause "ready" 8 30 5

## 設計檔案選擇的對話盒

dialog --fselect 指令會顯示一個檔案選擇器對話框，讓使用者能夠瀏覽檔案系統並選擇一個檔案或資料夾。

#!/bin/bash

# 顯示一個檔案選擇器，預設目錄為 /root/

# 8 30 代表對話框的高度和寬度

dialog --fselect /root/ 8 30

## 透過對話盒來編輯檔案內容

dialog --editbox 指令會顯示一個對話框，讓使用者可以直接在其中編輯一個檔案的內容

#!/bin/bash

# 顯示 /etc/hosts 檔案的內容，並允許使用者編輯

# 10 50 代表對話框的高度和寬度

dialog --editbox /etc/hosts 10 50

# 程式管理的技巧

## 查詢特定指令是否正在執行

當我們需要知道某個程式是否正在系統中運行時，可以使用 ps 指令來查看

#!/bin/bash

# 定義要檢查的程式列表

DAEMONS="cron rpc.idmapd syslogd xinetd upstart"

# 遍歷列表中的每一個程式

for daemon in $DAEMONS; do

    # 使用 ps 指令查找該程式，並使用 grep 進行過濾

    # `ps -ef` 列出所有程序

    # `grep` 過濾出包含程式名稱的行

    # `grep -v "grep"` 排除掉 grep 指令本身

    # 如果找到結果，代表程式正在執行

    out=$(ps -ef | grep "$daemon" | grep -v "grep")

    status="not found"

    if [ ! -z "$out" ]; then

        status="RUNNING"

    fi

    # 輸出結果

    printf "%-16s %s\n" "$daemon" "$status"

done

# cron             RUNNING

# rpc.idmapd       not found

# syslogd          RUNNING

# xinetd           not found

# upstart          not found

## 使用 ps 指令顯示系統程序

ps 指令是查看程序狀態的強大工具

* ps -ax：列出所有程序，包括沒有連結到任何終端機的程序。
* ps -u root：列出 root 使用者執行的所有程序。
* ps -f：顯示完整的格式，包含 PID、PPID（父程序ID）等詳細資訊。

## 使用 kill 指令傳送訊號

kill 指令用於向程序傳送訊號。最常用的訊號是 SIGTERM（終止）和 SIGKILL（強制終止）。

* kill <PID>：預設傳送 SIGTERM 訊號，會讓程序優雅地結束，有機會儲存資料。
* kill -9 <PID> 或 kill -SIGKILL <PID>：傳送 SIGKILL 訊號，會強制結束程序，程序沒有機會做任何清理工作。

範例：

* 啟動一個程式，例如 sleep 1000 &，讓它在背景執行。
* 使用 ps 找到這個 sleep 程序的 PID。
* 執行 kill <PID> 或 kill -9 <PID> 來終止它。

## 讓程序強制結束

killall 指令比 kill 更方便，它不需要 PID，直接使用**程式名稱**就可以終止所有符合的程序。

#!/bin/bash

# 啟動多個 sleep 程序

# & 符號：讓指令在背景執行，這樣你就可以繼續輸入其他指令。

sleep 200 &

sleep 200 &

# 等待一會兒，然後用 ps 檢查

ps -ef | grep sleep

# killall sleep：會終止所有名為 sleep 的程序。

killall sleep

## 在 script 中斷時執行函數

trap 指令可以捕捉到腳本接收到的訊號，並在收到訊號時執行特定的函式或指令。

#!/bin/bash

# 定義一個在腳本結束時執行的函式

clean() {

    echo "執行清理工作..."

    # 這裡可以放清理暫存檔等指令

}

# 捕捉 EXIT 訊號，當腳本結束時會執行 clean 函式

# EXIT 是當腳本正常結束時發出的訊號。當腳本執行完畢時，clean 函式會被自動呼叫。

trap clean EXIT

for i in 1 2 3; do

    echo "Loop $i"

    sleep 1

done

# 程序之間的溝通

## 指定 Script 的回傳碼

在 Shell Script 中，一個指令執行後會有一個回傳碼 (Return Code)，通常也稱為結束狀態碼 (Exit Status)。

* 回傳碼 0 代表指令成功執行。
* 回傳碼 非 0 代表指令失敗，不同的非零值可能代表不同的錯誤原因。

我們可以透過 return 指令來指定函式的回傳碼，或者透過 exit 指令來指定整個腳本的回傳碼。

#!/bin/bash

# 定義一個函式

foo() {

    echo "Input the number:"

    read number

    # 檢查輸入的數字是否為 5

    if [ "$number" -eq 5 ]; then

        return 0  # 成功，回傳碼為 0

    else

        return 1  # 失敗，回傳碼為 1

    fi

}

# 呼叫函式

foo

# 使用 $? 來取得上一個指令（foo 函式）的回傳碼

echo "The return code is $?"

## 使用訊號與其他程序溝通

├── a.sh：負責監控與回應訊號

└── b.sh：負責發送訊號

# a.sh

#!/bin/bash

# 定義一個函式，用於處理 USR1 訊號

do\_log\_usr1() {

    echo "$(date): user1 signal received" >> /tmp/pa.log

}

# 定義一個函式，用於處理 USR2 訊號

do\_log\_usr2() {

    echo "$(date): user2 signal received" >> /tmp/pa.log

}

# trap 指令：當收到 USR1 訊號時，執行 do\_log\_usr1 函式

trap do\_log\_usr1 USR1

# trap 指令：當收到 USR2 訊號時，執行 do\_log\_usr2 函式

trap do\_log\_usr2 USR2

# 無限迴圈，讓腳本持續運行

while true; do

    sleep 1

done

# b.sh

#!/bin/bash

# 找到 161\_a.sh 的 PID

pid=$(ps -ef | grep 161\_a.sh | grep -v "grep" | awk '{print $2}')

# 向 161\_a.sh 發送 USR1 訊號

kill -USR1 $pid

# 向 161\_a.sh 發送 USR2 訊號

kill -USR2 $pid

# 等待一會兒

sleep 1

# 檢查 /tmp/pa.log 的內容

cat /tmp/pa.log

# 結束 161\_a.sh

kill -9 $pid

## 查詢與設定網路資訊

ifconfig 指令是一個用來查詢和設定 IP 位址與子網路遮罩的指令 。它能顯示目前網路卡的所有資訊，包括 IP 位址、廣播位址等 。

## 查詢自己的 IP 位址