

【Treeman】Linux

- [shell](#)
 - [【jq】 linux json 查詢工具\(查詢篇\)](#)
 - [【jq】 linux json 查詢工具\(修改篇\)](#)
 - [【jq】 相關連結](#)
 - [【Shell】 【tip】 使用暫存表，更新crontab](#)
 - [【Shell】 【tip】 使用 echo 輸出排程時間到log](#)
 - [【Shell】 linux 下source、sh、bash、./執行指令碼的區別](#)
 - [【Shell】 date 做 timestamp 格式轉換計算](#)
 - [【Shell】 awk 取出特定欄位](#)
 - [【Shell】 取得pubic ip](#)
 - [【shell】 內建變數](#)
 - [【Shell】 動態呼叫awk自訂函數](#)
 - [【Shell】 【Tool】 測試機器防火牆是否開通](#)
 - [【Shell】 【Tool】 批次更新密碼](#)
 - [【Shell】 【Tip】 列出環境變數網址解析dns](#)
- [【茶包射手】](#)
 - [version `GLIBCXX_3.4.20' not found 解決方法](#)
 - [【cron】 沒有依排程執行](#)
- [【Top】 效能監控工具](#)
- [【資安】 強制登出使用者](#)
- [【圖形介面】 Linux安裝圖形介面](#)
- [【圖形介面】 tiger vnc](#)
- [【Linux】 帳號管理](#)
- [【Linux】 記憶體相關](#)
- [【crontab】 排程相關](#)
- [【vim】 快速鍵](#)
- [【Linux】 時區設定](#)
- [【Linux】 載入順序](#)
- [【更新】 Centos 7已於2024年6月30日停止維護，更換yum源解決yum404問題](#)
- [【Linux】 DNS 相關](#)
- [【Linux】 Dns 動態解析](#)
- [【Linux】 【權限管理】 【acl】 設定/var/www/html for tomcat 可讀寫](#)
- [【Linux】 SSH 跳板機設定](#)
- [【Tool】 tmux](#)

shell

【jq】linux json 查詢工具(查詢篇)

- data

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ],
  "gender": {
    "type": "male"
  }
}
```

- 取得資料(age)

```
[root@jylee-stres-stest250 tmp]# cat data.json | jq .age
```

```
25
```

- 取得資料(city) (address > city)

```
[root@jylee-stres-stest250 tmp]# cat data.json | jq .address.city
"New York"
```

- 輸出不帶雙引號 **-r** :

```
[root@jylee-stres-stest250 tmp]# cat data.json | jq .firstName
"John"
```

```
[root@jylee-stres-stest250 tmp]# cat data.json | jq -r .firstName
John
```

- 取得資料(city) (address > city)

```
[root@jylee-stres-stest250 tmp]# cat data.json | jq .address.city
"New York"
```

- 取得電話

取得電話陣列

```
[root@jylee-stres-stest250 tmp]# cat data.json | jq .phoneNumber
[
  {
    "type": "home",
    "number": "212 555-1234"
  },
  {
    "type": "fax",
```

```
    "number": "646 555-4567"
  }
]
```

#取得第一組電話

```
[root@jylee-stres-stest250 tmp]# cat data.json | jq .phoneNumber[0]
{
  "type": "home",
  "number": "212 555-1234"
}
```

#取得第一組電話(type + number)

```
[root@jylee-stres-stest250 tmp]#cat data.json | jq -r '.phoneNumber[0] | .type+" "+.number'
home,212 555-1234
```

#取得所有type

```
[root@jylee-stres-stest250 tmp]# cat data.json | jq '.phoneNumber[] | .type'
"home"
"fax"
```

- 使用map,join重組資料

```
[root@jylee-stres-stest250 tmp]# cat data.json | jq '.phoneNumber | map(.type)'
[
  "home",
  "fax"
]
[root@jylee-stres-stest250 tmp]# cat data.json | jq '.phoneNumber | map(.type) | join(",")'
"home,fax"
```

- 使用select 過濾資料

```
[root@jylee-stres-stest250 tmp]# cat sample.json | jq -r '.phoneNumber[] | select(.type == "home") '
{
  "type": "home",
  "number": "212 555-1234"
}

[root@jylee-stres-stest250 tmp]# cat sample.json | jq -r '.phoneNumber[] | select(.type == "home") | .number'
212 555-1234
```

- 依key取出值

回傳物件

```
[root@jylee-stres-stest250 tmp]# echo '{"a":1, "ip":["10.10.10.1","10.10.10.2"], "c":3, "d":4}' | jq '{ip, d}'
```

```
{
  "ip": [
    "10.10.10.1",
    "10.10.10.2"
  ],
  "d": 4
}
```

回傳傳列

```
[root@jylee-stres-stest250 tmp]# echo '{"a":1, "ip":["10.10.10.1","10.10.10.2"], "c":3, "d":4}' | jq ' [{ip, d}]'
```

```
[
  {
    "ip": [
      "10.10.10.1",
      "10.10.10.2"
    ],
    "d": 4
  }
]
```

jq_array_to_shell

data

```
{
  "values": [
    {
      "email": "user1@domain.com",
      "id": "USER1_ID"
    }, {
      "email": "user2@domain.com",
      "id": "USER2_ID"
    }
  ]
}
```

ss.sh

```
while IFS=' ' read -r email id; do
  echo "$email"
  echo "$id"
done <<EOF
$(jq -r '.values[] | "\(.email),\(.id)"' data2.json)
EOF
```

```
[root@jylee-stres-stest250 tmp]# sh ss.sh
user1@domain.com
USER1_ID
user2@domain.com
USER2_ID
```

<https://stedolan.github.io/jq/manual/#Invokingjq>

【jq】linux json 查詢工具(修改篇)

- 修改前節點資料

```
[root@jylee-stres-stest250 tmp]# echo '{"a":{"b":0, "c":1}}' | jq .
{
  "a": {
    "b": 0,
    "c": 1
  }
}
```

- 新增 / 修改 : setpath([路徑]; 值)

```
[root@jylee-stres-stest250 tmp]# echo '{"a":{"b":0, "c":1}}' | jq 'setpath(["a","b"];1)'
#修改 a . b ; 0 => 1
{
  "a": {
    "b": 1,
    "c": 1
  }
}

#修改 多筆值用 | 分開
#修改 a . b ; 0 => 1,
# 修改 a . c ; 1=> array( 1,2)
[root@jylee-stres-stest250 tmp]# echo '{"a":{"b":0, "c":1}}' | jq 'setpath(["a","b"];1) | setpath(["a","c"];[1,2])'
{
  "a": {
    "b": 1,
    "c": [
      1,
      2
    ]
  }
}
```

- 刪除 delpaths ([[刪除節點1] , [刪除節點2] ...])

```
[root@jylee-stres-stest250 tmp]# echo '{"a":{"b":0, "c":1},"d": 111}' | jq .
{
  "a": {
    "b": 0,
    "c": 1
  },
  "d": 111
}

# 刪除d節點
[root@jylee-stres-stest250 tmp]# echo '{"a":{"b":0, "c":1},"d": 111}' | jq 'delpaths([["d"]])'
{
  "a": {
    "b": 0,
    "c": 1
  }
}

# 刪除d節點, a.b 節點
[root@jylee-stres-stest250 tmp]# echo '{"a":{"b":0, "c":1},"d": 111}' | jq 'delpaths([["d"],["a","b"]])'
{
  "a": {
    "c": 1
  }
}
```

shell

【jq】相關連結

- [jq play](#)
- [jq 常用操作 - mozillazg's Blog](#)

【Shell】 【tip】 使用暫存表，更新 crontab

```
#!/bin/bash
# $$ => pid , 使暫存檔名不重複
tmp_file=/tmp/cron$$

# 利用trap 刪除暫存檔
#This installs a trap handler for exit (0) and signals 1 (SIGHUP), 3 (SIGQUIT) and 15 (SIGTERM)
# that removes a file presumably storing a process ID.
#See man 3 signal for details about signals.
trap "rm -rf $tmp_file" 0 1 3 15

cat > $tmp_file << EOF
*/10 * * * * echo "Hello World"
EOF

cat $tmp_file
cat $tmp_file | crontab
```


shell

【Shell】 【tip】 使用 echo 輸出排程時間到log

```
# 使用 echo 輸出排程時間到log
# 測試輸出到 /tmp/crontab.log
0 8 * * * echo "$(date '+\%Y-\%m-\%d \%H:\%M:\%S') - $(curl -s 'http://localhost:8080/api/schedule/sendSMS/sendDailySms'
--form "date=$(date -d 'yesterday' +\%Y-\%m-\%d)" --form 'onlyCheckData=true')"
>> /tmp/crontab_$(date '+\%Y-\%m').log
```

【Shell】linux 下source、sh、bash、./執行指令碼的區別

linux 下source、sh、bash、./執行指令碼的區別 - IT閱讀 (itread01.com)

1、source命令用法：

source FileName

作用:在當前bash環境下讀取並執行FileName中的命令。該filename檔案可以無"執行許可權"

注：該命令通常用命令"."來替代。

如：source .bash_profile

。 .bash_profile兩者等效。

source(或點)命令通常用於重新執行剛修改的初始化文件。

source命令(從 C Shell 而來)是bash shell的內建命令。

點命令，就是個點符號，(從Bourne Shell而來)。

source的程式主體是bash，指令碼中的\$0變數的值是bash，而且由於作用於當前bash環境，指令碼中set的變數將直接起效

2、sh, bash的命令用法：

sh/bash FileName

作用:開啟一個子shell來讀取並執行FileName中命令。該filename檔案可以無"執行許可權"

注：執行一個shell指令碼時會啟動另一個命令直譯器。

每個shell指令碼有效地執行在父shell(parent shell)的一個子程序裡。

這個父shell是指一個控制終端或在一個xterm視窗中給你命令指示符的程序。

shell指令碼也可以啟動他自己的子程序。

這些子shell(即子程序)使指令碼並行地，有效率地同時執行指令碼內的多個子任務。

在ubuntu中sh只是bash的一個連結。

由於是在子shell中執行，指令碼設定的變數不會影響當前shell。

3、./的命令用法：

./FileName

作用:開啟一個子shell來讀取並執行FileName中命令。該filename檔案需要"執行許可權"

注：執行一個shell指令碼時會啟動另一個命令直譯器。

每個shell指令碼有效地執行在父shell(parent shell)的一個子程序裡。

這個父shell是指一個控制終端或在一個xterm視窗中給你命令指示符的程序。

shell指令碼也可以啟動他自己的子程序。

這些子shell(即子程序)使指令碼並行地，有效率地同時執行指令碼內的多個子任務。

由於是在子shell中執行，指令碼設定的變數不會影響當前shell。

exec和source都屬於bash內部命令 (builtins commands)，在bash下輸入man exec或man source可以檢視所有的內部命令資訊。

bash shell的命令分為兩類：外部命令和內部命令。外部命令是通過系統呼叫或獨立的程式實現的，如sed、awk等等。內部命令是由特殊的檔案格式(.def)所實現，如cd、history、exec等等。

在說明exe和source的區別之前，先說明一下fork的概念。

fork是linux的系統呼叫，用來建立子程序 (child process)。子程序是父程序(parent process)的一個副本，從父程序那裡獲得一定的資源分配以及繼承父程序的環境。子程序與父程序唯一不同的地方在於pid (process id)。

環境變數 (傳給子程序的變數，遺傳性是本地變數和環境變數的根本區別) 只能單向從父程序傳給子程序。不管子程序的環境變數如何變化，都不會影響父程序的環境變數。

shell script:

有兩種方法執行shell scripts，一種是新產生一個shell，然後執行相應的shell scripts；一種是在當前shell下執行，不再啟用其他shell。

新產生一個shell然後再執行scripts的方法是在scripts檔案開頭加入以下語句

```
#!/bin/sh
```

一般的script檔案(.sh)即是這種用法。這種方法先啟用新的sub-shell (新的子程序)，然後在其下執行命令。

另外一種方法就是上面說過的source命令，不再產生新的shell，而在當前shell下執行一切命令。

source:

source命令即點(.)命令。

在bash下輸入man source，找到source命令解釋處，可以看到解釋"Read and execute commands from filename in the current shell environment and ...".從中可以知道，source命令是在當前程序中執行引數檔案中的各個命令，而不是另起子程序(或sub-shell)。

exec:

在bash下輸入man exec，找到exec命令解釋處，可以看到有"No new process is created."這樣的解釋，這就是說exec命令不產生新的子程序。那麼exec與source的區別是什麼呢？

exec命令在執行時會把當前的shell process關閉，然後換到後面的命令繼續執行。

* fork (/directory/script.sh)

fork是最普通的，就是直接在腳本里面用/directory/script.sh來呼叫script.sh這個指令碼。執行的時候開一個sub-shell執行呼叫的指令碼，sub-shell執行的時候，parent-shell還在。sub-shell執行完畢後返回parent-shell。sub-shell從parent-shell繼承環境變數。但是sub-shell中的環境變數不會帶回parent-shell

* source (source /directory/script.sh)

與fork的區別是不新開一個sub-shell來執行被呼叫的指令碼，而是在同一個shell中執行。所以被呼叫的指令碼中宣告的變數和環境變數，都可以在主指令碼中得到和使用。

* exec (exec /directory/script.sh)

exec與fork不同，不需要新開一個sub-shell來執行被呼叫的指令碼。被呼叫的指令碼與父指令碼在同一個shell內執行。但是使用exec呼叫一個新指令碼以後，父指令碼中exec行之後的內容就不會再執行了。這是exec和source的區別。

下面用一個例子來講解

1.sh

```
#!/bin/bash

A=B

echo "PID for 1.sh before exec/source/fork:$"
export A

echo "1.sh: \$A is $A"

case $1 in
    exec)
        echo "using exec..."
        exec ./2.sh ;;
    source)
        echo "using source..."
        . ./2.sh ;;
    *)
        echo "using fork by default..."
        ./2.sh ;;
esac

echo "PID for 1.sh after exec/source/fork:$"
echo "1.sh: \$A is $A"
```

2.sh

```
#!/bin/bash

echo "PID for 2.sh: $"
echo "2.sh get \$A=$A from 1.sh"

A=C

export A

echo "2.sh: \$A is $A"
```

下面在命令列中去執行

./1.sh fork

```
[root@localhost tmp]# ./1.sh fork
PID for 1.sh before fork/source/exec is 5344
1.sh:$A is B
using fork...
PID for 2.sh is 5345
2.sh get $A=B from 1.sh
2.sh:$A is C
PID for 1.sh after fork/source/exec is 5344
1.sh:$A is B
```

可以看到，1.sh是在父程序中執行，2.sh是在子程序中執行的，父程序的PID是5344，而子程序的是5345，當子程序執行完畢後，控制權返回到父程序。同時，在子程序改變環境變數A的值不會影響到父程序。

./1.sh source

```
[root@localhost tmp]# ./1.sh source
PID for 1.sh before fork/source/exec is 5367
1.sh:$A is B
using source...
PID for 2.sh is 5367
2.sh get $A=B from 1.sh
2.sh:$A is C
PID for 1.sh after fork/source/exec is 5367
1.sh:$A is C
```

由結果可知，1.sh和2.sh都是在同一程序中執行的，PID為5367

./1.sh exec

```
[root@localhost tmp]# ./1.sh exec
PID for 1.sh before fork/source/exec is 5371
1.sh:$A is B
using exec...
PID for 2.sh is 5371
2.sh get $A=B from 1.sh
2.sh:$A is C
[root@localhost tmp]#
```

可知，兩個指令碼都是在同一程序中執行，但是請注意，使用exec終止了原來的父程序，因此，可以看到

```
echo "PID for 1.sh after exec/source/fork:$"

echo "1.sh: $A is $A"
```

這兩個命令沒有執行

由這個例子，便大致可瞭解它們的區別了

【Shell】date 做 timestamp 格式轉換計算

要將日期與 timestamp 互相做格式轉換，只要使用 date 就可以達成，date 還可以順便做些日期的加減換算。

- 日期轉換成 timestamp
 - `$ date -d "2018-02-08" +%s`
1518019200
- timestamp 轉換成可識別的時間
 - `$ date --date=@1518019200`
四 2月 8 00:00:00 CST 2018
 - `$ date "+%Y-%m-%d" --date=@1518019200 # 指定格式`
2018-02-08
 - `$ date "+%Y-%m-%d" # 直接秀今天的日期`
- 日期加減換算
 - `$ date -d "-5 day" "+%Y-%m-%d" # 5天前的日期`

<來源>[Linux 使用 date 做 timestamp 格式轉換計算 - Tsung's Blog \(longwin.com.tw\)](http://longwin.com.tw)

【Shell】awk 取出特定欄位

- AWK 要取得最後一個欄位的值是使用：**`$NF`**
- AWK 要取得倒數第二個欄位的值是使用：**`$(NF-1)`**
- 由後往前取，就繼續 `-2`、`-3`... 下去即可
- 範例：**`awk '{print $(NF - 2)}'`** # 從後面數來第二個

所以要抓 access.log 倒數第二欄 和 第二欄，範例如下：

- `cat access.log | awk -F" '{print $(NF-2),$2}' # 使用 " 切割`

想要過濾掉其他欄位，主要就是把不要的欄位設定為「空」，再來印出 `$0` 就可以了～

- `$ cat access.log | awk '{$1=$2=$8=""; print $0}'`

cut 也可以達成：

- `$ cat access.log | cut --complement -d" " -f 1,2,8`

- [使用 AWK 過濾不要的欄位 - Tsung's Blog \(longwin.com.tw\)](#)
- [AWK 取得欄位的「最後」與「倒數第二個值」的方法 - Tsung's Blog \(longwin.com.tw\)](#)

shell

【Shell】取得public ip

```
#curl ifconfig.me; echo  
175.99.111.111
```

【shell】 內建變數

\$BASH_ENV	absolute path of startup file
\$CDPATH	directories searched by cd
\$FCEDIT	absolute path of history editor
\$HISTCMD	the history number of the current command
\$HISFILE	absolute path of history file
\$HISTSIZE	number of remembered commands
\$HOME	login directory
\$IFS	token delimiters
\$LINENO	current line number in shell script
\$LINES	terminal height
\$MAIL	absolute path of mailbox
\$MAILCHECK	number of seconds to check mail
\$OLDPWD	absolute path of previous directory
\$OPTARG	option set by getopt
\$OPTIND	option's ordinal position set by getopt
\$OSTYPE	the OS on which bash is executing
\$PATH	command search path
\$PPID	process ID of parent
\$PS1	primary prompt
\$PS2	secondary prompt
\$PWD	absolute path of current directory
\$RANDOM	random integer
\$REPLY	default variable for read
\$SECONDS	number of seconds since shell started
\$SHELL	absolute pathname of preferred shell
\$TMOUT	seconds to log out after lack of use
\$UID	user ID of the current user
\$\$	process ID of current shell
\$?	exit status of most recent statement
\$#	參數的數目
\$*	代表所有參數
\$!	PID of the most recently started background job

shell

【Shell】動態呼叫awk自訂函數

*test.sh

```
#!/bin/bash
awkshell="/tmp/newdf.$$"
#awkshell="/tmp/newdf"
trap "rm -f $awkshell" exit

cat << EOF > $awkshell
function showunit(size){
    mb = size / 1024;
    gb = mb / 1024;
    if ( size < 1024 || substr(size,1,1) !~ "[0-9]" || substr(size,2,1) !~ "[0-9]" ){ return size }
    else if ( mb < 1 ){ return size"K" }
    else if ( gb < 1 ){ return (int(mb*100)/100)"M" }
    else { return (int(gb*100)/100)"G" }
}
!/Filesystem/{
    size=showunit($1)
    print size
}
EOF

#echo $awkshell
#cat $awkshell
echo $1 | awk -f $awkshell

exit 0
```

```
[root]$ sh test.sh 6666
6.5M
[root]$ sh test.sh 66666
65.1M
[root]$ sh test.sh 666666666
635.78G
```

【Shell】【Tool】測試機器防火牆是否開通

<https://medium.com/%E8%BC%95%E9%AC%86%E5%B0%8F%E5%93%81-pks%E8%88%87k8s%E7%9A%84%E9%BB%9E%E6%BB%B4/firewall-proxy-i-hate-you-5923c77266d5>

```
#!/bin/bash
services=(
"gcr.io"
"checkpoint-api.hashicorp.com"
"storage.googleapis.com monitoring.googleapis.com"
"gkeconnect.googleapis.com"
"iam.googleapis.com"
"dl.google.com"
"gcr.io"
"googleusercontent.com"
"www.googleapis.com"
"accounts.google.com"
"cloudresourcemanager.googleapis.com"
"container.googleapis.com"
"gkeconnect.googleapis.com"
"gkehub.googleapis.com"
"iam.googleapis.com"
"iamcredentials.googleapis.com"
"logging.googleapis.com"
"monitoring.googleapis.com"
"oauth2.googleapis.com"
"securetoken.googleapis.com"
"servicecontrol.googleapis.com"
"serviceusage.googleapis.com"
"storage.googleapis.com"
"stackdriver.googleapis.com"
"sts.googleapis.com"
"checkpoint-api.hashicorp.com"
"releases.hashicorp.com"
"this-url-wont-work.comx"
)
for s in ${services[@]}; do
  nc -v -z -w 3 $s 443 && echo "$s:443 Online" || echo "$s:443 Offline"
done
```

結果：

```
gcr.io:443 Online
checkpoint-api.hashicorp.com:443 Online
storage.googleapis.com:443 Online
monitoring.googleapis.com:443 Online
gkeconnect.googleapis.com:443 Online
iam.googleapis.com:443 Online
dl.google.com:443 Online
gcr.io:443 Online
googleusercontent.com:443 Offline
www.googleapis.com:443 Online
accounts.google.com:443 Online
cloudresourcemanager.googleapis.com:443 Online
container.googleapis.com:443 Online
gkeconnect.googleapis.com:443 Online
gkehub.googleapis.com:443 Online
iam.googleapis.com:443 Online
iamcredentials.googleapis.com:443 Online
logging.googleapis.com:443 Online
monitoring.googleapis.com:443 Online
oauth2.googleapis.com:443 Online
```

securetoken.googleapis.com:443 Online
servicecontrol.googleapis.com:443 Online
serviceusage.googleapis.com:443 Online
storage.googleapis.com:443 Online
stackdriver.googleapis.com:443 Online
sts.googleapis.com:443 Online
checkpoint-api.hashicorp.com:443 Online
releases.hashicorp.com:443 Online
this-url-wont-work.comx:443 Offline

【Shell】 【Tool】 批次更新密碼

如果有許多使用者需要批次重設密碼並強制其在下次登入時更新，可以使用一個簡單的 Shell 腳本來自動化這個過程。假設你有一個包含所有目標使用者名稱的文字檔，例如 `users.txt`，可以按照以下步驟操作：

1. 建立使用者清單：

在 `users.txt` 中列出所有需要重設密碼的使用者名稱，每行一個。例如：

```
user1
user2
user3
```

2. 編寫批次重設密碼的腳本：

建立一個 Shell 腳本，批次執行密碼重設並強制更新。例如，將以下內容儲存為 `reset_password.sh`：

```
#!/bin/bash

# 檢查 users.txt 是否存在
if [[ ! -f users.txt ]]; then
    echo "Error: users.txt not found!"
    exit 1
fi

# 循環讀取每一個使用者
while IFS= read -r username; do
    # 跳過空行
    if [[ -z "$username" ]]; then
        continue
    fi

    # 設定預設密碼或隨機密碼
    new_password="DefaultPassword123" # 或使用 `new_password=$(openssl rand -base64 12)` 生成隨機密碼
    echo "重設 $username 的密碼為 $new_password"

    # 重設密碼
    echo "$username:$new_password" | sudo chpasswd

    # 強制下次登入時更改密碼
    sudo chage -d 0 "$username"

    echo "$username 的密碼已重設並強制更新"
done < users.txt
```

3. 執行腳本：

為腳本增加執行權限，然後執行：

```
chmod +x reset_password.sh
sudo ./reset_password.sh
```

4. 檢查結果：

執行完成後，你可以再次使用 `chage -l username` 指令逐一檢查使用者的密碼到期設定，確認是否成功。

這樣便可批次重設多位使用者的密碼，並強制其在下次登入時更新密碼。

【Shell】 【Tip】 列出環境變數網址解析dns

```

列出所有環境變數並解析疑似網址的 IP：
=====
目前使用的 DNS server： 67.207.67.3 67.207.67.2
-----
C_HOST=https://www.microsoft.com/zh-tw/#bb
↳ www.microsoft.com → 184.51.98.103
-----
B_HOST=https://tw.yahoo.com/a?a=a&b=b
↳ tw.yahoo.com → 180.222.114.12
-----
A_HOST=https://www.google.com/a/b
↳ www.google.com → 74.125.68.99
-----

系統 /etc/hosts 內容（不含註解與空行）：
=====
127.0.1.1 ubuntu-s-1vcpu-1gb ubuntu-s-1vcpu-1gb
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

```

```

#!/bin/bash

export A_HOST="https://www.google.com/a/b"
export B_HOST="https://tw.yahoo.com/a?a=a&b=b"
export C_HOST="https://www.microsoft.com/zh-tw/#bb"

echo "列出所有環境變數並解析疑似網址的 IP："
echo "===== "

# 顯示目前 DNS server
# 嘗試從 systemd-resolved 的配置中找出真實 DNS
if command -v resolvectl >/dev/null 2>&1; then
    dns_servers=$(resolvectl status | grep 'DNS Servers' | head -n 1 | awk -F: '{print $2}')
#elif [ -f /run/systemd/resolve/resolv.conf ]; then
# dns_servers=$(grep -E '^nameserver' /run/systemd/resolve/resolv.conf | awk '{print $2}')
else
    dns_servers=$(grep -E '^nameserver' /etc/resolv.conf | awk '{print $2}')
fi

echo "目前使用的 DNS server：$dns_servers"
echo "-----"

# 暫存解析過的域名，避免重複查詢
declare -A resolved

while IFS='=' read -r name value; do
    if [[ "$value" =~ ^https?:/[[:^:]]+ ]]; then
        host="${BASH_REMATCH[1]}"
    elif [[ "$value" =~ ^([a-zA-Z0-9-]+\.[a-zA-Z]{2,})$ ]]; then
        host="${BASH_REMATCH[1]}"
    else
        continue
    fi

    if [[ -n "$host" && -z "${resolved[$host]}" ]]; then

```

```
ip=$(dig +short "$host" | grep -Eo '([0-9]{1,3}\.){3}[0-9]{1,3}' | head -n 1)
resolved["$host"]="$ip"
echo "$name=$value"
if [[ -z "$ip" ]]; then
    echo "↳ $host → [無法解析]"
else
    echo "↳ $host → $ip"
fi
echo "-----"
fi
done < <(env)

echo ""
echo "系統 /etc/hosts 內容（不含註解與空行）："
echo "===== "
grep -vE '^\s*#|^\s*$' /etc/hosts
```

【茶包射手】

【茶包射手】

version `GLIBCXX_3.4.20' not
found 解决方法

<https://www.jianshu.com/p/050b2b777b9d>

【cron】沒有依排程執行

檢查服務

```
# systemctl status crond
● crond.service - Command Scheduler
   Loaded: loaded (/usr/lib/systemd/system/crond.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2024-04-22 13:11:51 CST; 7min ago
     Main PID: 17677 (crond)
        CGroup: /system.slice/crond.service
                └─17677 /usr/sbin/crond -n

Apr 22 13:11:51 localhost.localdomain systemd[1]: Started Command Scheduler.
Apr 22 13:11:51 localhost.localdomain crond[17677]: (CRON) INFO (RANDOM_DELAY will be scaled with factor 74% if used.)
Apr 22 13:11:51 localhost.localdomain crond[17677]: (CRON) INFO (running with inotify support)
Apr 22 13:11:51 localhost.localdomain crond[17677]: (CRON) INFO (@reboot jobs will be run at computer's startup.)

# 如果是ubuntu服務是 cron

# 如果沒有起請將服務起來
# systemctl start crond
# 設定開機自動起服務
# systemctl enable crond
```

檢查log

```
# 檢查cron log
cat /var/log/cron

# 檢查sys log
grep CRON /var/log/messages
```

【Top】效能監控工具

```
top - 10:51:26 up 19 days, 44 min, 0 users, load average: 0.43, 0.19, 0.16
Tasks: 351 total, 1 running, 350 sleeping, 0 stopped, 0 zombie
%Cpu0  : 1.0 us, 0.3 sy, 0.0 ni, 98.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu1  : 1.0 us, 0.3 sy, 0.0 ni, 98.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu2  : 1.3 us, 1.0 sy, 0.0 ni, 97.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu3  : 0.3 us, 0.3 sy, 0.0 ni, 99.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu4  : 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu5  : 1.0 us, 0.7 sy, 0.0 ni, 98.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu6  : 1.0 us, 0.7 sy, 0.0 ni, 98.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu7  : 0.7 us, 0.3 sy, 0.0 ni, 99.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu8  : 0.3 us, 0.0 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu9  : 0.3 us, 1.0 sy, 0.0 ni, 98.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu10 : 0.3 us, 0.3 sy, 0.0 ni, 99.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu11 : 0.7 us, 0.3 sy, 0.0 ni, 99.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu12 : 0.0 us, 0.3 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu13 : 0.3 us, 0.7 sy, 0.0 ni, 99.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu14 : 0.0 us, 0.3 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu15 : 0.3 us, 0.0 sy, 0.0 ni, 99.3 id, 0.3 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 15877032 total, 2425864 free, 8958068 used, 4493100 buff/cache
KiB Swap: 1044476 total, 1029220 free, 15256 used, 6562812 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5305	1000	20	0	10.3g	1.4g	28572	S	5.0	9.0	77:46.06	java
15910	1000	20	0	10.6g	1.8g	76656	S	3.0	11.9	343:59.06	java
14429	1000	20	0	11.2g	409404	23696	S	2.0	2.6	146:30.24	node
22134	root	20	0	1011308	105412	19524	S	1.7	0.7	0:11.91	node
25810	root	20	0	162256	2532	1584	R	1.0	0.0	0:00.10	top
1159	root	20	0	2256000	99192	29200	S	0.7	0.6	138:01.18	dockerd
2496	polkitd	20	0	1588364	130232	20564	S	0.7	0.8	141:32.39	mongod
2810	polkitd	20	0	1631640	152044	19940	S	0.7	1.0	187:11.83	mongod
25470	root	20	0	2562424	1.6g	19604	S	0.7	10.5	10:27.94	node
25513	root	20	0	1117752	45796	15912	S	0.7	0.3	4:06.74	node
9	root	20	0	0	0	0	S	0.3	0.0	19:27.34	rcu_sched
1043	root	20	0	1744564	44368	15508	S	0.3	0.3	85:02.00	containerd
1	root	20	0	191296	4268	2616	S	0.0	0.0	6:32.92	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.50	kthreadd
4	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
6	root	20	0	0	0	0	S	0.0	0.0	0:07.43	ksoftirqd/0

```
top - 16:12:13 up 42 days, 37 min, 6 users, load average: 1.66, 1.67, 2.39
Tasks: 255 total, 8 running, 247 sleeping, 0 stopped, 0 zombie
Cpu0 : 31.4%us, 51.5%sy, 0.0%ni, 10.0%id, 0.0%wa, 0.0%hi, 6.7%si, 0.3%st
Cpu1 : 89.0%us, 9.0%sy, 0.0%ni, 2.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu2 : 31.4%us, 51.8%sy, 0.0%ni, 15.4%id, 0.0%wa, 0.0%hi, 1.3%si, 0.0%st
Cpu3 : 34.2%us, 52.5%sy, 0.0%ni, 7.6%id, 0.0%wa, 0.0%hi, 5.6%si, 0.0%st
Mem: 16269604k total, 15986032k used, 283572k free, 1069332k buffers
Swap: 4193272k total, 0k used, 4193272k free, 3629176k cached
```

Linux上的top命令是一個非常有用的工具，它可以顯示系統中正在運行的進程的詳細信息，包括CPU、記憶體、磁盤和網路使用情況。以下是一些使用top命令的基本說明：

- 開啟top：在終端窗口中輸入`top`並按下Enter鍵，即可開啟top命令的監視模式。
- 介面說明：top的介面由多個欄位組成，每個欄位提供了特定的系統信息。其中一些常見的欄位包括：
 - PID：進程ID。
 - USER：執行進程的使用者名稱。
 - PR：進程的優先級。
 - NI：進程的優先級數值。
 - VIRT：進程使用的虛擬記憶體大小。
 - RES：進程使用的實體記憶體大小。
 - SHR：進程共享的記憶體大小。
 - %CPU：進程使用的CPU百分比。
 - %MEM：進程使用的記憶體百分比。
 - TIME+：進程運行的總時間。
 - COMMAND：進程的命令名稱。
- 排序方式：預設情況下，top按照CPU使用百分比（%CPU）進行排序。你可以按下鍵盤上的`<`或`>`鍵來改變排序方式。例如，按下`<`鍵可以按照進程ID（PID）進行排序，按下`>`鍵可以按照記憶體使用百分比（%MEM）進行排序。
- 刷新頻率：預設情況下，top每隔3秒刷新一次顯示的進程列表。你可以按下鍵盤上的`s`鍵來改變刷新頻率，然後輸入所需的秒數。
- 執行命令：在top界面中，你可以執行一些特定的命令，例如：

- `k` : 終止選中的進程。輸入進程的PID並按下Enter鍵，然後選擇終止原因。
- `r` : 修改進程的優先級。輸入進程的PID並按下Enter鍵，然後輸入新的優先級數值。
- `l` : 切換到全局(CPU)總覽模式。
- `h` : 顯示top命令的幫助信息。

6. q :退出top

Help 說明

Z、B、E、e : 全局設置 : 'Z' 切換彩色顯示 ; 'B' 切換粗體顯示 ; 'E' 或 'e' 調整總結/任務記憶體體的刻度。

l、t、m : 切換總結顯示 : 'l' 切換載入平均值 ; 't' 切換任務/ CPU 狀態 ; 'm' 顯示記憶體信息。

0、1、2、3、l : 切換顯示 : '0' 切換顯示 0 值 ; '1/2/3' 切換 CPU 或 NUMA 點視圖 ; 'l' 切換到 Irix 模式。

f、F、X : 欄位設置 : 'f'/'F' 添加/移除/調整/排序欄位 ; 'X' 增加固定寬度。

L、&、<、> : 定位操作 : 'L'/'&' 查找/重複查找 ; 移動排序列 : '<'/'>' 左移/右移。

R、H、V、J : 切換設置 : 'R' 切換排序方式 ; 'H' 切換線程顯示 ; 'V' 切換樹狀視圖 ; 'J' 切換數字對齊。

c、i、S、j : 切換設置 : 'c' 切換命令名稱/命令行顯示 ; 'i' 切換空間時間顯示 ; 'S' 切換時間顯示 ; 'j' 切換字符串對齊。

x、y : 切換突出顯示 : 'x' 切換排序欄位的突出顯示 ; 'y' 切換運行的任務突出顯示。

z、b : 切換設置 : 'z' 切換彩色/單色顯示 ; 'b' 切換粗體/反轉顯示 (僅在 'x' 或 'y' 選項下) 。

u、U、o、O : 過濾條件 : 'u'/'U' 過濾按有效/任意使用者 ; 'o'/'O' 過濾其他條件。

n、#、^O : 設定顯示的最大任務數量 : 'n'/'#' 設定最大顯示任務數 ; 按下 Ctrl+'O' 顯示其他過濾條件。

C、... : 切換滾動座標顯示 : 'up'、'down'、'left'、'right'、'home'、'end'

k、r : 操作任務 : 'k' 終止任務 ; 'r' 修改任務的優先級。

d 或 s : 設定更新間隔。

W、Y : 寫入配置文件 'W' ; 檢查其他輸出 'Y' 。

load average 負載平均

load average: 1.66, 1.67, 2.39

最近 1分鐘 / 5分鐘 / 15分鐘

單核cpu來看，load值大於1代表系統滿載，load值為0代表完全空間，理想數字在**0.7**以下會比較好
如果是4核心則為 $0.7 * 4 = 2.8$ 以下比較好

參考

<https://www.itread01.com/content/1548662434.html>

us: is meaning of "user CPU time"
sy: is meaning of "system CPU time"
ni: is meaning of " nice CPU time"
id: is meaning of "idle"
wa: is meaning of "iowait"
hi : is meaning of "hardware irq"
si : is meaning of "software irq"
st : is meaning of "steal time"

Top 排序

在Linux中，`top`命令可以用查看系統的實時性能信息。默認情況下，`top`的顯示界面按照CPU使用率降序排列。但是，你可以通過按鍵改變排序方式。以下是一些常用的排序：

1. **按CPU使用率排序**：

- 按下 **Shift + P**

2. **按內存使用率排序**：

- 按下 **Shift + M**

3. ****按累CPU使用率排序：****

- 按下 `t`，然后按 `%CPU`

4. ****按累内存使用率排序：****

- 按下 `t`，然后按 `%MEM`

5. ****按PID（进程ID）排序：****

- 按下 `Shift + N`

6. ****按进程名排序：****

- 按下 `c`

7. ****按照其他列排序：****

- 在 `top` 的显示界面中，你可以使用左右箭头选择不同的列，然后按下 `o` 要选择排序的列。

8. ****切换升序/降序排序：****

- 在排序之后，你可以按下 `R` 来切换排序的顺序（升序或降序）。

注意，有些选项在 `top` 运行时无效，按下相应的键后，`top` 会根据你的选择重新排序并更新显示。如果你希望在 `top` 启动时就按照特定的排序方式显示，你可以使用 `top` 命令的一些参数，例如：

```
top -o %CPU # 按照 CPU 使用率降序排列
top -o %MEM # 按照内存使用率降序排列
```

这些是一些基本的排序操作，根据你的需求可以选择适合的排序方式。

【資安】強制登出使用者

使用who查詢在線使用者

kill -kill -t {tty} 強制登出

```
[root@jylee-stres-stest250 srou]# who
root      pts/6      2021-07-19 09:06 (:1)
root      pts/7      2021-09-09 09:59 (mopc06-000063.fm.local)
root      pts/9      2021-07-23 09:18 (:7)
srou      pts/10     2021-10-07 09:30 (mopc06-000063.fm.local)
srou      pts/11     2021-10-07 09:44 (:15.0)
jylee     pts/13     2021-09-28 11:44 (:12.0)
wckuo     pts/15     2021-09-28 13:14 (:14.0)
[root@jylee-stres-stest250 srou]# kill -kill -t pts/15
```

【圖形介面】Linux安裝圖形介面

* 安裝圖形介面

```
```\nyum groupinstall -y "GNOME Desktop"
```

```
|run level|<|h
|0|關機(不可設為預設)|
|6|重新開機(不可設為預設)|
|1|單人文字模式|
|2|多人模式文字(無網路)|
|3|多人模式文字(有網路)|
|4|客製化模式3|
|5|多人模式圖形(有網路)|
```

系統使用"targets"取代以前的"run-levels"。下列為主要的兩個設定值：

multi-user.target: 表示 runlevel 3

graphical.target: 表示 runlevel 5

檢查現在系統的預設target的指令：systemctl get-default

```
```\n
```

*查看現在啟動

```
```\n
```

```
[snowflake@www ~]$ sudo systemctl get-default
multi-user.target
```

```
```\n
```

*設定圖形啟動

```
```\n
```

```
[snowflake@www ~]$ sudo systemctl set-default
graphical.target
```

```
```\n
```

*忘記指令可以查看/etc/inittab 檔案

```
```\n
```

```
cat /etc/inittab
inittab is no longer used when using systemd.

ADDING CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.

Ctrl-Alt-Delete is handled by /usr/lib/systemd/system/ctrl-alt-del.target

systemd uses 'targets' instead of runlevels. By default, there are two main targets:

multi-user.target: analogous to runlevel 3
graphical.target: analogous to runlevel 5

To view current default target, run:
systemctl get-default

To set a default target, run:
systemctl set-default TARGET.target

```\n
```

【圖形介面】tiger vnc

安裝

```
yum install tigervnc-server -y
```

VNC 預設 port 為 5900，而 :1 代表一個虛擬桌面，所以連結這個虛擬桌面的 port 為 5901 (5900 + 1) 或 1

複製設定檔

```
cp /lib/systemd/system/vncserver@.service /etc/systemd/system/vncserver@:1.service
```

以下是用root 帳號，如果是一般使用者，可使用註解區塊(使用者jacky，把root註解)

```
[Unit]
Description=Remote desktop service (VNC)
After=syslog.target network.target

[Service]
Type=simple

# Clean any existing files in /tmp/.X11-unix environment
ExecStartPre=/bin/sh -c '/usr/bin/vncserver -kill %i > /dev/null 2>&1 || :'
ExecStop=/bin/sh -c '/usr/bin/vncserver -kill %i > /dev/null 2>&1 || :'

# 一般帳號
# #ExecStart=/usr/sbin/runuser -l jacky -c "/usr/bin/vncserver %i"
# #PIDFile=/home/jacky/.vnc/%H%i.pid
#
# root 帳號
ExecStart=/usr/sbin/runuser -l root -c "/usr/bin/vncserver %i -geometry 1280x1024"
PIDFile=/root/.vnc/%H%i.pid

[Install]
WantedBy=multi-user.target
```

設定VNC密碼

如果是一般使用者要先切換到該身分下

```
su jacky
```

設定vnc 密碼

```
vncpasswd
```

使用root 帳號

```
su -
systemctl daemon-reload
```

啟用服務，並設定開機自動執行

```
systemctl start vncserver@:1.service
systemctl enable vncserver@:1.service
```

可使用vnc

【Linux】帳號管理

新增/刪除帳號

```
# -u 自訂UID
# -g 加入某群組，可使用GID, 或是群組名
# -d 自定使用的家目錄
# -M 不建立家目錄
# -s 自訂shell
# 新增帳號(包含家目錄)
# centos
usradd 新帳號名稱
# ubuntu
adduser 新帳號名稱

# 刪除帳號(保留家目錄)
userdel user1
# 刪除帳號(刪除家目錄)
userdel -r user1

# 修改帳號
usermod [-LU][-c <備註>][[-d <登入目錄>][-e <有效期限>][-f <緩沖天數>][-g <群組>][-G <群組>][-l <帳號名稱>][-s ] [-u ][用戶帳號]
#-c<備註> 修改用戶帳號的備註文字。
#-d登入目錄> 修改用戶登入時的目錄。
#-e<有效期限> 修改帳號的有效期限。
#-f<緩沖天數> 修改在密碼過期後多少天即關閉該帳號。
#-g<群組> 修改用戶所屬的群組。
#-G<群組> 修改用戶所屬的附加群組。
#-l<帳號名稱> 修改用戶帳號名稱。
#-L 鎖定用戶密碼，使密碼無效。
#-s 修改用戶登入後所使用的shell。
#-u 修改用戶ID。
#-U 解除密碼鎖定。

# user1 加入 root 群組
usermod -aG root user1
# user1 移出 root 群組
gpasswd -d root user1
```

設定密碼過期

```
#chage：密碼失效是通過此命令來管理的。
#參數意思：
#-m 密碼可更改的最小天數。為零時代表任何時候都可以更改密碼。
#-M 密碼保持有效的最大天數。
#-W 用戶密碼到期前，提前收到警告信息的天數。
#-E 帳號到期的日期。過了這天，此帳號將不可用。
#-d 上一次更改的日期
#-i 停滯時期。如果一個密碼已過期這些天，那麼此帳號將不可用。
#-l 例出當前的設置。由非特權用戶來確定他們的密碼或帳號何時過期。

# 設定密碼過期時間0天(使用者登入需更改密碼)
sudo chage -d 0 使用者名稱
# 設定密碼永不過期
sudo chage -M -1 使用者名稱
# 設定帳號不會過期
sudo usermod -e '' 使用者名稱
# 設定帳號密碼都不會過期
sudo chage -M -1 -E -1 使用者名稱
```

禁止 root ssh 登入


```
# 修改/etc/ssh/sshd_config
vim /etc/ssh/sshd_config

# PermitRootLogin yes
# 改為
PermitRootLogin no

# 重新啟動 ssh服務
systemctl restart sshd
```

群組相關

```
# 新增群組
# groupadd [-g GID] 群組名
groupadd group1

# 刪除群組
# groupdel群組名
groupdel group1

# 查詢使用者群組
# 方法一 groups 使用者
groups webmaster
#####/*
[root@test webmon]# groups webmaster
webmaster : webmaster nginx
#####*/
# 方法二 cat /etc/group
cat /etc/group | grep webmaster
#####/*
[root@webmon205 webmon]# cat /etc/group | grep webmaster
webmaster:x:1000:webmaster
nginx:x:979:webmaster
#####*/

# 將使用者加入某群組
# ex.將user1加入ftp群組
usermod -a -G ftp user1
# ex.將user1移出ftp群組
gpasswd -d ftp app

# 修改主群組
# ex.將user1主群組改為nginx
usermod -g nginx user1
```

【Linux】記憶體相關

<https://huenlil.pixnet.net/blog/post/26822270>

釋放記憶體

```
echo 1 > /proc/sys/vm/drop_cache
```

釋放dentries、inodes所用的 cache memory

```
echo 2 > /proc/sys/vm/drop_caches
```

釋放pagecache、dentry、inode 所用的 cache memory

```
echo 3 > /proc/sys/vm/drop_caches
```

完全釋放cache memory，**必須先執行sync**，避免錯誤。

```
sync
```

```
#在釋放記憶體後再將/proc/sys/vm/drop_caches的值設為0  
echo 0 > /proc/sys/vm/drop_caches
```

釋放swap

```
#此swap 在/dev/cobd1  
swapoff /dev/cobd1;swapon /dev/cobd1
```

常用記憶體指令

```
free
```

```
vmstat
```

```
top
```

```
watch cat /proc/meminfo
```

【crontab】排程相關

使用nginx 設定排程

```
sudo su -c "crontab -e" nginx -s /bin/bash
```

【vim】快速鍵

1. vi 編輯器中跳到檔案的第一行：

- a 輸入 :0 或者 :1 回車
- b 鍵盤按下 小寫 gg

2. vi 編輯器跳到檔案最後一行：

- a 輸入 :\$ 回車
- b 鍵盤按下大寫 G
- c 鍵盤按 shift + g （其實和第二種方法一樣）

Vim快速移動游標至行首和行尾

1、需要按行快速移動游標時，可以使用鍵盤上的編輯鍵Home，快速將游標移動至當前行的行首。除此之外，也可以在命令模式中使用快捷鍵"^"（即Shift+6）或0（數字0）。

2、如果要快速移動游標至當前行的行尾，可以使用編輯鍵End。也可以在命令模式中使用快捷鍵"\$"（Shift+4）。與快捷鍵"^"和0不同，快捷鍵"\$"前可以加上數字表示移動的行數。例如使用"1\$"表示當前行的行尾，"2\$"表示當前行的下一行的行尾。

Vim多行註解

多行注：

- 1. 入命令行模式，按ctrl + v入 visual block模式（可视快模式），然后按j, 或者k中多行，把需要注的行起
- 2. 按大字母l，再插入注符，例如//
- 3. 按esc就会全部注了（我的是按两下）

取消多行注：

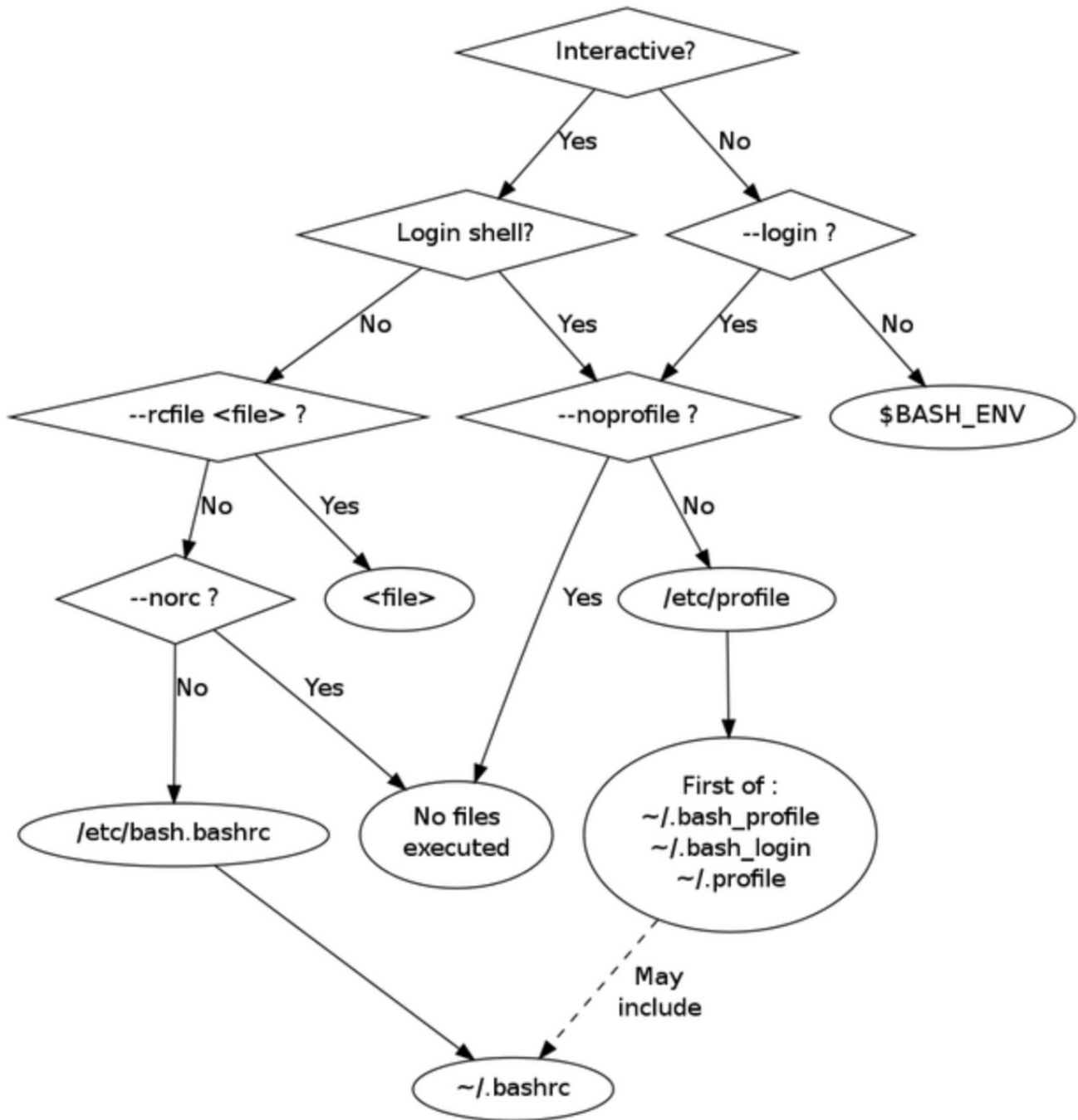
- 1. 入命令行模式，按ctrl + v入 visual block模式（可视快模式），按小字母l向中列的个，例如 // 需要中2列
- 2. 按字母j，或者k中注符号
- 3. 按d就可全部取消注

【Linux】時區設定

Centos7

```
timedatectl set-timezone Asia/Taipei
```

【Linux】載入順序



<https://blog.miniasp.com/post/2021/07/26/Bash-and-Zsh-Initialization-Files>

- 這張圖的第一層 `Interactive?` 的意思
Yes: 當你透過 Console 或 SSH 登入 Linux 主機，預設會進入**互動模式** (Interactive shell)，也就是這裡說的 **Interactive** 的意思。
No: 任何透過 `bash -c '<command>'` 去執行的腳本，就屬於 **非互動模式** (Non-Interactive shell) 的執行。
- 這張圖第二層的 `Login shell?` 的意思
基本上，透過 Console 或 SSH 登入 Linux 主機時，這個 Shell 就跑在所謂的**Login shell** 模式下！
不過，當你透過 SSH 遠端執行一個命令，此時就不會啟動 **Login shell** 模式。而直接呼叫一個使用 Bash 執行的腳本，也不是 **Login shell** 模式。例如以下命令：

```
ssh user@host <COMMAND>
```

- 這張圖第二層的 `--login ?` 的意思
就算你的 Bash 不是執行在 Login shell 模式，你一樣可以在呼叫 `/bin/bash` 的時候特別加上 `--login` 參數，這樣也可以被視為是 **Login shell** 模式。
例如以下這段範例，就會被視為使用 Login shell 模式啟動：

```
#!/bin/bash --login
```

- 這張圖第三層的 `$BASH_ENV` 是什麼環境變數？
當你將 Bash 啟動在**非互動模式**，也沒有特別加上 `--login` 參數的情況，同時這也是大多數執行的預設值，Bash 會優先尋找目前的環境變數中有沒有一個名為 `$BASH_ENV` 的變數，這個變數其實是指向一個檔案路徑。你從 `man bash` 可以發現，Bash 在**非互動模式**啟動的時候，預設會執行以下命令：

```
if [ -n "$BASH_ENV" ]; then . "$BASH_ENV"; fi
```

這也意味著，他會去 sourcing `"$BASH_ENV"` 這個檔案！

- 這張圖第三層的 `--noprofile` 是什麼參數？
執行 `/bin/bash` 的時候，可以額外加上 `--noprofile` 參數，加上之後就不會載入任何啟動檔。
如果你沒有加上 `--noprofile` 參數，也是一般大多數命令的執行方式，預設是會先載入系統全域的 `/etc/profile` 檔案。

⚡ 注意：系統全域的 `/etc/profile` 檔案，還會額外載入 `/etc/profile.d/*.sh` 檔案。

然後再從 `$HOME` 目錄下依序找到這三個檔案執行，但重點是，他只會選擇一個檔案來執行，先找到的先執行，後面的就不會執行！

1. `~/.bash_profile`
2. `~/.bash_login`
3. `~/.profile`

由於上述三個檔案，最終只有一個會被執行，所以這絕對是一個潛在的地雷！☹

我在多年前，就曾經因為我的 `$HOME` 目錄下同時出現 `~/.bash_profile` 與 `~/.profile` 而發生系統異常，原來只要目錄中有出現 `~/.bash_profile` 檔案，就再也不會載入 `~/.profile` 檔案啊！！！

⚡ 在 Ubuntu 的作業系統中，預設是看不到 `~/.bash_profile` 檔案的，建議都以 `~/.profile` 為主要的登入啟動檔！

然而，大多數的 `~/.profile` 登入啟動檔，都會在檔案中額外載入 `~/.bashrc` 檔案，因此有些 Bash 相關的環境設定，如 `shopt` 之類的，都會放在 `~/.bashrc` 檔案中。

- 這張圖第三層的 `--rcfile <file>` 是什麼參數？
當你不是以 Login shell 的方式啟動 Bash，啟動時也沒加上 `--login` 參數，他就會去找你有沒有特別加上 `--rcfile <file>` 參數，明確載入你所指定的檔案路徑。

如果你額外加上的是 `--norc` 參數的話，那就代表你完全不想載入系統全域的 `/etc/bash.bashrc` 與 `$HOME` 目錄下的 `~/.bashrc` 檔案。

如果你用 `--rcfile <file>` 參數找不到指定的檔案，或完全沒用 `--rcfile <file>` 或 `--norc` 參數，也是大多數命令預設的方式，那麼 Bash 就會依序載入 `/etc/bash.bashrc` 與 `~/.bashrc` 檔案，兩個檔案都會載入。這種情境下，是不會載入 `~/.bash_profile`，`~/.bash_login` 或 `~/.profile` 檔案的！

可以完整釐清 Bash 所有啟動檔的載入條件與順序，心裡的感覺格外踏實，非常棒！☺

完整的 Zsh 啟動檔載入順序

由於許多 macOS 用戶都採用 Zsh 為主，這邊我也特別研究了一下 Zsh 的啟動檔載入順序，基本上 Zsh 會依據以下順序載入：

1. `~/.zshenv`
任何啟動情境下，都會載入這個檔案，請將各種環境變數請全部設定在這裡。
2. `/etc/zsh/zprofile` 與 `~/.zprofile`
如果執行在 Login shell 才會依序執行 `/etc/zsh/zprofile` 與 `~/.zprofile` 檔案。
3. `/etc/zsh/zshrc` 與 `~/.zshrc`
如果執行在 Interactive 互動模式下，才會依序執行 `/etc/zsh/zshrc` 與 `~/.zshrc` 檔案。
4. `/etc/zsh/zlogin` 與 `~/.zlogin`
如果執行在 Login shell 下，最後才會依序執行 `/etc/zsh/zlogin` 與 `~/.zlogin` 檔案。
5. `~/.zlogout` 與 `/etc/zsh/zlogout`
當你使用 `exit` 或 `logout` 命令登出時，會自動依序執行 `~/.zlogout` 與 `/etc/zsh/zlogout` 檔案。

我只能說，Zsh 的啟動順序實在比 Bash 好理解太多了，也沒什麼地雷！☺

總結幾種常見情境

1. 直接呼叫某個 Shell Script

```
#!/bin/bash
[ -z "$PS1" ] && echo "Non-Interactive" || echo "Interactive"
shopt -q login_shell && echo "Login shell" || echo "Not login shell"
```

- Interactive? ☐ No
- Login shell? ☐ No
- --noprofile? ☐ No

執行時完全不會載入任何新的啟動檔設定！

2. 使用 SSH 登入遠端主機

```
ssh user@host
```

依序載入 `/etc/profile` --> `/etc/bash.bashrc` --> `~/.profile` [-> `~/.bashrc`]

- Interactive? ☐ Yes
- Login shell? ☐ Yes

- `--noprofile?` ☐ No

⚠ 注意: Bash 不會主動載入 `~/bashrc` 執行，而是我們通常在 `~/.profile` 會載入 `~/bashrc` 執行。

3. 使用 SSH 登入遠端主機後，執行 `bash` 命令

```
ssh user@host
```

剛登入，依序載入 `/etc/profile` --> `/etc/bash.bashrc` --> `~/.profile` [-> `~/.bashrc`]

- `Interactive?` ☐ Yes
- `Login shell?` ☐ Yes
- `--noprofile?` ☐ No

然後我們在遠端主機執行 `bash` 命令，進入下一層 Shell 環境：

```
bash
```

登入後執行 `bash` 預設是**互動模式**，會依序載入 `/etc/bash.bashrc` --> `~/.bashrc`

- `Interactive?` ☐ Yes
- `Login shell?` ☐ No
- `--rcfile <file> ?` ☐ No
- `--norc ?` ☐ No

4. 使用 SSH 登入遠端主機後，執行 `bash -c '<command>'` 命令

```
ssh user@host
```

剛登入，依序載入 `/etc/profile` --> `/etc/bash.bashrc` --> `~/.profile` [-> `~/.bashrc`]

- `Interactive?` ☐ Yes
- `Login shell?` ☐ Yes
- `--noprofile?` ☐ No

```
bash -c '[ -z "$PS1" ] && echo "Non-Interactive" || echo "Interactive"
```

登入後執行 `bash -c` 命令屬於非互動模式，而且沒有 `$BASH_ENV` 的情況下，不會載入任何啟動檔！

- `Interactive?` ☐ No
- `Login shell?` ☐ No
- `$BASH_ENV` ☐ No

5. 使用 SSH 執行遠端命令

```
ssh user@host '[ -z "$PS1" ] && echo "Non-Interactive" || echo "Interactive"
```

```
ssh user@host 'shopt -q login_shell && echo "Login shell" || echo "Not login shell"
```

透過 SSH 執行遠端命令，預設將會跑在 **Non-Interactive** 模式下，但還是會依序載入 `/etc/bash.bashrc` --> `~/.bashrc`

- `Interactive?` ☐ No
- `Login shell?` ☐ No

如果你想在執行遠端命令時載入額外的啟動檔，也可以這樣寫：

```
ssh user@host "source /etc/profile; source ~/.profile; /your/script.sh"
```

6. 使用 SSH 執行遠端命令，透過 `bash` 呼叫另一個命令

```
ssh user@host -t 'bash -c ''[ -z "$PS1" ] && echo "Non-Interactive" || echo "Interactive"''
```

```
ssh user@host -t 'bash -c ''shopt -q login_shell && echo "Login shell" || echo "Not login shell"''
```

⚠ 要在單引號(')的字串中間加入一個單引號，必須輸入五個字元 `''''''` 才能代表一個單引號！（噁心的語法）

⚠ 在 ssh 執行時加入 `-t` 參數，可以在遠端執行時取得一個 pseudo-terminal allocation (pts/0) ！

透過 SSH 執行遠端命令，預設將會跑在 **Non-Interactive** 模式下，但還是會依序載入 `/etc/bash.bashrc` --> `~/.bashrc`

- `Interactive?` ☐ No
- `Login shell?` ☐ No

第二層 `bash` 應該是跑在 **Non-Interactive** 模式，但是卻依序載入 `/etc/bash.bashrc` --> `~/.bashrc` 檔案，這部分我還沒辦法理解為什麼會這樣，感覺透過 SSH 執行遠端程式，預設就會載入這兩個檔案！

- `Interactive?` ☐ No
- `Login shell?` ☐ No

底下這段是讓命令跑在 **Interactive** 模式，但載入啟動檔的順序竟然跟 **Non-Interactive** 竟然一樣！

```
ssh user@host -t 'bash -i -c ''[ -z "$PS1" ] && echo "Non-Interactive" || echo "Interactive"''
```

```
ssh user@host -t 'bash -i -c ''shopt -q login_shell && echo "Login shell" || echo "Not login shell"''
```

簡單來說，使用 SSH 直接遠端命令時，使用 `-i` (互動模式) 或不使用 `-i` 對載入啟動檔沒有什麼兩樣，不過都會載入兩次。不過還是有一個地方不同，如果你的**啟動檔**有設定 `alias` 的話，只有使用 `-i` 互動模式才能使用。

例如以下這段命令，就會得到 `bash: ll: command not found` 的錯誤：

```
ssh user@host -t 'll'
```



```
ssh user@host -t bash -c 'll'
```

如果改用 `-i` 來執行，就可以正常執行 `ll` 這個 `alias` 命令：

```
ssh user@host -t bash -i -c 'll'
```

7. 使用 SSH 執行遠端命令，並以 **Login shell** 啟動 Bash

```
ssh user@host bash -l -c 'set'
```

第一層 `bash` 依序載入 `/etc/bash.bashrc` --> `~/.bashrc`
第二層 `bash` 依序載入 `/etc/profile` --> `~/.profile` [-> `~/.bashrc`]

【更新】Centos 7已於2024年6月30日停止維護，更換yum源解決yum404問題

```
mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup  
mv /etc/yum.repos.d/epel.repo /etc/yum.repos.d/epel.repo.backup
```

```
curl -o /etc/yum.repos.d/CentOS-Base.repo \  
http://mirrors.vpnforgame.net/centos/7/CentOS-Base.repo
```

```
curl -o /etc/yum.repos.d/epel.repo \  
http://mirrors.vpnforgame.net/epel/7/epel.repo
```

```
yum clean all
```

```
yum makecache
```

【Linux】DNS 相關

如何知道本機dns server 設定在哪

```
# 查詢/etc/resolv.conf
cat /etc/resolv.conf

# ip 出現 類似nameserver 127.0.0.53
# 代表不設定在本機 使用resolvectl
resolvectl status

#####
Global
  Protocols: -LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
resolv.conf mode: stub

Link 2 (eth0)
  Current Scopes: DNS
    Protocols: +DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
Current DNS Server: 10.10.10.247
DNS Servers: 10.10.10.246 10.10.10.247
```

【Linux】Dns 動態解析

若想實現同一個 DNS Server 根據不同來源 IP 返回不同的結果，這在 DNS 中稱為**視圖（View）**或**基於來源的策略解析**，以下是詳細實現方式。

使用 Bind 實現（最佳選擇）

Bind 支持基於來源 IP 的視圖配置，可以為不同來源的請求返回不同的 DNS 記錄。

配置步驟

1. **安裝 Bind** 安裝 Bind DNS Server：

```
sudo apt install bind9
```

2. **配置視圖（View）** 編輯 Bind 的主配置文件 `/etc/bind/named.conf`，添加基於來源 IP 的視圖。
範例配置：

```
acl "internal-network" {
    192.168.1.0/24;
};

acl "external-network" {
    any;
};

view "internal" {
    match-clients { "internal-network"; };
    zone "example.com" {
        type master;
        file "/etc/bind/db.internal.example.com";
    };
};

view "external" {
    match-clients { "external-network"; };
    zone "example.com" {
        type master;
        file "/etc/bind/db.external.example.com";
    };
};
```

3. **創建區域文件** 根據來源 IP 創建不同的區域文件。例如：

- **內部網絡的區域文件** `/etc/bind/db.internal.example.com`：

```
$TTL 604800
@ IN SOA example.com. admin.example.com. (
    1 ; Serial
    604800 ; Refresh
    86400 ; Retry
    2419200 ; Expire
    604800 ) ; Negative Cache TTL
;
@ IN NS ns.example.com.
@ IN A 192.168.1.1
```

- **外部網絡的區域文件** `/etc/bind/db.external.example.com`：

```
$TTL 604800
@ IN SOA example.com. admin.example.com. (
    1 ; Serial
    604800 ; Refresh
    86400 ; Retry
    2419200 ; Expire
    604800 ) ; Negative Cache TTL
;
@ IN NS ns.example.com.
@ IN A 203.0.113.1
```

4. 重啟 Bind 重啟 Bind 服務：

```
sudo systemctl restart bind9
```

使用 dnsmasq 實現 (輕量級)

配置步驟

1. 安裝 dnsmasq

```
sudo apt install dnsmasq
```

2. 配置條件解析 編輯 `/etc/dnsmasq.conf`，添加基於來源 IP 的條件解析規則：

```
# 為內部網絡配置
dhcp-range=set:internal,192.168.1.0,192.168.1.255
address=/example.com/192.168.1.1

# 為外部網絡配置
dhcp-range=set:external,0.0.0.0,255.255.255.255
address=/example.com/203.0.113.1
```

3. 重啟 dnsmasq

```
sudo systemctl restart dnsmasq
```

使用 PowerDNS (動態解析)

- <https://wiki.freedomstu.com/books/%E9%96%8B%E6%BA%90%E8%BB%9F%E9%AB%94%E5%AE%89%E8%A3%9Cdebian>
- <https://doc.powerdns.com/authoritative/indexTOC.html>
- <https://blog.csdn.net/sdhzdtwhm/article/details/135910057>
- <https://lantian.pub/article/modify-website/powerdns-lua-diy-geodns.lantian/>
- <https://github.com/PowerDNS/pdns/blob/master/docker-compose.yml>

配置 PowerDNS + Lua 脚本

PowerDNS 支持使用 Lua 腳本進行動態解析。

1. 安裝 PowerDNS 和相關組件

```
sudo apt install pdns-server pdns-backend-sqlite3
```

2. 啟用 Lua 腳本支持 在 `/etc/powerdns/pdns.conf` 中啟用 Lua Backend :

```
launch=sqlite3,lua
lua-dns-script=/etc/powerdns/dns.lua
```

3. 編寫 Lua 腳本 創建 `/etc/powerdns/dns.lua`，根據來源 IP 返回不同的結果：

```
function postresolve(remoteip, domain, qtype)
  if string.match(remoteip, "^192%.168%.1%.") then
    return {{qtype="A", content="192.168.1.1", ttl=60}}
  else
    return {{qtype="A", content="203.0.113.1", ttl=60}}
  end
end
```

4. 重啟 PowerDNS

```
sudo systemctl restart pdns
```

測試

使用 `dig` 測試

針對不同的來源 IP 測試返回的解析結果。

- 從內部網絡：

```
dig @<DNS_SERVER_IP> example.com
```

返回內部地址（例如 `192.168.1.1`）。

- 從外部網絡：

```
dig @<DNS_SERVER_IP> example.com
```

返回外部地址（例如 `203.0.113.1`）。

注意事項

- 來源 IP 檢測：**
 - 檢查網絡路由，確保 DNS 伺服器能正確檢測請求的來源 IP。
 - 若有 NAT 設置，需配置正確的來源地址。
- DNS 緩存：**
 - 若存在 DNS 緩存伺服器，可能會干擾條件解析結果。
- 性能：**
 - `Bind` 和 `PowerDNS` 適合高流量環境。
 - `dnsmasq` 適合輕量級應用場景。

這些方案可以靈活地根據來源 IP 實現不同的 DNS 解析結果。

【Linux】【權限管理】【acl】設定/var/www/html for tomcat 可讀寫

以下是如何在 Ubuntu 中設置這些 ACL 權限的步驟。

步驟 1：啟用 ACL 支持

如果 ACL 尚未啟用，請確保相關功能已經安裝並啟用：

1. 安裝 ACL 工具：

```
sudo apt install acl
```

2. 確保文件系統支持 ACL：

```
sudo mount -o remount,acl /
```

步驟 2：設置基本權限

設定 `/var/www/html` 的基本權限：

```
sudo chmod 775 /var/www/html
sudo chown root:root /var/www/html
```

步驟 3：配置 ACL 權限

根據需求逐步配置 ACL：

1. 設置具體使用者和群組的 ACL 權限：

```
sudo setfacl -m u:tomcat:rwX /var/www/html
sudo setfacl -m g::r-x /var/www/html
sudo setfacl -m o::r-x /var/www/html
```

- `u:tomcat:rwX`：給 tomcat 用戶設置 `rwX` 權限。
- `g::r-x`：默認組的權限為 `r-x`。
- `o::r-x`：其他人的權限為 `r-x`。

2. 設置默認 ACL（適用於未來子目錄和文件）：

```
sudo setfacl -d -m u::rwX /var/www/html
sudo setfacl -d -m u:tomcat:rwX /var/www/html
sudo setfacl -d -m g::r-x /var/www/html
sudo setfacl -d -m o::r-x /var/www/html
```

- `-d`：表示設置默認 ACL（對新創建的文件/目錄生效）。

步驟 4：檢查 ACL 設置

使用以下指令檢查 ACL 配置是否正確：

```
getfacl /var/www/html
```

輸出應類似於：

```
# file: var/www/html
# owner: root
# group: root
user::rwX
user:tomcat:rwX
```

```
group::r-x
mask::rwx
other::r-x
default:user::rwx
default:user:tomcat:rwx
default:group::r-x
default:mask::rwx
default:other::r-x
```

ACL 設置的解釋

1. **普通權限**：
 - **user::rwx**：目錄擁有者 `root` 的權限。
 - **user:tomcat:rwx**：為用戶 `tomcat` 設置的權限。
 - **group::r-x**：目錄的群組成員的權限。
 - **other::r-x**：其他用戶的權限。
 - **mask::rwx**：ACL 中權限的上限（最高可設置為 `rwx`）。
2. **默認權限（default: 開頭）**：
 - **default:user::rwx**：默認擁有者的權限。
 - **default:user:tomcat:rwx**：默認為 `tomcat` 設置的權限。
 - **default:group::r-x**：默認群組的權限。
 - **default:other::r-x**：默認其他用戶的權限。

總結指令

以下是一鍵完成配置的指令集合：

```
sudo setfacl -m u:tomcat:rwx /var/www/html
sudo setfacl -m g::r-x /var/www/html
sudo setfacl -m o::r-x /var/www/html
sudo setfacl -d -m u::rwx /var/www/html
sudo setfacl -d -m u:tomcat:rwx /var/www/html
sudo setfacl -d -m g::r-x /var/www/html
sudo setfacl -d -m o::r-x /var/www/html
```

執行後確認權限是否正確，應與您列出的設定一致！

是否需要設定 `-d` 取決於您的需求，以下是有關 **默認 ACL（-d）** 的詳細說明以及什麼情況下需要設定。

什麼是默認 ACL（-d）？

默認 ACL 是一組規則，會自動應用到 **新創建的文件或子目錄** 中。當一個目錄設置了默認 ACL，該目錄下創建的任何新文件或目錄都會繼承這些默認規則。

是否需要設定 -d

1. **需要設定 -d 的情況**
 - 如果您希望 **新創建的文件或子目錄** 自動繼承特定的 ACL 設置（如 `tomcat` 擁有特定權限），那麼您需要設置默認 ACL。例如：
 - `/var/www/html` 是一個共享的目錄，將會持續新增文件。
 - 您希望 `tomcat` 用戶對新增的文件有 `rwx` 權限。

```
sudo setfacl -d -m u:tomcat:rwx /var/www/html
```

當某人創建一個新文件時，`tomcat` 就會自動擁有該文件的權限。

2. **不需要設定 -d 的情況**
 - 如果您僅關心目錄 **當前的文件和目錄**，而不需要未來的新增文件繼承權限。
 - 如果您會手動或通過腳本為新文件設置權限，而不依賴默認 ACL。在這種情況下，您可以省略 `-d`，只設置當前的 ACL 即可：

```
sudo setfacl -m u:tomcat:rwx /var/www/html
```


範例：對比是否設置 `-d` 的效果

假設您在 `/var/www/html` 下創建了兩個文件：

1. **沒有設置 `-d` 的情況**
 - 當您創建新文件時，不會繼承 `tomcat` 的權限。
 - 新文件的權限取決於當前的系統默認值（如 `umask`）。
2. **設置了 `-d` 的情況**
 - 新文件或目錄將繼承默認 ACL。例如，`tomcat` 將擁有對新文件的權限：

```
touch /var/www/html/newfile  
getfacl /var/www/html/newfile
```

結果顯示 `tomcat` 的權限被自動應用。

結論

- **需要持續新增文件/目錄並繼承權限**：建議使用 `-d` 設置默認 ACL。
- **僅對當前文件和目錄設置權限**：可以省略 `-d`。

建議根據場景選擇適合的設置，以滿足項目需求。

【Linux】SSH 跳板機設定

如果你經常需要透過主機A跳到主機B進行操作，可以考慮以下幾種方法來提升效率：

1. SSH 跳板機設定 (ProxyJump 或 ProxyCommand)

你可以在 `~/.ssh/config` 檔案中設定跳板機 (主機A) 和目標主機 (主機B)。

方法一：使用 ProxyJump (推薦，簡潔且較新)

```
Host 主機B
  HostName <主機B的IP或主機名>
  User <使用者名稱>
  ProxyJump <主機A的使用者名稱>@<主機A的IP或主機名>
```

使用這個設定後，你可以直接用 `ssh 主機B` 連接到主機B，不需先登入主機A。

方法二：使用 ProxyCommand (舊版本支援)

```
Host 主機B
  HostName <主機B的IP或主機名>
  User <使用者名稱>
  ProxyCommand ssh <主機A的使用者名稱>@<主機A的IP> -W %h:%p
```

這個設定同樣讓你直接 `ssh 主機B` 連接到主機B。

2. SSH 快速別名設定 (使用別名簡化命令)

你也可以在 `~/.bashrc` 或 `~/.zshrc` 中設定別名，直接執行一個指令來登入主機B。

```
alias sshB='ssh -j <主機A的使用者名稱>@<主機A的IP或主機名> <主機B的使用者名稱>@<主機B的IP或主機名>'
```

之後，你只需要輸入 `sshB` 就能登入主機B。

3. 使用 SSH Multiplexing (重複利用 SSH 連線)

如果你需要頻繁連線到同一個主機，可以開啟 SSH 連線複用功能，減少重複登入的時間：

在 `~/.ssh/config` 加入以下內容：

```
Host *
  ControlMaster auto
  ControlPath ~/.ssh/control-%r@%h:%p
  ControlPersist 10m
```

這樣在第一次登入主機A時，SSH 會保持連線，之後登入主機B就會變得更快。

4. 使用 SSH Agent 轉發 (Agent Forwarding)

如果主機A和主機B需要同一組 SSH 金鑰，可以使用 SSH Agent Forwarding，避免在主機A上存放私鑰：

```
ssh -A <主機A的使用者名稱>@<主機A的IP>
ssh <主機B的使用者名稱>@<主機B的IP>
```

或者在 `~/.ssh/config` 啟用 Agent Forwarding：

```
Host 主機A
ForwardAgent yes
```

這樣就能直接將本地端的 SSH 金鑰轉發到主機A，並用來登入主機B。

5. 使用 `tmux` 或 `screen` (維持長時間工作環境)

如果你的操作過程需要持續很長時間，建議在主機A或B上使用 `tmux` 或 `screen` 來保持工作環境，這樣即使斷線也能恢復。

安裝並啟動 `tmux`：

```
tmux new -s session_name
```

之後可以斷開連線並重新附加：

```
tmux attach -t session_name
```

6. 使用 `Ansible` 自動化操作 (進階)

如果你有固定的操作流程，可以使用 `Ansible` 寫 `Playbook`，批量操作多個主機，包括跳板機情境：

```
- name: Jump to 主機B via 主機A
hosts: 主機B
vars:
  ansible_ssh_common_args: '-o ProxyJump=<主機A的使用者名稱>@<主機A的IP>'
tasks:
  - name: Run shell command
    command: <你的shell指令>
```

使用 `Ansible` 來批量執行，避免手動登入和操作。

這幾個方法可以根據需求選擇，**ProxyJump** 和 **SSH Config** 設定最為方便、簡潔，推薦優先使用！

【Tool】tmux

▢ tmux 簡易說明

tmux (Terminal Multiplexer) 是一個強大的終端複用工具，可以在單一終端視窗中同時管理多個工作階段 (sessions)、視窗 (windows)、與窗格 (panes)。

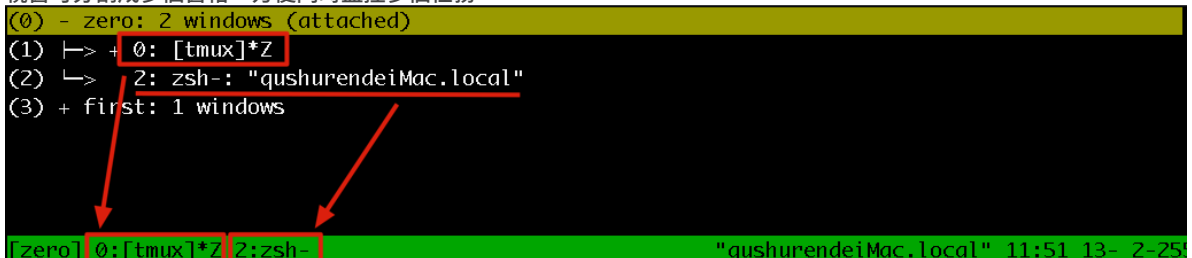
▢ tmux 的核心概念

1. **Session (工作階段) :**
 - 一個 tmux session 可以包含多個視窗，像是一個獨立的工作空間。
 - 支援在背景運行，即使關閉終端也不會中斷。

```
$tmux ls
first: 1 windows (created Thu Feb 13 11:27:20 2025)
zero: 2 windows (created Thu Feb 13 11:02:59 2025) (attached)
```

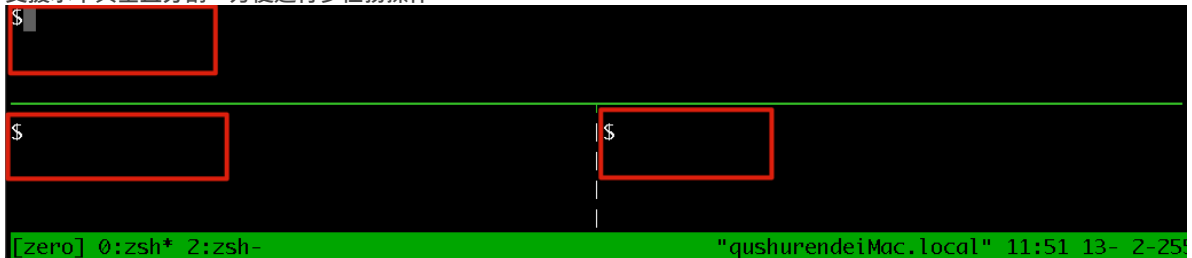
2. **Window (視窗) :**
 - 每個 session 裡可以有多个視窗，每個視窗類似一個完整的終端畫面。
 - 視窗可分割成多個窗格，方便同時監控多個任務。

```
(0) - zero: 2 windows (attached)
(1) ↳ + 0: [tmux]*Z
(2) ↳ 2: zsh-: "qushurendeiMac.local"
(3) + first: 1 windows
```



```
[zero] 0:[tmux]*Z 2:zsh- "qushurendeiMac.local" 11:51 13- 2-25
```

3. **Pane (窗格) :**
 - 視窗內的分割區域，每個窗格可獨立執行命令。
 - 支援水平與垂直分割，方便進行多任務操作。



```
[zero] 0:zsh* 2:zsh- "qushurendeiMac.local" 11:51 13- 2-25
```

安裝 tmux

Ubuntu 安裝 tmux

Ubuntu 官方倉庫提供了 tmux，可直接使用 `apt` 安裝：

```
sudo apt update
sudo apt install tmux -y
```

macOS 安裝 tmux

macOS 用戶可以透過 **Homebrew** 安裝：

```
brew install tmux
```

▢ 更新或移除 tmux

Ubuntu

- 更新：

```
sudo apt update && sudo apt upgrade tmux
```
- 移除：

```
sudo apt remove tmux -y
```

macOS

- 更新：

```
brew upgrade tmux
```
- 移除：

```
brew uninstall tmux
```

常用指令快速上手

功能	指令 / 快捷鍵	說明
session		
啟動 tmux	<code>tmux</code>	開啟一個新的 tmux session
	<code>tmux new -s 名稱</code>	開啟一個新的 tmux session 使用自訂名稱
查看所有 session	<code>tmux ls</code>	列出目前所有 tmux session
切換 session	<code>Ctrl-b s</code>	圖形化切換session (如果該session 內有多視窗，可按 -> 選擇換到該視窗)
離開 tmux (保持運行)	<code>Ctrl-b d</code>	分離 session，回到原本終端
離開 tmux (結束此session)	<code>Ctrl-b x</code>	結束目前session，回到原本終端
重新連接 session	<code>tmux attach -t 名稱</code>	重新連接到未結束的 session
	<code>tmux a</code>	重新連接上次中斷seeion
刪除 session	<code>tmux kill-session -t 名稱</code>	刪除指定名稱session
刪除所有 session	<code>tmux kill-server</code>	
重新命名 session	<code>Ctrl-b \$</code>	當前session 改名
	<code>tmux rename-session -t 舊名稱 新名稱</code>	修改指令 session 名稱
window		
建立新視窗	<code>Ctrl-b c</code>	建立一個新的視窗
切換視窗	<code>Ctrl-b w</code>	圖形化切換window
	<code>Ctrl-b 視窗編號</code>	
	<code>Ctrl-b n / p</code>	切換到下一個/上一個視窗
關閉視窗	<code>Ctrl-b &</code>	關閉當前視窗

功能	指令 / 快捷鍵	說明
重新命名 window	Ctrl-b ,	更名目前window
	tmux rename-window -t 舊名稱 新名稱	更名指定window
Pane		
分割窗格 (垂直)	Ctrl-b "	垂直分割出新窗格
分割窗格 (水平)	Ctrl-b %	水平分割出新窗格
切換窗格	Ctrl-b 方向鍵	在窗格間切換
	vi ~/.tmux.conf 加入set -g mouse on	使用滑鼠切換 使用滑鼠調整分割線
關閉窗格	Ctrl-b x	
與上一格窗格互換	Ctrl-b {	
與下一格窗格互換	Ctrl-b }	
顯示窗格編號	Ctrl-b q	
指定窗格互換	tmux swap-pane -s 1 -t 2	
其他		
命令模式	Ctrl-b :	進入命令模式，可執行 tmux 指令
配置設定檔	vi ~/.tmux.conf set -g mouse on :支持滑鼠 set -g prefix C-s :ctrl + b前改成ctrl + s	

當滑鼠模式關閉(set -g mouse off)時，作業系統提供的標準複製/貼上功能將如預期運作。
但滑鼠模式開啟(set -g mouse on)時，剪貼簿的功能會造成失效
下面的程式碼可以新增到您的 .tmux.conf 檔案中，以將m和M鍵分別綁定到「滑鼠開啟」和「滑鼠關閉」。

這讓你能夠使用Ctrl+ B,m來開啟滑鼠支援；Ctrl+ B , M將其關閉。

```
vi ~/.tmux.conf

#####

# toggle mouse mode to allow mouse copy/paste
# set mouse on with prefix m
bind m \
    set -g mouse on \;
    display 'Mouse: ON'
# set mouse off with prefix M
bind M \
    set -g mouse off \;
    display 'Mouse: OFF'
```

參考:<https://stackoverflow.com/questions/17445100/getting-back-old-copy-paste-behaviour-in-tmux-with-mouse>

常見使用情境

- **遠端連線不中斷**：即使 SSH 斷線，tmux session 仍會在伺服器上持續運行。
- **多任務同時進行**：在一個終端視窗內同時監控多個任務、Log 或服務狀態。
- **自動化腳本執行**：方便長時間執行的任務或背景作業管理。

常見問題

已在某session 內要新建session

```
$tmux
sessions should be nested with care, unset $TMUX to force
$tmux new -s aa
sessions should be nested with care, unset $TMUX to force
$
```

[1] 0:zsh* "qushurendeiMac.local" 11:08 13- 2-25

或是要切換到另一個session

```
$tmux ls
0: 1 windows (created Thu Feb 13 11:02:59 2025)
1: 1 windows (created Thu Feb 13 11:05:42 2025) (attached)
$tmux attach -t 0
sessions should be nested with care, unset $TMUX to force
$
```

[1] 0:zsh* "qushurendeiMac.local" 11:11 13- 2-25

這代表 你已經在一個 tmux session 裡，然後又試圖附加 (attach) 到另一個 session。tmux 預設不建議這種「嵌套 session」的行為，因為可能會導致控制鍵衝突或混亂。

建議方式：

建立session:離開tmux 模式 `Ctrl-b d`，在建立新的session：`tmux` or `tmux new -s 名稱`

切換session: `Ctrl-b s` 切換 or `tmux switch-client -t 名稱`

```
(0) + 0: 1 windows
(1) - 1: 1 windows (attached)
(2) ↵ 0: [tmux]*: "qushurendeiMac.local"

[1] 0:[tmux]*
```

以下是 tmux 快速鍵表格：

快速鍵	功能說明
C-b Space	選擇下一個佈局
C-b !	將窗格分離到新視窗
C-b "	垂直分割視窗
C-b #	列出所有剪貼簿
C-b \$	重新命名當前工作階段
C-b %	水平分割視窗
C-b &	關閉當前視窗

快速鍵	功能說明
C-b '	輸入視窗索引以切換
C-b (切換到上一個客戶端
C-b)	切換到下一個客戶端
C-b ,	重新命名當前視窗
C-b -	刪除最近的剪貼簿
C-b .	移動當前視窗
C-b /	顯示按鍵綁定說明
C-b 0~9	選擇對應編號的視窗
C-b :	輸入命令提示符
C-b ;	切換到上次使用的窗格
C-b =	從清單中選擇剪貼簿
C-b ?	列出所有按鍵綁定
C-b C	自訂選項
C-b D	從清單中選擇並分離客戶端
C-b E	平均分配所有窗格
C-b L	切換到最後一個客戶端
C-b M	清除標記的窗格
C-b [進入複製模式
C-b]	貼上最近的剪貼簿
C-b c	建立新視窗
C-b d	分離當前客戶端
C-b f	搜尋窗格
C-b i	顯示視窗資訊
C-b l	切換到上個視窗
C-b m	標記或取消標記窗格
C-b n	選擇下一個視窗
C-b o	選擇下一個窗格
C-b p	選擇上一個視窗
C-b q	顯示窗格編號
C-b r	重新整理當前客戶端
C-b s	從清單中選擇工作階段
C-b t	顯示時鐘
C-b w	從清單中選擇視窗
C-b x	關閉當前窗格
C-b z	放大/還原當前窗格
C-b {	與上方窗格交換位置
C-b }	與下方窗格交換位置
C-b ~	顯示訊息
C-b DC	重置視窗以跟隨游標
C-b PPage	進入複製模式並向上捲動
C-b ↑/↓/←/→	選擇相對方向的窗格
C-b M-1~M-7	設定不同的視窗佈局
C-b M-n	選擇下一個有警示的視窗
C-b M-o	反向循環切換窗格
C-b M-p	選擇上一個有警示的視窗
C-b M-↑/↓/←/→	調整窗格大小 (每次5個單位)
C-b C-b	傳送前綴鍵
C-b C-o	循環切換窗格
C-b C-z	暫停當前客戶端

快速鍵	功能說明
C-b C-↑/↓/←/→	調整窗格大小
C-b S-↑/↓/←/→	移動視窗的可見部分

Reference

- [tmux 小記](#)
- [tmux 使用和基礎配置 從入門到加班 一個視頻全搞定！](#)
- <https://hackmd.io/@Cheng-Hao/Hyk9f6mZd>
- https://blog.csdn.net/qq_41634814/article/details/119426093
- [you need to learn tmux RIGHT NOW!!](#)