一、 Script 基本

```
#!/bin/bash                    # Shebang：指定用 bash 執行
# 檔案以「一行一行」自上而下執行
```

執行方式:

```
bash /path/to/script.sh        # 方式 1：用解譯器執行
./script.sh                    # 方式 2：直接執行（需先 chmod +x script.sh）
source script.sh               # 方式 3：在「目前 shell」執行，同義：. script.sh
# 差異：./ 會在「子 shell」跑；source 會改變當前 shell 的變數/環境。
```

二、 環境與變數

```
env / printenv                 # 列出所有環境變數
echo $HOME $PWD $SHELL $USER   # 常用系統變數
echo $PATH $HOSTNAME $LANG     # 其他常見
```

自訂變數:

```
name="Alice"          #= 兩側不能有空白；建議字串加引號
echo "$name"
export name           # 變成「環境變數」(可傳給子行程)
readonly ver=1.0      # 只讀
# 取消變數：unset name
```

參數與特殊變數:

```
$0   腳本名           $#   參數個數
$1 $2 …               # 各參數
$@                    # 以「多個獨立參數」展開（建議搭配雙引號使用："$@"）
```

```
$*                    # 合成一個字串（由 IFS 連接），較少用
$?                    # 前一個指令的回傳碼（0=成功）
$$                    # 目前 shell 的 PID
$!                    # 最近一次背景行程 PID
```

三、 數學運算

```
expr 1 + 2                  # 3（運算子兩邊要有空白）
echo $[1+2]                 # 3（舊式）
echo $(( 1 + 2 ))           # 3（建議）
let x=1 y=2 sum=x+y         # 也可用 let
```

四、 數學運算

1) 整數比較:

-eq 等於 -ne 不等於

-lt 小於 -le 小於等於

-gt 大於 -ge 大於等於

範例：

```
a=15
[ "$a" -lt 20 ] && echo "$a < 20"
[[ $a -gt 10 && $a -lt 35 ]] && echo "10 < a < 35"
# 在 [ ] 中，-a 等價 &&；-o 等價 ||（不如直接用 [[ ... && ... ]] 清楚）
```

2) 字串比較:

```
[ "$s" = "hi" ]          # 相等
[ "$s" != "hi" ]         # 不等
[ -n "$s" ]              # 非空字串
```

[-z "$s"]　　# 空字串

舊技巧：避免空值出錯會寫成 ["${s}x" = "atguigux"]；用 [[ ]] 通常不必這樣

3) 檔案/目錄測試:

-e 存在　　-f 一般檔

-d 目錄　　-L 符號連結

-r 可讀　　-w 可寫

-x 可執行　　-s 檔案大小 > 0

五、 if／邏輯運算

```
if [[ $a -gt 18 && $a -lt 35 ]]; then
   echo "18~35"
elif [[ $a -ge 35 ]]; then
   echo ">= 35"
else
   echo "<= 18"
fi
```

- && 與 || 為 shell 的邏輯運算，**優先推薦**（比 -a/-o 清楚）。

六、 實用片段

```
if [[ "$1" == "atguigu" ]]; then
   echo "welcome, atguigu"
fi
```

一行式:

```
a=15; [ "$a" -lt 20 ] && echo "$a < 20"
```

遍歷所有參數:

```
for arg in "$@"; do
  echo "$arg"
done
```

case 分支:

```
case "$變數" in
  "值 1")
    # 動作 1
    ;;
  "值 2"|"值 2 別名")
    # 動作 2
    ;;
  *)
    # 其他情況（預設）
    ;;
Esac
```

- 每個分支以 ;; 結束；*) 為「否則」。

for 迴圈（C 風格）:

```
for (( i=1; i<=N; i++ )); do
  echo "$i"
done
```

- 三段：初始值；條件；遞增。可搭配 ((…)) 做整數運算。

for … in（走訪清單/陣列）：

```
for os in linux windows macos; do
    echo "$os"
done
arr=(a b c)
for x in "${arr[@]}"; do
    echo "$x"
done
```

while 迴圈（條件為真就執行）：

```
a=1; sum=0
while [ "$a" -le "$1" ]; do
    sum=$(( sum + a ))      # 或 let sum+=a
    a=$(( a + 1 ))          # 或 let a++
done
echo "$sum"
```

提醒：[…] 內 **兩邊要有空白**；建議用整數運算 ((…)) 或 let。

使用者輸入（read）：

```
read -t 10 -p "請輸入您的芳名: " name
echo "welcome, $name"
```

- -t 秒數：逾時；-p：提示字串；輸入存到變數 name。

basename（取檔名）：

basename <path> [suffix]

- 回傳路徑最後一段（檔名）。
- 若給了 suffix，會把結尾的該字串去掉。

basename /home/user/banzhang.txt　　# => banzhang.txt

basename /home/user/banzhang.txt .txt　　# => banzhang

小補充：**搭配 dirname 取路徑**

dirname /home/user/banzhang.txt　　# => /home/user

- 變數含空白時務必加引號：basename "$path"。

Bash 函式（function）：

兩種宣告方式（作用相同）：

funname() { commands; }　　# 常見

function funname { commands; }　　# 另一種

使用與回傳值：

```
say_hi() {
  local who="$1"                  # 建議用 local，避免污染外部環境
  echo "hi, $who"                 # 回傳字串：用 echo + 指令替換
  return 0                        # 回傳狀態碼(0~255)，不寫則為最後一行指令的狀態碼
}

msg="$(say_hi Alice)"             # 以字串接收
echo "$msg"
say_hi Bob >/dev/null; echo $? # 以$? 取狀態碼
```

- 先宣告再呼叫（腳本是自上而下解譯）。

grep 正規表示式速查:

錨點與萬用符:

^ 行首　　$ 行尾

. 任意單一字元

* 前一個字元重複 0 次以上

例：

grep '^a' /etc/passwd　　# 以 a 開頭

grep 't$' /etc/passwd　　# 以 t 結尾

grep 'r..t' /etc/passwd　　#r 任兩字元 t

grep 'ro*t' /etc/passwd　　#r 後接 0+ 個 o 再接 t（rt/rot/root/rooot…）

grep -E '.*' file　　#.*：任意長度任意字元（-E 用擴充語法）

字元類別（中括號 []）:

[68]　　匹配 6 或 8

[0-9]　　一個數字

[0-9]*　　任意長度的數字字串

[a-z]　　一個小寫英文字母

[a-z]*　　任意長度的小寫字母字串

[a-c e-f] a~c 或 e~f 範圍內任一字元（空白可省略）

[^0-9]　　非數字（^ 在 [] 內代表「否定」）

◆ 進階數量詞（需要 grep -E 或使用 grep 搭配跳脫大括號）

{m}　　重複 m 次

{m,}　　m 次以上

{m,n}　　m~n 次

例：grep -E '^[0-9]{3}-[0-9]{2}$' file

建議：字串樣式加引號；需要更強語法用 grep -E（等同 egrep）。

cut：按欄/字元切割輸出

cut [選項] filename

常用選項:

- -d '符號'：指定分隔符（預設是 tab）
- -f 列號：取第幾欄；可用 1,3,5 或 2-（第 2 欄到最後）
- -c 位數：按「字元位置」切割；如 -c 1-5,10
- -b 位元組：按位元組切割（多位元組字元要小心）
- -s：沒有分隔符的行不輸出（否則原樣輸出）

取 /etc/passwd 的使用者與 shell（冒號分隔）

cut -d: -f1,7 /etc/passwd

取 CSV 的第 2 欄之後

cut -d, -f2- file.csv

小提醒（限制）

- 分隔符只能是**單一字元**；多個空白不穩定。
  多空白建議先「壓縮空白」或改用 awk：

  # 壓縮成單一空白再 cut

  command | tr -s ' ' | cut -d' ' -f10

  # 或直接用 awk（更穩定）

  command | awk '{print $10}'

awk：欄位導向的文字分析器

awk [選項] 'pattern { action } pattern2 { action2 } ...' file

預設分隔符為「空白（連續空白視同一個）」

欄位與變數

- $0 整行、$1..$NF 第 1…最後欄
- NR 目前行號、FNR 目前檔案內的行號、NF 本行欄位數
- FS 輸入分隔符、OFS 輸出分隔符（可在 BEGIN 設定）
- -F '符號' 指定輸入分隔符；-v 變數=值 傳入外部變數

流程區塊

- BEGIN{...}：讀檔前執行（常用來設 FS/OFS、印表頭）
- 主體 {...}：逐行匹配 pattern 後執行
- END{...}：全部處理完後執行（常用來印總計）

常用動作:

```
print $1, $7                # 以 OFS 連接輸出
printf "%-10s %s\n", $1, $7
```

匹配條件（擇一）:

```
/regex/                     # 正規表示式
$3 > 100 && $2 == "ok"   # 條件運算
```

常用內建函式

- length(), substr(s,i,n), tolower()/toupper()
- split(s, arr, FS), gsub(r, repl, s)（全域取代）

例子:

1) 只列出 /etc/passwd 的帳號與 shell；逗號分隔、加表頭、最後再加一行

```
awk -F: 'BEGIN{print "user,shell"} {print $1","$7} END{print
"dahaige,/bin/zuishuai"}' /etc/passwd
```

2) 以空白為分隔，印第 10 欄（比 cut 對多空白更穩）

ifconfig | awk '/netmask/ {print $10}'

3) 以逗號分隔的 CSV，跳過表頭，統計第 3 欄總和

awk -F, 'NR>1 {sum+=$3} END{print sum}' data.csv

4) 傳入外部變數並用它做篩選

awk -F: -v u="$USER" '$1==u {print $0}' /etc/passwd

輸出分隔符小技巧:

```
# 讓 print 以逗號當輸出分隔

awk -F: 'BEGIN{OFS=","} {print $1,$7}' /etc/passwd
```

參考來源:

https://www.bilibili.com/video/BV1WY4y1H7d3?spm_id_from=333.788.videopod.episodes&vd_source=a6cbb8d6eb12bab9b5314690e3b03bd2