Power Shell：指令操作

**目錄**

[Power Shell：指令操作 1](#_Toc205827639)

[1. 前景執行與背景執行 1](#_Toc205827640)

[2. 一行同時執行多個指令 1](#_Toc205827641)

[3. 指令的路徑問題 2](#_Toc205827642)

[4. 檔案的權限設定 2](#_Toc205827643)

[4.1 看檔案權限 2](#_Toc205827644)

[4.2 修改檔案權限 2](#_Toc205827645)

[4.3 指令的安裝 2](#_Toc205827646)

# 認識 Shell 與Shell Script

在 Linux 系統中可以分成三層：

* Kernel（核心）：直接與硬體溝通，負責系統資源管理。
* Shell（殼層）：接收使用者輸入的指令，把它轉交給 Kernel 執行，再把結果顯示給使用者。
* File System Structure（檔案系統結構）：管理檔案和資料夾。

簡單說：Shell 是使用者與核心之間的「翻譯官」。

## Shell Script

Shell 也可以寫成腳本檔案（.sh），一次執行多行指令。

用途：自動化工作、系統管理、定時排程

Shell Script 用 Shell 語法編寫，執行時由 Shell 解譯。

## Linux 中常見的 Shell：

* Bourne Shell (sh)：最早的 Unix Shell
* C Shell (csh)：語法類似 C 語言
* Korn Shell (ksh)：結合 Bourne 與 C Shell 優點
* Bourne Again Shell (bash)：Linux 預設 Shell，功能最完整
* T shell (tcsh)：加強版 C Shell

# 認識 Bash

Bash 全名 Bourne-Again Shell，是 GNU 專案開發的 Unix Shell。

Bash 是最常見的 shell，也是許多 Linux 系統的預設 shell。

## Bash 的環境變數

Bash 啟動時會載入一組環境變數，影響系統運作與使用者操作。

|  |  |
| --- | --- |
| # 查看變數  echo $變數名 | # 設定變數（只對當前 session 有效）  # 如果要永久生效：修改 ~/.bashrc 或 /etc/profile  變數名=值  export 變數名 |

## 常見環境變數

|  |  |
| --- | --- |
| **變數名** | **功能** |
| PATH | 命令搜尋路徑（用 : 分隔） |
| HOME | 使用者家目錄 |
| USER | 使用者名稱 |
| SHELL | 當前使用的 Shell 路徑 |
| PS1 | 主提示符號格式 |
| HISTSIZE | 歷史指令記錄數量 |
| LANG | 語系設定 |
| PWD | 當前工作目錄 |
| RANDOM | 產生隨機數 |
| UID | 使用者 ID |

範例：查看查看 Bash 版本

echo $BASH\_VERSION

# 變數

要加上「$」符號，等號左右不可以有空格。

## 範例：初始化變數hello

# !/bin/bash

hello="Hello World"

echo $hello

## 範例：變數沒有一定資料類型

# !/bin/bash

t=100              # t 是整數

let "t+=23"        # 整數運算，t = t + 23

echo "t = $t "     # 印出 t=123

s=AB3

echo "s = $s" # s = AB3

## 範例：空變數自動轉成整數

z=""             # 空字串

echo "z = $z"    # 會輸出 z =

let "z += 1"     # 嘗試用整數運算，空字串會當 0

echo "z = $z"    # 變成 z=1

## 區域變數

#!/bin/bash

# 區域變數

hello="hello world"

# 使用函數

function hello01(){

    local hello="這是hello01"

    echo $hello

}

function hello02(){

    local hello="這是hello01"

    echo $hello

}

echo $hello

hello01

hello02

## 參數變數

在Shell Script中的參數,會自動變成程式內的變數。

|  |  |
| --- | --- |
| **變數** | **意義** |
| $0 | 腳本名稱 |
| $1 | 第一個參數 |
| $2 | 第二個參數 |
| $3 | 第三個參數 |
| $@ | 可以在迴圈中安全地取得每一個參數（遇到空白不會被切掉）。 |
| $\* | 跟 $@ 很像，但會把全部參數當成一個字串。 |

#!/bin/bash

echo "\$0 = $0"

echo "\$1 = $1"

echo "\$2 = $2"

echo "\$3 = $3"

echo "\$@ = $@"

# 執行方式 bash test.sh arg1 arg2 arg3

# $0 = test.sh

# $1 = arg1

# $2 = arg2

# $3 = arg3

# $@ = arg1 arg2 arg

## 變數?

$? 會儲存上一個指令的退出碼（Exit Status）。

0 代表指令執行成功，非 0 代表失敗或條件不成立。

|  |  |
| --- | --- |
| 範例：條件成立 | 範例：條件不成立 |
| #!/bin/bash  [ 3 -lt 2 ]   # 判斷 3 是否小於 2  echo $?       # 輸出 1（不成立） | #!/bin/bash  [ 3 -lt 2 ]   # 判斷 3 是否小於 2  echo $?       # 輸出 1（不成立） |

Bash 的數字比較運算子（在 [ ] 或 test 裡用）

|  |  |
| --- | --- |
| **比較符號** | **意義** |
| -eq | 等於 |
| -ne | 不等於 |
| -lt | 小於 |
| -le | 小於等於 |
| -gt | 大於 |
| -ge | 大於等於 |

Bash 的字串比較運算子

|  |  |
| --- | --- |
| **比較符號** | **意義** |
| = | 相等 |
| != | 不相等 |
| -z s | 長度為 0（空字串） |
| -n s | 長度非 0 |

#!/bin/bash

x=10

y=20

if [ $x -lt $y ]; then

    echo "$x 小於 $y" # 10 小於 20

fi

#!/bin/bash

name="Amy"

if [ "$name" = "Amy" ]; then

    echo "名字一樣"

fi

檔案測試運算子

|  |  |
| --- | --- |
| **運算子** | **意義** |
| -e file | 存在 |
| -f file | 是一般檔 |
| -d file | 是目錄 |
| -s file | 檔案大小 > 0 |
| -r file | 可讀 |
| -w file | 可寫 |
| -x file | 可執行 |

檔案與目錄的比較

|  |  |
| --- | --- |
| **運算子** | **意義** |
| file1 -nt file2 | file1 比 file2 新 |
| file1 -ot file2 | file1 比 file2 舊 |
| file1 -ef file2 | 兩者指向同一檔案（硬連結） |

# 判斷式

## if 判斷語法

**if** **[** 條件1 **];** **then**

動作1

**elif** **[** 條件2 **];** **then**

動作2

**elif** **[** 條件3 **];** **then**

動作3

**else**

動作4

**fi**

## 範例：密碼比對

#!/bin/bash

I\_PASSWORD="abc123"  # 預設密碼

echo "Please enter the password:"

read PASSWORD         # 讀取使用者輸入

if [ "$PASSWORD" == "$I\_PASSWORD" ]; then

    echo "Welcome login!"

else

    echo "ACCESS DENIED!"

fi

## 範例：成績判斷 (if-elif-else)

#!/bin/bash

echo "Please enter your score (0-100):"

read grade

if [ "$grade" -lt 0 ] || [ "$grade" -gt 100 ]; then

    echo "Sorry, you are out of range."

elif [ "$grade" -lt 60 ]; then

    echo "Not pass."

elif [ "$grade" -lt 80 ]; then

    echo "Good!"

elif [ "$grade" -le 90 ]; then

    echo "Great!"

else

    echo "Excellent!"

fi

## case 判斷語法

**case** $變數 **in**

模式1**)** 動作1 **;;**

模式2**)** 動作2 **;;**

模式3**)** 動作3 **;;**

**\*)** 預設動作 **;;**

**esac**

## 範例：輸入 1~5 對應文字

#!/bin/bash

echo "Enter a number between 1 to 5:"

read NUM

case $NUM in

    1) echo "one" ;;

    2) echo "two" ;;

    3) echo "three" ;;

    4) echo "four" ;;

    5) echo "five" ;;

    \*) echo "INVALID NUMBER!" ;;

esac

# 迴圈

## for 迴圈

**for** 變數 **in** 值1 值2 值3 **...**

**do**

指令

**done**

for name in Alice Bob Carol

do

    echo "Hello, $name"

done

## while 迴圈

**while** **[** 條件判斷式 **]**

**do**

指令

**done**

count=1

while [ $count -le 5 ]

do

    echo "count = $count"

    let count++

done

# 函數

## 定義函式 (Defining a Function)

函式名稱**()** **{**

# 要執行的指令...

**echo** "You are a boy"

**}**

## 範例：使用函數

#!/bin/bash

myfunction() {

    echo "you are a pretty girl"

}

myfunction # you are a pretty girl

## 取消定義函式 (Unsetting a Function)

#!/bin/bash

# 定義函式

funtion() {

    echo "you are a boy"

}

funtion # 第一次呼叫，正常執行

unset -f funtion # 取消函式定義

# 第二次呼叫，會發生錯誤

funtion # test.sh: line 15: funtion: command not found

## 在呼叫函式時傳遞參數 (Passing Arguments to a Function)

#!/bin/bash

echo() {

    echo "總共有 $# 個參數"

    echo "第一個是: $1"

    echo "第二個是: $2"

    echo "第三個是: $3"

}

# 呼叫函式並傳遞三個參數

echo YA NO YO

## 在函式內部取得結果 (Getting a Return Value from a Function)

#!/bin/bash

# 這個函式用來判斷今天是不是星期一

ismonday() {

    # `date +%A` 會輸出今天的星期幾 (例如: Monday)

    if [ "$(date +%A)" == "Monday" ]; then

        return 0  # 如果是星期一，回傳 0 (成功)

    else

        return 1  # 如果不是，回傳 1 (失敗)

    fi

}

# 呼叫 ismonday 函式

ismonday

# `$?` 變數會儲存上一個指令 (也就是 ismonday) 的回傳值

if [ $? -eq 0 ]; then

    echo "今天是星期一，工作順利"

else

    echo "今天不是星期一"

fi

## 遞迴：呼叫函式本身 (Recursive Function)

#!/bin/bash

# 計算階乘的函式

CallFunction() {

    if [ $1 -eq 1 ]; then

        Value=1

    else

        # 呼叫自己，但參數減 1

        CallFunction $(($1 - 1))

        Value=$(($1 \* $Value))

    fi

}

# 計算 5!

CallFunction 5

echo $Value

## 呼叫其他檔案中定義的函式 (Sourcing Functions from Other Files)

你可以將常用的函式整理在一個獨立的檔案中，然後在需要使用的腳本裡，透過 source 指令或 . 指令將它載入進來。這樣可以提高程式碼的重複使用性。

├── a.sh

└── test.sh(主檔案)

# a.sh

#!/bin/bash

a123() {

    echo "pig"

}

# test.sh

#!/bin/bash

. ./a.sh # 要空格歐

a123

echo "dog"

# .使用變數

## 接收傳遞 script 或函式的參數 (Receiving Arguments)

執行腳本時，可以在檔名後面加上參數，腳本內部可以用特殊變數來接收。

* $#: 代表傳入參數的總個數。
* $1, $2, ...: 依序代表第一個、第二個...參數。
* shift: 這個指令可以讓參數列表往前移。例如執行 shift 後，原本的 $2 會變成 $1，原本的 $3 會變成 $2，以此類推。

#!/bin/bash

echo "Number of arguments: $#"

# 當還有參數存在時，就繼續迴圈

while [ "$#" -gt "0" ]; do

    echo "第一個參數是: $1"

    shift  # 將參數往前移

done

> bash test.sh a b c d

# Number of arguments: 4

# 第一個參數是: a

# 第一個參數是: b

# 第一個參數是: c

# 第一個參數是: d

## 取消定義變數 (Unsetting Variables)

當你不再需要一個變數時，可以使用 unset 指令將它從記憶體中完全移除。這比將它設為空字串 (var="") 更徹底，因為 unset 後，變數本身就不存在了。

#!/bin/bash

my\_secret="TOP\_SECRET\_INFO"

echo "執行 unset 之前: $my\_secret"

unset my\_secret # 取消定義該變數

echo "執行 unset 之後: $my\_secret" # 再次嘗試存取，會得到空值

if [ -z "$my\_secret" ]; then

  echo "變數 'my\_secret' 已經不存在了。"

fi

## 查詢變數是否被定義 (Checking if a Variable is Set)

使用 -n 和 -z 運算子，通常與if 條件式搭配使用

**if** **[** **-n** "$username" **];** # 檢查變數的字串長度是否 "不為零"

**if** **[** **-z** "$user\_email" **];** # 檢查變數的字串長度是否 "為零"

#!/bin/bash

# 情況 1: username 有被設定值

username="gordon"

if [ -n "$username" ]; then

  echo "變數 'username' 已定義，值為: $username"

fi

# 情況 2: user\_email 未被定義

if [ -z "$user\_email" ]; then

  echo "變數 'user\_email' 是空的或未定義。"

fi

# 情況 3: user\_id 被定義為空字串

user\_id=""

if [ -z "$user\_id" ]; then

  echo "變數 'user\_id' 是空的。"

fi

## 在變數未被定義時，使用預設值或顯示錯誤

Shell提供「參數擴展」(Parameter Expansion)，可以讓你用非常簡潔的語法處理變數未定義的情況。

${VARIABLE:-default\_value}

#!/bin/bash

# 1. color 未定義，使用預設值 'blue'

echo "你最喜歡的顏色是 ${color:-blue}。"

echo "檢查變數本身: color 的值是 '$color'" # 變數 color 依然是空的

echo "---"

# 2. color 有被定義，使用它自己的值

color="red"

echo "你最喜歡的顏色是 ${color:-blue}。"

## 將變數的值設為唯讀 (Read-only Variables)

一旦變數被設為唯讀 (readonly)，它的值就不能再被修改或 unset。

#!/bin/bash

readonly myname="jack"

myname="sue" # 任何嘗試修改唯讀變數的行為都會導致錯誤

## 將檔案的內容寫入變數 (Assigning File Content to a Variable)

# 要先建立一個檔案text.txt，裡面隨便寫些東西

#!/bin/bash

animal=`cat text.txt`

echo $animal

# 使用字串 (Using Strings)

## 寫一個橫跨多行的字串 (Creating a Multi-line String)

有時候你需要處理的字串會跨越多行。常見的方法是使用 cat 搭配 << (Here Document)。

cat <<'結束標記' 的語法可以讓你輸入多行文字，直到你再次輸入 結束標記 為止。所有在此之間的內容，都會被 cat 指令輸出，然後可以透過指令替換 (`...` 或 $(...)) 將這些多行內容存入一個變數中。

#!/bin/bash

# 使用 Here Document 將多行文字存入變數 str

# 'EOF' 是一個自訂的結束標記，可以是任何你喜歡的詞

# 結束標記前後都不能有任何空格

str=$(cat <<'EOF'

a/b/c/d/apple/e/tomato/g

a/brother/sister/g

apple

EOF

)

echo "$str" # 印出這個多行變數的內容

# test.sh: line 15: warning: here-document at line 11 delimited by end-of-file (wanted `EOF')

# a/b/c/d/apple/e/tomato/g

# a/brother/sister/g

# apple

## 跳脫字元使用方法 (Using Escape Characters)

跳脫字元 (Escape Character)，通常是反斜線 \，它能讓緊跟在它後面的那個特殊字元失去其特殊意義，變回一個普通的字元。echo 指令搭配 -E (預設) 或 -e 選項時，對跳脫字元的處理會不同：

* echo (或 echo -E): 不會解釋跳脫序列，\ 就只是一個 \。
* echo -e: 會解釋跳脫序列，例如 \n 會變成換行，\t 會變成 Tab。

#!/bin/bash

# 使用 -e 來讓 \t 被解釋為 Tab 鍵

echo -e "Hello\tWorld" # Hello   World

# 不使用 -e，\t 就只是普通字元

echo "Hello\tWorld" # Hello\tWorld

## 字串切片

取出字串中的第 M 個字元到第 N 個字元的內容 (Extracting a Substring)

${變數:起始位置:長度}

#!/bin/bash

word=applebanana

# 從第 2 個位置 (第三個字元 'p') 開始，取 5 個字元

getout=${word:2:5}

echo $getout # pleba