



# 第五章: 純文字訊息之處理

## 大綱

- 純文字訊息之處理.....2
- 重點提要.....3
- Shell 之字串型特殊替換.....4
- 正規表示式.....7
- 正規式的分類.....8
- egrep.....9
- 正規式的建構分子.....11
- 建構分子--一般字元.....12
- 建構分子--字元集合.....14
- 建構分子--任意字元.....17
- 建構分子--定位字元.....18
- 建構分子--字的邊界.....19
- 建構分子--量詞.....20,23
- 建構分子--『(....)』.....21
- sed 簡介.....25
- sed 的編輯命令--s.....26
- sed 的定址模式.....29
- sed 的編輯命令--d,p,q.....32
- 實作練習.....34

<http://edu.uuu.com.tw>



## 第五章

### 純文字訊息之處理

---

有沒有搞錯? 都什麼時代了, 還在提純文字檔? 處理純文字的訊息? 但別忘了, 人類所能接受之最原始資料型態, 還是純文字, 而不是 0 與 1 的二進位格式. 而在 Linux 下, 絕大部分的設定檔, 訊息檔, 都是純文字檔的格式. 此外, 常用的管理指令, 如 **df**, **dumpe2fs**, **uptime** 等所顯示的訊息也是純文字的格式. 若要從這些工具程式回報的訊息中, 利用 **script** 擷取相關數據, 以決定是否採取某些行動, 例如,

『當 **/var** 所在設備空間利用率已達 90% 時, 自動發送電子郵件至某處』  
此時, 文字檔的處理工具就得上場了.

## 重點提要

- shell 之字串型特殊替換
- 正規表示式
  - ▶ 正規式的分類
  - ▶ 正規式與 `egrep`
  - ▶ 正規式的建構分子
  - ▶ 正規式與 `sed`

## Shell 之字串型特殊替換

- Shell 提供字串類的特殊參數替換, 可對變數值進行基本的字串處理.  
例如:
  - ▶ 將變數值內為首的數個字元刪除
  - ▶ 刪除起始/結尾符合字樣的字串.
  - ▶ 將符合字樣的字串替換成指定字串.

- 
- 請思考如何利用上述之 shell 字串處理功能, 來找出以下所需的資料?
    - ▶ 目前的 runlevel.
    - ▶ 自『ifconfig eth0』的輸出中找出 eth0 的 IP.
    - ▶ 自『df /var』的輸出中, 找出/var 所在的空間利用率.

## Shell 之字串型特殊替換(續)

與字串處理相關之參數替換有

- `${參數:指定位置}`
- `${參數:指定位置:長度}`
- `${參數#字串} ${參數##字串}`
- `${參數%字串} ${參數%%字串}`
- `${參數/字樣/字串}`  
`${參數//字樣/字串}`

- 在撰寫 script 時, 欲對參數值進行字串處理, 如『自變數 X 值中, 取出第 6 至第 9 個字元』時, 除了可利用文字檔處理工具外, 也可考慮使用 shell 內建之特殊參數替換中, 有關字串處理的功能。
- 與字串處理有關之特殊參數替換表列如下:

參數替換	替換為
<code>\${參數:長度}</code>	將參數值從字首刪除指定長度字串後所得之結果。 <code>\${X:5}</code> 為變數 X 的值刪除前 5 個字元後所得之結果。
<code>\${參數:長度<sub>1</sub>:長度<sub>2</sub>}</code>	先將參數值從字首刪除長度 <sub>1</sub> 之字串, 再取出長度 <sub>2</sub> 的部分。 <code>\${X:5:4}</code> 為變數 X 的值先刪除前 5 個字元, 再取出 4 個字元。
<code>\${參數#字串}</code> <code>\${參數##字串}</code>	將參數值自字首刪除指定字串後的結果。 若為單一『#』, 則刪除符合字串中長度最長者; 若為『##』, 則刪除符合字串中長度最長者。 <code>\${X#/var/log}</code> 為變數 X 值自字首刪除/var/log 後之結果。
<code>\${參數%字串}</code> <code>\${參數%%字串}</code>	將參數值自字尾刪除指定字串後的結果。 若為單一『%』, 則刪除符合字串中長度最長者; 若為『%%』, 則刪除符合字串中長度最長者。

第五章: 純文字訊息之處理

參數替換	替換為
$\${參數/字樣/字串}$ $\${參數//字樣/字串}$	將參數值中符合字樣之部分以指定字串取代, 並依最長符合(longest match)的原則來決定被取代的部分。 若參數後為單一『/』, 只取代第一個符合的部分; 若為『//』, 則取代所有符合的部分

可使用字樣比對機制(如『\*』與『?』等字元)來描述欲刪除/替換之字串。

● 執行實例

若 X 值為 /var/www/html/index.html	
$\${X:3}$	r/www/html/index.html
$\${X:3:5}$	r/www
$\${X#/var}$	/www/html/index.html
$\${X#/var/www}$	/html/index.html
$\${X%.html}$	/var/www/html/index
$\${X#*/}$	var/www/html/index.html
$\${X##*/}$	index.html
$\${X%/*}$	/var/www/html
$\${X%%/*}$	
若 X 為 00-0d-63-2d-f2-e8	
$\${X/-/:}$	00:0d-63-2d-f2-e8
$\${X//-/:}$	00:0d:63:2d:f2:e8

● 實作練習

▶ 若 X 的值為『This is a test』, 由四個英文單字所構成, 其間由多個空格隔開, 如何取出為首與結尾的單字(分別為『This』與『test』)?

▶ 找出目前的 runevel.

▶ 從『ifconfig eth0』的輸出中, 找出 eth0 的 IP.

▶ 找出/var 所在設備的空間利用率.

## 正規表示式

- 正規表示式(regular expressions, 簡稱正規式)提供了比 shell 之字串型特殊替換更強大的功能.
- 欲指明多個具有共同特徵之字串時, 常可利用正規式; 如欲指明『以 a 為首, 長度為 4 的字串』時可用『`^a...$`』.
- 支援正規式的工具程式有: `more`, `less`, `egrep`, `vi`, `sed`, `awk`, `perl` 等.

---

正規為正規語言(formal language), 計算理論(computation theory)中的一環, 為資訊領域的基礎理論之一.

正規式看似神祕且高不可攀, 其實您常使用但不自知. 當您執行『`egrep 'PATH' /etc/profile`』要求 `egrep` 在 `/etc/profile` 內搜尋是否有如『`PATH`』的字樣時, 您正是透過『`PATH`』這個正規式告知 `egrep` 欲搜尋的字串.

正規式具有相當高的實用價值: 當您想在設定檔中搜尋具有某些特徵的字串, 或在訊息檔中找出某一期間所產生的訊息時, 如能利用正規式, 常能有效率地取得所需的資訊. 因此, Linux 下多種程式與均支援正規式, 如 `more`, `less` 等瀏覽工具, 編輯器 `vi`, 以及 `sed`, `perl` 等知名的 script language.

## 正規式的分類

- 正規式語法可分成基本型(basic)與延伸型(extended)兩種, 須依工具支援之型式來使用:
  - ▶ 基本型  
grep, sed, less, more, vi
  - ▶ 延伸型  
egrep, awk, sed, perl

- POSIX 將正規式之語法區分成基本型(basic regular expressions, BRE)與延伸型(extended regular expressions, ERE), 兩者功能表列如下:

功能	BRE	ERE
., ^, \$, [...], [^...]	有	有
不限次數之量詞	*	*
符合至少一次之量詞	<b>\+</b>	+
符合零至一次之量詞	<b>\?</b>	?
符合至少 n, 至多 m 次	\{n,m\}	{n,m}
標記為群組	\(...\)	(...)
量詞套用於群組	有	有
反向參考	\1 至 \9	<b>\1至\9</b>
或	<b>\ </b>	

\* 註: 有灰底者為 GNU 所提供, 並未定義在 POSIX 中.

## egrep

- **egrep** 可用以搜尋檔案中, 含有符合正規式之字串的資料行, 如  
『`egrep 'PATH' /etc/profile`』  
可用以列出 `/etc/profile` 中含有『符合 `PATH`』字串的資料行。
- **語法**  
`egrep [選項] 正規式 [檔案]`

- 『`egrep 'PATH' /etc/profile`』可用來搜尋 `/etc/profile` 中有哪些資料行含有『`PATH`』這個字串. 實際上, `egrep` 把『`PATH`』視為正規式. 因此, 另一種解釋方式為『搜尋 `/etc/profile` 中, 哪些資料行含有符合 `PATH` 之字串』.
- `grep` 以 BRE(basic regular expressions)來解譯正規式;  
`egrep` 則以 ERE(extended regular expressions)來解譯.
- 實用選項

選項	說明
<code>--color</code>	將每一行中符合正規式的部分以紅色標示
<code>-c</code>	僅計算檔案中共有多少符合正規式之資料行
<code>-i</code>	不區分大小寫
<code>-n</code>	在列出含有符合之資料行時, 同時顯示行號
<code>-l</code>	僅列出含有符合的資料行之檔案; 搜尋一檔案時, 在找到第一個符合的資料行後, 不再搜尋該檔, 轉而搜尋下個檔案
<code>-r</code>	針對目錄下之所有檔案, 不論深度, 進行搜尋.
<code>-q</code>	不列出符合之資料行, 僅利用傳用返回值來表達搜尋結果(0: 有符合之資料行)
<code>-v</code>	列出不符合之資料行
<code>-o</code>	將符合正規式的部分, 以一個一行的方式列出

● 執行實例

```
[qoo@solomon:~]$ egrep --color 'PATH' /etc/profile
    if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
        PATH=$PATH:$1
        PATH=$1:$PATH
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC
[qoo@solomon:~]$ egrep -n 'PATH' /etc/profile
7:      if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
9:          PATH=$PATH:$1
11:         PATH=$1:$PATH
40:export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC
[qoo@solomon:~]$ egrep -c 'PATH' /etc/profile
4
[qoo@solomon:~]$ egrep -l 'PATH' /etc/profile
/etc/profile
[qoo@solomon:~]$ egrep -i --color 'PATH' /etc/profile
pathmunge () {
    if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
        PATH=$PATH:$1
        PATH=$1:$PATH
# Path manipulation
    pathmunge /sbin
    pathmunge /usr/sbin
    pathmunge /usr/local/sbin
pathmunge /usr/X11R6/bin after
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC
unset pathmunge
```

● 實作練習

- ▶ 針對/etc單層目錄, 找出含有『account』的檔案.
  
- ▶ 針對/etc/X11下所有的目錄, 找出含有『user』的檔案.

## 正規式的建構分子

- 正規式的建構分子有
  - ▶ 一般字元
  - ▶ 字元集合
  - ▶ 任意字元
  - ▶ 定位字元
  - ▶ 量詞
  - ▶ 『(...)]

## 建構分子--一般字元

- 一般字元

除 `^$. *[]`, 等特定字元具有特殊意義外, 其餘之字元均為一般字元; 如大小寫字母, 數字, 底線, 空白字元等.

- 執行『`egrep 'PATH' /etc/profile`』時,
  - ▶ `egrep` 接收到之正規式為『`PATH`』, 即 `P` 之後依序接著 `A`, `T` 與 `H`.
  - ▶ 由於『`PATH`』這個正規式內不含有定位字元(『`^`』或『`$`』), `egrep` 在檢查 `/etc/profile` 中的每一行時, 會自左而右, 檢查該行是否有符合『`PATH`』者; 若有, 則將該行印出.
  - ▶ 執行實例

```
$ head -10 /etc/profile
# /etc/profile

# System wide environment and startup programs, for login setup
# Functions and aliases go in /etc/bashrc

pathmunge () {
    if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
        if [ "$2" = "after" ] ; then
            PATH=$PATH:$1
        else
$ egrep --color 'PATH' /etc/profile
    if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
        PATH=$PATH:$1
        PATH=$1:$PATH
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC
```

● 範例

▶ `redat1`的內容如下:

```
he can't hear the noises  
made by the bear  
near the pear tree in  
the rear garden  
since he is deaf  
in one ear
```

- ▶ 執行『`egrep 'bear' redat1`』時,
- 『`bear`』之意義為 `b` 之後依序接著 `e`, `a` 與 `r`.
  - 由於其中未含有定位字元(^或\$), 在檢視每一行時, `egrep` 會自左而右進行比對, 若該行含有符合『`bear`』之字串時, 則將該行印出.
- ▶ 執行實例

```
$ egrep --color 'bear' redat1  
made by the bear  
$ egrep -n 'bear' redat1  
2: made by the bear
```

## 建構分子--字元集合

- 字元集合

以 [字元列表] 表示, 用以比對一字元位置, 惟該位置必須是字元列表中之字元.

- 範例

[bp]ear 長度為四個字元, 以  
b 或 p 為首, 之後依序為  
e, a 與 r

- 字元集合(character class)的格式為『[字元列表]』, 在字元列表中標明該位置之允許值; 例如『[bp]』代表該位置只能是 b 或 p.
- 若字元列表中有某些字元構成連續性之序列時, 可利用『起首字元-終止字元』的方式來簡化寫法, 惟必須注意程式執行時所使用之語系; 例如在 POSIX(即 C) 的語系下:
  - ▶ 『[abcdef56789]』可簡寫為『[a-f5-9]』
  - ▶ 一個『大小寫字母或數字字元』可簡寫為『[A-Za-z0-9]』
- 執行實例

```
$ egrep --color 'bear' redata1
made by the bear
$ egrep --color 'pear' redata1
near the pear tree in
$ egrep --color '[bp]ear' redata1
made by the bear
near the pear tree in
```

## 建構分子--字元集合(續)

- 反相字元集合

以 `[^字元列表]` 表示, 用以比對一字元位置, 惟該位置不可以是字元列表中之字元.

- 範例

`[^bp]ear` 長度為四個字元, 不可以 `b` 或 `p` 為首, 之後依序為 `e`, `a` 與 `r`

- 反相字元集合(negated character class)的格式為『`[字元列表]`』, 在字元列表中標明該位置之不允許值, 不屬於字元列表之字元均為允許值; 例如『`[^bp]`』代表該位置只要是 `b` 或 `p` 之外的字元均可.
- 若字元列表中有某些成員構成連續性之範圍時, 可利用『起首字元-終止字元』的方式來簡化寫法, 但必須注意程式執行時所使用之語系; 例如在 POSIX(即 C)的語系下
  - ▶ 『`[^abcdef56789]`』可簡寫為『`[^a-f5-9]`』.
  - ▶ 一個『非大小寫字母及數字字元』可簡寫為『`[^A-Za-z0-9]`』.
- 執行實例

```
$ egrep --color '[^bp]ear' redata1
he can't hear the noises
near the pear tree in
the rear garden
in one ear
$ egrep -n '[a-m]ear' redata1
he can't hear the noises
made by the bear
$ egrep -n '[^a-m]ear' redata1
near the pear tree in
the rear garden
in one ear
```

### 建構分子--字元集合(續)

- **POSIX character class**  
 在字元集合中, 可用指明字元列表之成員時, 如  
 [:upper:] 代表大寫字母  
 [:lower:] 代表小寫字母  
 [:digit:] 代表數字
- **範例**  
 [ABC0-9] 相當於[ABC[:digit:]]

- POSIX 定義以下數種 character class:

表示法	說明
[:alnum:]	字母與數字, 相當於[:alpha:]與[:digit:]之聯集
[:alpha:]	大小寫字母, 相當於[:lower:]與[:upper:]之聯集
[:blank:]	空格字元, 空格與 TAB
[:cntrl:]	控制字元, 若為 ASCII 字集, 代表數值為 0 至 31 以及 127
[:digit:]	數字, 由 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 所構成
[:graph:]	相當於[:alnum:]與[:punct:]之聯集
[:lower:]	小寫字母
[:print:]	可列印之字元, 相當於[:alnum:], [:punct:] 與空白之聯集
[:punct:]	! " # \$ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` {   } ~
[:space:]	空白字元, 由空格, TAB, NL(newline), VT(vertical tab), FF(form feed), CR(carriage return)所構成
[:upper:]	大寫字母
[:xdigit:]	十六進位字元, 相當於 A-Fa-f0-9.

## 建構分子--任意字元

- 任意字元  
以『.』表示, 用以比對一字元位置, 該位置可以是任意字元.
- 範例  
.ear      四個字元, 第一個字元  
          不設限, 之後依序為 e, a  
          與 r

- 
- 執行實例

```
$ egrep --color '.ear' redata1
he can't hear the noises
made by the bear
near the pear tree in
the rear garden
in one ear
```

- 若要表示『.』.這個字元時, 必須在其前方加上反斜線, 如『\.'.

## 建構分子--定位字元

- 定位字元

『^』 『\$』 分別代表行首與行尾.

- 範例

**^.ear** 符合 .ear 且位於行首  
(第一個字元前即為行首).

**.ear\$** 符合 .ear 且位於行尾  
(r 後即為行尾).

- 若一正規式之
  - ▶ 為首字元為 『^』, **egrep** 在進行比對時會將 **RE** 之左側固定於行首.
  - ▶ 結尾字元為 『\$』, **egrep** 在進行比對時會將 **RE** 之右側固定於行尾.
  - ▶ 『^.ear』 符合長度為 4, 且位於行首之字串, 其中第一個字元不設限, 之後依序為 e, a, r.
  - ▶ 『.ear\$』 符合長度為 4, 且位於行尾之字串, 其中第一個字元不設限, 之後依序為 e, a, r.
- 執行實例

```
$ egrep --color '^.ear' redata1
near the pear tree in
$ egrep --color '.ear$' redata1
made by the bear
in one ear
```

## 建構分子--字的邊界

- 字的邊界(word boundary)
  - \b 長度為零, 該處為字的邊界
  - \B 長度為零, 該處不為字的邊界
- 範例
  - cat\b t 為該字之最後一個字元
  - \bcat c 為該字之第一個字元
  - \bcat\b 『cat』這個字

- 
- 在字的邊界(word boundary)上, 其兩側之字元, 一為構成字之字元, 另一則否.
  - 構成字的字元有大小寫字母, 數字與底線(underscore, 『\_』).
  - 範例  
redata2 的內容如下:

```
A polecat is a small dark brown wild animal
but it is not a cat.
There are several types of indicators
in that catalog.
```

請執行以下實例:

```
$ egrep --color 'cat' redata2
A polecat is a small dark brown wild animal
but it is not a cat.
There are several types of indicators
in that catalog.
$ egrep --color 'cat\b' redata2
A polecat is a small dark brown wild animal
but it is not a cat.
$ egrep --color '\bcat' redata2
but it is not a cat.
in that catalog.
$ egrep --color '\bcat\b' redata2
but it is not a cat.
```

### 建構分子--量詞

- 正規式中出現連續相同的子 RE, 可利用量詞(quantifier)來簡化, 如『aaaaa』可簡寫為『a{5}』
  - ▶ {5}代表重覆 5 次,
  - ▶ {5}與其前方長度為 1 之子 RE 結合.

- 正規式中出現連續相同的子 RE 時, 可使用量詞(quantifier)來簡化寫法. 量詞用以描述該子 RE 重覆的次數.
- 量詞『{n}』用以描述與其結合之子 RE 重覆 n 次; 尚有數種量詞容後介紹.
- 量詞與其前方之子 RE 結合, 用以代表該子 RE 重覆之次數,
- 而該子 RE 可以是(1)長度為 1 之子 RE(2)群組:

- ▶ 長度為 1 之 RE

類別	範例
一般字元	a{5}, b{3}
字元集合	[0-9]{5}, [^0-6]{3}
任意字元	.{7}

- ▶ 群組

若欲與量詞結合之子 RE 長度大於 1 時, 必須用括弧將該 RE 設定為群組 (group), 如『(ab){3}』相當於『ababab』.

- 範例--找出/etc/rc.sysinit 中有連續四個數字字元的資料行:

- ▶ `egrep --color '[0-9][0-9][0-9][0-9]' /etc/rc.sysinit` 與
- ▶ `egrep --color '[0-9]{4}' /etc/rc.sysinit`

所列的結果相同:

```
mount "/dev/mapper/$dst" "$mdir" && chmod 1777 "$mdir"
chmod 0664 /var/run/utmp /var/log/wtmp
chmod 0664 /var/run/utmpx /var/log/wtmpx
mkdir -m 1777 -p /tmp/.ICE-unix >/dev/null 2>&1
dmesg -s 131072 > /var/log/dmesg
usleep 100000
```

### 建構分子-- 『(...)』

- 正規式中, 括弧 『(...)』 的兩項功能為**群組與擷取**.
- 群組(Grouping)  
可用以將長度大於 1 之正規式與量詞結合, 如 『(ab){3}』 相當於 『ababab』

- 
- 正規式中, 括弧可用以將長度大於 1 之子 RE 設定為群組, 以與量詞結合.

## 建構分子--『(...)]』(續)

### ● 擷取(Capturing)

若在 RE 中將某部分利用『(』與『)』標記起來,則可在該 RE 之後段以『\編號』來引用該部分所擷取到之字串,例如

『([0-9][0-9])-\1』

- 若在 RE 中將某部分利用『(』與『)』標記起來,則可在該 RE 之後段引用該部分所擷取到之字串.
- 每一組括弧均有編號,編號從 1 開始,由左至右遞增.在計算編號時,必須將所有括弧列入考量,而不論該組括弧之用途(群組或擷取).
- 在 RE 中可利用『\編號』來引用前段經由『(』與『)』標記起來的部分所擷取到的字串,如『\2』為引用第 2 組.

### ● 範例

redat4 內的資料均為『年-月-日』之格式,其中年為 4 位數,月,日分別為 2 位數.請建構一正規式,列出月份與日期為同一數字之日期,如 9 月 9 日.資料檔內容如右:

- ▶ 『([0-9][0-9])-\1』這個正規式是否符合我們的需求?

```
$ egrep -n '([0-9][0-9])-\1' redat4
2:2008-09-09
4:2009-02-02
5:2010-08-08
```

```
2009-07-05
2008-09-09
2007-12-31
2009-02-02
2010-08-08
2009-07-08
```

- ▶ 『([0-9][0-9])-\1』這個正規式有何缺點?若資料檔中有『2010-10-04』,是否會被 egrep 列出來?

### ● 實作練習

找出/bin 下內,有哪些物件的檔名中有連續兩個相同字元.

### 建構分子--量詞(續)

- 用以限制符合次數的量詞:
  - ▶ \* 不限次數(零或多次)
  - ▶ + 至少一次
  - ▶ ? 零或一次
  - ▶ {n} 符合 n 次
  - ▶ {n,} 符合至少 n 次
  - ▶ {n,m} 符合至少 n, 至多 m 次

- 正規式中有多種量詞(quantifiers), 可用以限制符合的次數. 量詞之前必須有
  - ▶ 一個一般字元,
  - ▶ 字元集合(含反相字元集合),
  - ▶ 以『(』及『)』標記起來的群組,
 量詞與上述之單位結合, 限制其符合的次數.
- 常用的量詞有
  - ▶ 『\*』:不限次數(符合零或多次)
    - 如 a, aa, aaa, 連續多個 a, 或空字串均符合 『a\*』,
    - 而 ab, abab, ababab, 連續多組 ab, 或空字串均符合 『(ab)\*』.
  - ▶ 『+』:至少一次
    - 如 『a+』代表至少一個 『a』.
  - ▶ 『{n}』:符合 n 次
    - 如 aaaaa 符合 『a{5}』, 而 ababab 符合 『(ab){3}』.
  - ▶ 『{n,}』:符合至少 n 次
    - 如 aaa, aaaa, aaaaa, 等只要連續至少 3 個 a, 均符合 『a{3,}』.
  - ▶ 『{n,m}』:符合至少 n 次, 至多不超過 m 次
    - 如 aaa, aaaa, aaaaa 等符合 『a{3,5}』.
- 在進行比對時, 『\*』 『?』 『{n,}』 『{m,n}』 (即除了 『{m}』 外), 會儘可能將最長字串分配與該量詞所屬之子 RE, 此種特性稱為 greedy.
- 撰寫正規式時, 若兩個子 RE 間沒有長度與內容的限制, 可用 『.\*』 來連接這兩個子 RE.
- 實作練習
  - ▶ 搜尋檔案中有哪些行含有人名 『Solomon Chang』, 其中 『Solomon』 與 『Chang』 之間可有一或多個空格, 也可以沒有空格.

## 第五章: 純文字訊息之處理

- ▶ 承前題, 若『Solomon』與『Chang』間至少有一個空格, 正規式應如何修改?
- ▶ 搜尋/etc/rc.sysinit 中, 哪幾行含有至少兩組數字者. 每組數字由一或連續多個數字字元構成. 相鄰兩個數字字元屬於同一組數字.
- ▶ 搜尋檔案中哪幾行含有由雙引號標記的字串(double-quoted string), 如『"one"』, 『"two"』, 『"final numbers"』等.

## sed 簡介

- sed--stream editor
  - 以行為單位
  - 支援正規式
  - 提供流程控制指令
- 語法
  - sed [選項][檔案...], 如
  - sed -r -e 編輯命令...[檔案..]
  - sed -r -f 編輯命令檔 [檔案..]

- egrep 用以將符合條件的資料行印出, 若要進行修改, 必須利用 sed.
- sed 為 stream editor, 以行為單位讀取檔案內容加以編輯, 支援正規式並提供流程控制指令, 因此可視為一種小型之 script language.
- sed 執行的流程為
  - ①自檔案中讀取一行資料, 放入工作區(pattern space).
  - ②對工作區內的資料依序執行指定的編輯命令.
  - ③把工作區內的資料(即編輯後的結果)送往標準輸出.
  - ④回到①.
 『①讀取一行資料→②進行編輯→③輸出結果』稱為一個循環(cycle).
- sed 之『-r』選項可用以要求 sed 以延伸型語法來解讀正規式.
- 範例
  - 『sed -r -e '' datafile』對檔案『datafile』的每一行進行以下處理:
  - ①讀取一行資料→②進行編輯(因無編輯命令, 故內容不會被更動)→③輸出結果  
因此會將該檔案的每一行印出.
- 執行 sed 時,
  - 可在命令列利用『-e』來指定一或多個編輯命令;
  - 也可將這些編輯命令以一行一命令的方式記錄於一文字檔中, 再以『-f』指明該檔, 如『sed -r -e cmd<sub>1</sub> -e cmd<sub>2</sub> -e cmd<sub>3</sub> datafile』的另一種執行方式是將『cmd<sub>1</sub>』『cmd<sub>2</sub>』『cmd<sub>3</sub>』存成另一個檔案如『myscript』, 再執行『sed -r -f myscript datafile』.

```
cmd1
cmd2
cmd3
```

## sed 編輯命令--s

- s(substitute)
  - ▶ 語法: s/RE/替換字串/[選項]
  - ▶ 將一行中符合 RE 的部分以指定字串取代; 預設只替換第一個.
  - ▶ 選項『g』(global)可替換一行中所有符合 RE 者.
- 範例  
sed -e 's/cat/dog/g' redata3

- 編輯命令 s 可用以將一行中符合 RE 的部分替換成指定的字串, 如
  - ▶ s/cat/dog/ 把一行中第一個 cat 替換成 dog.
  - ▶ s/cat/dog/3 把一行中第三個 cat 替換成 dog.
  - ▶ s/cat/dog/g 把一行中所有的 cat 替換成 dog.
  - ▶ s/cat//g 把一行中所有的 cat 刪除.
  - ▶ s/^/Begin:/ 在行首加上『Begin:』的字串.
  - ▶ s/\$/--End/ 在行尾加上『---End』的字串.
- 實作練習  
redata3 的內容如下

```
A cat is neither a catalog nor a polecat.
```

請執行以下範例:

```
▶ sed -r -e 's/cat/dog/' redata3
▶ sed -r -e 's/cat/dog/3' redata3
▶ sed -r -e 's/cat/dog/g' redata3
▶ sed -r -e 's/cat\b/dog/g' redata3
▶ sed -r -e 's/\bcat/dog/g' redata3
▶ sed -r -e 's/\bcat\b/dog/g' redata3
▶ sed -r -e 's/cat//g' redata3
▶ sed -r -e 's/^/Hi.../' redata3
```

### sed 編輯命令--s(續)

- RE 中可利用前述之群組機制, 並在替換字串中, 利用『\群組編號』來引用.

- 
- 若要將 `redata4` 內的日期格式由『年-月-日』改成『月-日-年』時, 可利用『`s/^[0-9]{4}-([0-9]{2}-[0-9]{2})/\2-\1/`』這個編輯命令:
    - ▶ 在 RE 中, 利用兩組『(』與『)』把年, 以及月-日分別標記起來; 欲引用時, 『\1』為年, 『\2』為月-日.
    - ▶ 而替換字串為『\2-\1』即替換成『月-日-年』.
    - ▶ 執行實例:

```
$ cat redata4
2009-07-05
2008-09-09
2007-12-31
2009-02-02
2010-08-08
2009-07-08
$ sed -r -e 's/^[0-9]{4}-([0-9]{2}-[0-9]{2})/\2-\1/' redata4
07-05-2009
09-09-2008
12-31-2007
02-02-2009
08-08-2010
07-08-2009
```

### sed 編輯命令--s(續)

- 在編輯命令 s 的『替換字串』中,, 可用『&』代表搜尋到(即符合 RE) 的字串; 如

『s/[0-9]+/<&>/g』可用以將每行中連續多個數字字元用一組『<』及『>』框起來。

---

- 範例

把『one 123 two 13579 three 25149191 end』中『連續多個數字字元』用一組『<』及『>』框起來:

```
$ echo 'one 123 two 13579 three 25149191 end' |  
sed -r -e 's/[0-9]+/<&>/g'  
one <123> two <13579> three <25149191> end
```

## sed 的定址模式

- 預設編輯命令會執行於每一行。
- 編輯命令前可搭配定址模式，以限制該命令僅執行於指定的資料行上。
- 定址模式有單位址『位址』與雙位址『位址 1, 位址 2』
- 位址可為行號或『/正規式/』，如『3』 『5,8』 『/dog/』 『5,/dog/』。

- 預設編輯命令會執行於每一行。  
若欲限制僅執行於指定的資料行時，可於編輯命令前搭配定址模式來達成。
- 定址模式有
  - ▶ 零位址，即針對所有資料行，
  - ▶ 單位址--『位址』，
  - ▶ 雙位址--『位址 1, 位址 2』，
 其中位址可以為
  - ▶ 行號，如『3』代表第 3 行，『\$』代表檔案的最後一行。
  - ▶ 『/正規式/』。
- 範例
  - ▶ 3                    第 3 行。
  - ▶ 5,8                第 5 至 8 行。
  - ▶ /dog/              含有『dog』的資料行。
  - ▶ 5,/dog/            自第 5 行起至其後首先含有『dog』的資料行為止。
  - ▶ /^#/               以『#』為首之資料行。
  - ▶ /cat/,/dog/        自含有『cat』之資料行起，至其後首先含有『dog』之資料行為止。
  - ▶ /[bp]ear/          含有符合『[bp]ear』字串之資料行。

### sed 的定址模式(續)

- 整組位址後若加上『!』, 代表否定, 亦即不在該位址範圍內之資料行才進行編輯, 如
  - ▶ 3! 非第 3 行
  - ▶ /dog/! 不含有『dog』的資料行
  - ▶ 5,10! 不屬於第 5 至 10 行
  - ▶ 5,/dog/! 第 5 行以前或其後首先含有『dog』之行之後

- 執行實例

```
$ cat redata5
L1:A polecat is a small dark brown wild animal
L2:but it is not a cat.
L3:Besides, cats do not like dogs at all.
L4:There are several types of dogs
L5:in that catalog.
$ sed -r -e 's/cat/<&>/g' redata5
L1:A pole<cat> is a small dark brown wild animal
L2:but it is not a <cat>.
L3:Besides, <cat>s do not like dogs at all.
L4:There are several types of dogs
L5:in that <cat>alog.
$ sed -r -e '2,5s/cat/dog/g' redata5
L1:A polecat is a small dark brown wild animal
L2:but it is not a <cat>.
L3:Besides, <cat>s do not like dogs at all.
L4:There are several types of dogs
L5:in that <cat>alog.
```

## ● 執行實例(續)

```
$ sed -r -e '2,4s/:.*:/:' redata5
L1:A polecat is a small dark brown wild animal
L2:
L3:
L4:
L5:in that catalog.
$ sed -r -e '2,4!s/:.*:/:' redata5
L1:
L2:but it is not a cat.
L3:Besides, cats do not like dogs at all.
L4:There are several types of dogs
L5:
$ sed -r -e '/dog/s/cat/<&>/g' redata5
L1:A polecat is a small dark brown wild animal
L2:but it is not a cat.
L3:Besides, <cat>s do not like dogs at all.
L4:There are several types of dogs
L5:in that catalog.
$ sed -r -e '2,/dog/s/cat/<&>/' redata5
L1:A polecat is a small dark brown wild animal
L2:but it is not a <cat>.
L3:Besides, <cat>s do not like dogs at all.
L4:There are several types of dogs
L5:in that catalog.
$ sed -r -e '/dog/,/cat/s/:/---/' redata5
L1:A polecat is a small dark brown wild animal
L2:but it is not a cat.
L3---Besides, cats do not like dogs at all.
L4---There are several types of dogs
L5---in that catalog.
```

## sed 編輯命令--d, p, 與 q

- q(quit)  
在該循環(cycle)完成後結束 sed.
- d(delete)  
可用以將工作區內的資料刪除, 開始另一循環.
- p(print)  
可用以將工作區內的資料送往標準輸出.

- 『sed -r -e ' datafile』對檔案『datafile』的每一行進行以下處理:  
①讀取一行資料→②進行編輯(因無編輯命令, 故內容不會被更動)→③輸出結果  
因此會將該檔案的每一行印出.
- 編輯命令『q』可用以要求 sed 在此循環結束後結束執行. 如  
『sed -r -e '5q' datafile』可用以印出 datafile 之第 1 至 5 行後結束:  
▶ 第 1 至 4 行之循環: ①讀取資料, ②進行編輯(無編輯命令), ③輸出結果  
▶ 第 5 行之循環: ①讀取資料→②進行編輯(『q』: 此循環完成後結束 sed)→③輸出結果, 因此在③之後就結束, 不會再讀入其後之資料.
- 編輯命令『d』可用以將工作區內的資料刪除, 該循環結束, 再開始另一循環.  
以『sed -r -e 'd' datafile』為例,  
sed 對 datafile 的每一行之處理過程為:  
①讀取一資料行→②進行編輯(在此為『d』--將該行刪除, 不會有③之動作而直接回到①, 開始另一循環). 因此本例將不會印出任何資料行.
- 編輯命令『d』配合定址模式使用時, 可用以要求 sed 刪除某些資料行, 藉以印出欲取得之資料行. 如針對 datafile,  
▶ 欲印出第 5 行  
可要求 sed 刪除第 1 至 4, 及第 6 至最後一行以達成:  
sed -r -e '1,4d' -e '6,\$d' datafile  
執行時, sed 在②對  
■ 檔案的第 1 至 4, 及第 6 至最後一行執行『d』後結束本循環, 回到①開始下一循環.  
■ 對其餘之行(即第 5 行)無命令可執行, 接著進入③而將該行印出.
- ▶ 欲印出第 5, 及第 8 行  
可要求 sed 刪除第 1 至 4, 6 至 7, 及第 9 至最後一行以達成:  
sed -r -e '1,4d' -e '6,7d' -e '9,\$d' datafile  
▶ 印出第 5, 8, 及至第 15 行

可要求 sed 刪除第 1 至 4, 6 至 7, 9 至 14, 及第 16 至最後一行以達成:

```
sed -r -e '1,4d' -e '6,7d' -e '9,14d' -e '16,$d' datafile
```

- 在前例中, 在印出所需資料之最末行後, 仍對其後至檔尾之每一行執行『**①**讀取**→****②**執行「d」以刪除』, 而這些動作可藉由對所需資料之最末行執行『**q**』來取代: 如
  - ▶ 欲印出第 5 行
 

```
sed -r -e '1,4d' -e '5q' datafile
```
  - ▶ 欲印出第 5, 及第 8 行
 

```
sed -r -e '1,4d' -e '6,7d' -e '8q' datafile
```
  - ▶ 印出第 5, 8, 及至第 15 行
 

```
sed -r -e '1,4d' -e '6,7d' -e '9,14d' -e '15q' datafile
```
- 刪除含有『dog』之資料行
  - ▶ 

```
sed -r -e '/dog/d' datafile
```
- 只印出含有『dog』之資料行
  - ▶ 

```
sed -r -e '/dog/!d' datafile
```
- 編輯命令『**p**』可用以將工作區內的資料送往標準輸出。以『`sed -r -e 'p' datafile`』為例, 每一行的循環為**①**讀取資料**→****②**進行編輯(『**p**』--送往標準輸出)**→****③**輸出結果因此每一行會印出兩次。
- sed 預設於一循環的結束時, 會將工作區內的資料送往標準輸出(即**③**), 此功能可利用選項『**-n**』加以關閉。
- 執行 sed 搭配選項『**-n**』時, 亦可利用編輯命令『**p**』印出指定之資料行, 如:
  - ▶ 欲印出第 5 行
 

```
sed -r -n -e '5p' -e '5q' datafile
```
  - ▶ 欲印出第 5, 及第 8 行
 

```
sed -r -n -e '5p' -e '8p' -e '8q' datafile
```
  - ▶ 印出第 5, 8, 及至第 15 行
 

```
sed -r -n -e '5p' -e '8p' -e '15q' -e '15q' datafile
```
  - ▶ 刪除含有『dog』之資料行
 

```
sed -r -n -e '/dog/!p' datafile
```
  - ▶ 只印出含有『dog』之資料行
 

```
sed -r -n -e '/dog/p' datafile
```

## 實作練習

- 目標
  - ▶ 利用 `egrep` 練習建構正規式
  - ▶ 利用 `sed` 處理文字檔

### 1. 利用 `egrep` 搜尋 `/etc/profile`, 列出

#### 【基本練習】

- (1) 含有『`PATH`』字串之資料行, 又一共有幾行?
- (2) 含有『`PATH`』字串(不區分大小寫)之資料行.
- (3) 含有『`.`』之資料行.
- (4) 『`#`』為首之資料行.
- (5) 以非字母字元為首之資料行.
- (6) 以數字結尾之資料行.
- (7) 空行有多少行.
- (8) 不以『`#`』為首之資料行, 不含空行.
- (9) 不以『`#`』為首之資料行, 包含空行.

#### 【量詞】

- (10) 倒數第 6 個字元為『`e`』之資料行.
- (11) 長度為 15 之資料行.
- (12) 長度小於 15 之資料行.
- (13) 長度大於 15 之資料行.
- (14) 長度介於 15 與 20 間(包含邊界值)之資料行.

#### 【`.`的應用】

- (15) 以『`#`』為首, 且長度至少 20 之資料行.
- (16) 倒數第 6 個字元為『`e`』, 且以『`#`』為首之資料行.
- (17) 沒有出現數字字元的資料行.
- (18) 沒有出現數字字元, 亦非空行之資料行.
- (19) 含有字串『`PATH`』(不區分大小寫), 同時不以『`#`』為首之資料行.
- (20) 含有至少兩組數字的資料行. 其中每組數字由一或連續多個數字字元所構成; 兩個相鄰的數字字屬於同一組數字.
- (21) 出現至少兩個『`PATH`』字串的資料行.

(22) 至少有三個『/』字元之資料行.

**【字的邊界】**

(23) 含有『PATH』這個英文單字之資料行, 不區分大小寫.

(24) 含有長度為 7 之『字』(word, 一個 word 由大小寫字母, 數字與底線所構成)的資料行.

(25) 含有長度為 8 的『字』, 且該字全由大寫字母構成的資料行.

(26) 含有『a』的『字』(word).

**【進階練習】**

(27) 正好有三個『/』字元之資料行.

(28) 只出現偶數個『a』的資料行.

(29) 只出現奇數個『a』的資料行.

2. 確認系統中是否有 qoo 這個使用者帳號.

3. [sed]

(1) 利用 sed 讀取/etc/profile, 輸出時將所有『path』, 不分大小寫, 全替換成『PATH』.

(2) 利用 sed 讀取/etc/profile, 輸出時將所有長度為 7 的『字』之前後各加上『\_』.