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

【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) (addres > 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) (addres > 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** delpath`s`([[`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
 }
}
```

【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】【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是指在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是指在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

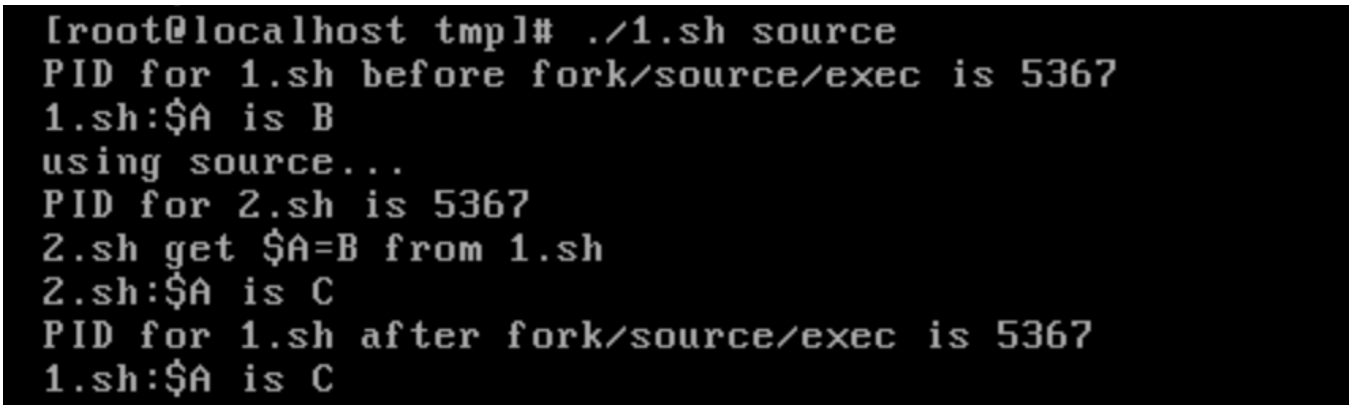
```
#!/bin/bash

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
```

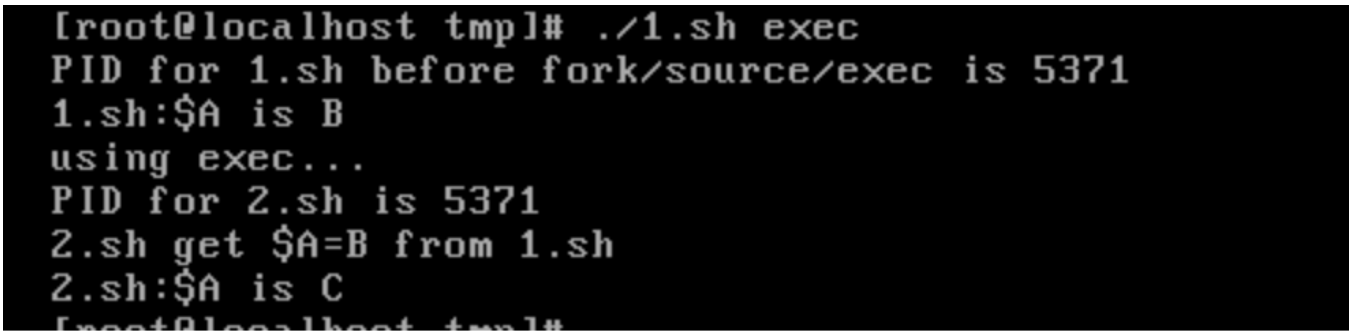
可以看到，1.sh是在父程序中執行，2.sh是在子程序中執行的，父程序的PID是5344，而子程序的是5345，當子程序執行完畢後，控制權返回到父程序。同時，在子程序改變環境變數A的值不會影響到父程序。

./1.sh source



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



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)

【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】取得pubic 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 backgroup job

【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 $s 443 &> /dev/null && echo "$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。