

Shell & shell script

USING csh/tcsh

Part 1

Shell

:: Outline ::

- Shell是什麼
- 爲什麼要學Shell
- 各式各樣的shell
- 如何選擇自己的shell
- Csh及tcsh不同的地方
- C shell的基本運用

Shell是什麼

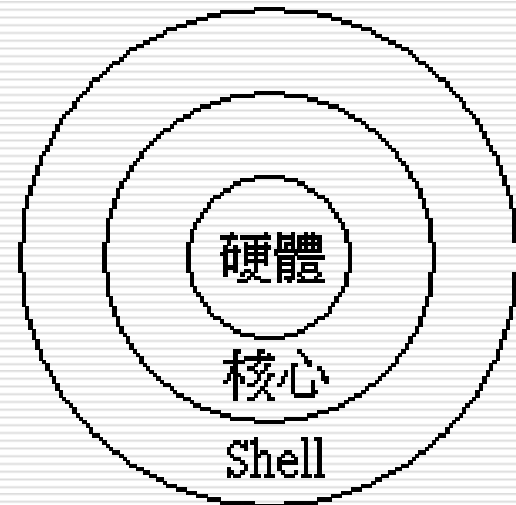
☐ 貝殼??....NO

☐ 溝通的橋樑??...YES

-- getty --> login --> shell --> logout --

^

|-----|



爲什麼要學Shell

- ❑強大的組合命令
 - ❑快速且方便的文字介面
 - ❑簡單好學的好工具
 - ❑分析系統記錄檔的利器
-

各式各樣的shell

- sh (已經被 /bin/bash 所取代)
 - bash--- (bourne-again shell Linux 預設的 shell)
 - ksh (Kornshell 由 AT&T Bell lab. 發展出來的，相容於 bash)
 - zsh (基於 ksh 發展出來的，功能更強大的 shell)
 - tcsh (整合 C Shell，提供更多的功能)
 - csh (就是FreeBSD預設的shell是由 Bill Joy為 BSD UNIX 系統所創造)
 - 如何看自己目前的shell? `echo $shell`
 - **`% cat /etc/shells`**
-

/etc/passwd

- ❑ 修改及觀看帳號的登入時預設shell
- ❑ 除非必要,不要修改root的預設shell
 - 開啓一個新的subshell

```
% less /etc/passwd
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1:/:
nobody:x:60001:60001:Nobody:/:
ftp:x:1000:1000:ftp:/:/usr/sbin/nologin
```

如何選擇自己的shell

- 大多數的人是懶惰的...XD
 - FreeBSD default shell→csh/tcsh
 - Linux default shell→sh/bash
 - 套用在學習哪種unix-like OS也是一樣低
-

Which shell will be introduced??

csch/tcsch

Differences between csh and tcsh

- ❑ csh = shell command interpreter with a C-like syntax
 - ❑ tcsh = C shell with file name completion and command line editing
 - ❑ DOSKEY (command line editing)
 - 由上下左右四個方向鍵,觀看history list中之命令並可加以修改或執行。
 - ❑ 完成命令或檔名 (file name completion)
 - Tab鍵
 - 不需要輸入完整的檔名即可自動補齊檔名
 - ❑ 命令或檔案之搜尋
 - Ctrl+d
 - 列出可能的檔案名稱
 - ❑ 計中sparc工作站
 - csh嘗試平台
 - ssh sparcID@sparc20.cc.ncu.edu.tw
-

設定Shell

- 不同的shell有不同的設定方式

Shell種類	sh/bash	ksh/zsh	csh/tcsh
設定檔名稱	“.profile” “.login”	“.profile” “.login”	“.cshrc” “.login” “.logout”

/etc/csh.cshrc

- /etc/csh.cshrc -> /etc/csh.login ->
~/.cshrc -> ~/.login

```
% more /etc/csh.cshrc
# $FreeBSD: src/etc/csh.cshrc,v 1.3 1999/08/27 23:23:40 peter Exp $
#
# System-wide .cshrc file for csh(1).
set prompt = "%B%m [%~]-%n->" #改變提示字元格式
set no beep #不準叫
```

基礎知識—路徑

□ 絕對路徑

- 由 / 寫起

□ 相對路徑

- 絕對不是由 / 寫起
- . #目前目錄
- .. #上一層目錄
- ~ #家目錄

□ 寫shell最好用絕對路徑

□ 如何查絕對路徑

- **which** command
-

Csh/tcsh 的基本運用

- 指令運用
 - 輸入 / 輸出重導向
 - 管線的觀念與運用
 - 萬用字元的運用
 - History 的設定與運用
 - 別名的設定與運用
 - 工作控制的運用
 - 數值運算
-

指令運用(1)

□ 單一指令

- ls
- ps -aux
- netstat -an

□ 連續指令

- mkdir ~/backup ; cp ~/.* ~/backup/; logout

指令運用(2)

□ 條件式指令

- `grep google sampleCom.txt && more sampleCom.txt`
- `grep google sampleCom.txt || echo "no"`
- `cc prog.c || echo 'compile error'`
- `&&` -> then
- `||` -> else

□ 指令類別

- 內建命令(built-in command)
 - 程式(utility)
 - 如何辨別差別 %**which command**
-

指令運用(2)－範例

```
% fetch http://tinyurl.com/erug8
```

```
% grep google erug8 && who
```

```
% grep test erug8 || echo "no"
```


指令運用(3)

□ 指令列的格式

- | | | |
|-----------|---------|---------------|
| ■ command | options | arguments |
| ■ ls | -al | ./ |
| ■ rm | -i | test.txt |
| ■ echo | | "hello world" |

□ 遇到問題

- 可靠的男人?? **man** command
 - info
 - option 加入 -h or --help
-

輸入 / 輸出重導向

□ 標準輸出入符號

- < Input Redirection
- << keyword input redirection
- > Output Redirection
- >> Appending Output Redirection

```
% ls ./ > result.txt #資料夾中的子資料夾存到文字檔
% echo "appending text" >> result.txt
% mail xxx@hotmail.com < result.txt
% mail xxx@hotmail.com << End_Of_File
> this is a test mail
> this is a test mail
> this is a test mail
> End_Of_File
```

管線的觀念與運用

□ command1 | command2

```
%ls | wc -w #計算字數
```

```
%cat /etc/passwd
```

```
%cat /etc/passwd | sort #將/etc/passwd檔作排序
```

```
%history | less #看history中的歷史指令
```

萬用字元的運用

□ *

■ 可以代替任何字元

□ ?

■ 可以代替一個字元

□ [...]

■ 可以代替中括號內的任一個字元

萬用字元的運用—範例

```
% touch a.c; touch b.c;touch cc.c;mkdir dir1;mkdir dir2
% ls *.c
a.c b.c cc.c
% ls ?.c
a.c b.c
% ls [ac]*
a.c b.c
% ls [A-Za-z]*
```

History 的設定與運用

- `set $history = n`
 - 設定有南
 - `history`
 - 觀看歷史指令
 - `!!`
 - 執行上一個指令
 - `!n`
 - 執行第 `n` 個指令
 - `!string`
 - 搜尋以某字串 `string` 為開頭的過去指令並加以執行
 - 搜尋時以由 `event` 數字大到小
 - `!?string?`
 - 搜尋過去指令行中有某字串 `?string?` 並加以執行
-

History 的設定與運用—範例

```
% history
1 8:00      pwd
2 8:01      ls
3 8:02      date
4 8:04      grep google erug8 && who
5 8:06      rm test
6 8:07      history
% !3        #執行第3個指令
% !!        #執行上個指令
% !grep     #搜尋符合字串並執行
```

別名的設定與運用

- ❑ 設定 `alias name 'command'`
 - ❑ 解除 `unalias name`
 - ❑ 顯示全部設定 `alias`
 - ❑ 顯示某個設定 `alias name`
 - ❑ 可在一開始就設定在 `.cshrc`
-

別名的設定與運用

```
% touch test
```

```
% rm test
```

```
% alias rm 'rm -i'
```

```
% touch test
```

```
% rm test
```

```
remove test?
```

```
% who
```

```
% date
```

```
% alias ww 'date;who'
```

```
% ww
```

```
Thu Oct 5 08:40:53 UTC 2006
```

```
callmelei      tty6    Oct 5 01:42 (v254-44.NTCU.net)
```

工作控制的運用

- 什麼是工作控制
 - 前景工作(foreground jobs)
 - 需等提示符號出現才能進行下一個工作
 - 背景工作(background jobs)
 - 在一般的指令後面加個 &
 - 常用的指令
 - Ctrl+c 中斷指令
 - Ctrl+z 暫停指令
-

工作控制的運用—範例

```
% vi test &
```

```
% jobs
```

```
[1]+  Stopped                  vi test
```

```
% fg
```

```
% vi test
```

```
Ctrl+z
```

```
% jobs
```

```
[1]+  Stopped                  vi test
```

```
% fg
```

內建的指令

- ☐ Echo
 - ☐ pushd popd
 - ☐ dirs
 - ☐ repeat
 - ☐ exec
 - ☐ exit
 - ☐ rehash unhash hashstat
-

內建的指令 -- echo

☐ echo

- 顯示文字訊息

☐ echo -n

- 將你的游標停在 echo 所顯示出的訊息之後

☐ echo \$shell

☐ echo "hello world"

內建的指令 – pushd popd

- ❑ pushd directory
 - 將目錄放進stack中，並移到目標目錄中
- ❑ popd directory
 - 將目錄拿出stack中，並移到目標目錄中
- ❑ Stack ??
 - 推疊
 - 先進後出(first in first out)

```
% pushd /etc
/etc ~
% pushd /usr/ports
/usr/ports /etc ~
% popd
```

內建的指令--dirs

- ❑ dirs
- ❑ 此內建指令功能與 pwd 指令相似均是顯示目前的工作目錄
- ❑ dirs 所顯示出來的資料其實是目錄堆疊 (directory stack)

```
% pushd /etc      #指令
```

```
/etc ~  #結果
```

```
% pushd /usr/ports
```

```
/usr/ports /etc ~
```

```
% dirs
```

```
/usr/ports /etc ~
```

內建的指令--repeat

- ❑ repeat 執行次數 command
- ❑ 懶人用的好工具
- ❑ 重開機時可用 repeat 3 sync

```
% repeat 3 echo "hello world"    #指令
hello world
hello world
hello world    #結果
% repeat 3 sync    #確保記憶體和硬碟中的資料有無同步
```


內建的指令--exec

- ❑ exec command
 - ❑ 不會產生subshell,因此執行完後會直接結束程序
 - ❑ 常用在登出系統時備份檔案或是清理舊檔案
-

內建的指令--exit

- ❑ exit (n)
 - ❑ 結束程序
 - ❑ echo \$? or echo \$status
 - ❑ 可放在shell script中 當作結束
-

內建的指令--rehash、unhash、hashstat

□ rehash

- 當path變數中的指令有所更新時，要執行此指令才能找到新的指令

□ unhash

- 取消 hash功能，直接到path目錄中去比對，但會降低搜尋指令的速度

□ hashstat

- 5 hits, 1 misses, 83%
 - 第一個參數是指有效執行幾次指令
 - 第二個參數是指執行幾次無效指令
-

引號的運用

- 單引號(single quote)(')—兩個單引號中的內容是不變的
 - 雙引號(double quote)(")—會自動替換雙引號中的特殊字元
 - 變數
 - 倒單引號(back quote)(`)—會被先執行
 - 如果將倒單引號放在單引號中，會沒有作用
-

引號的運用—範例

```
% set color=red    #設定變數
% echo "$color"    #印出指定變數
red                #結果
% echo '$color'    #示範以單括號印出
$color            #結果
% echo "There are `who | wc -l` people online" #示範印出有倒單引號
There are 1 people online #結果
```

習題

- 當我離開 `tcsh` 後，希望系統可以幫我將最近工作的：1.) 工作日期； 2.) 100 個歷史命令獨立記錄到 `~/.tcsh_local` 檔案中，該如何設定？

編輯 `~/.logout`，將這個檔案內容變成：

```
# ~/.logout
```

```
date >> ~/.tcsh_local
```

```
history 100 >> ~/.tcsh_local
```

```
clear
```

習題

- 如何秀出 `/bin` 底下，檔名為四個字元的檔案？
 - `ls /bin/????`
 - 如何秀出在 `/bin` 底下任何以 `a` 為開頭的檔案檔名的詳細資料？
 - `ls -l /bin/a*`
 - 如何將 `last` 的結果中，印出你曾經登入過的資料
 - `last | grep pcbsd`
-

Part 2

Shell script

::Outline::

- Shell script是什麼
- 爲什麼要學Shell script
- 第一次shell script

在開始寫shell script之前??

JOKE

□ 在一個資訊展覽會(COMDEX)上，微軟公司的創辦人比爾蓋茲語重心長的說：「如果通用汽車(美國最大的汽車製造商)能使他們的科技進展速度如同資訊產業的進展速度，那麼我們今天的汽車售價僅需廿五美元，每一加侖汽油可以跑一千英哩。」全場來賓皆覺深有同感並給予他熱烈的掌聲。

爲了回應比爾蓋茲的評論，通用汽車當時的總裁傑克威治特別發佈一篇新聞稿，內容重點爲：如果通用汽車發展科技的方式如同微軟一樣，那麼我們今天的汽車就有以下特色：

1. 汽車每天會毫無理由的撞毀(Crash)兩次。
2. 每一次道路標線重漆或是交通標識改變時，你就必須買一輛新車。
3. 有時候你的汽車會毫無理由在高速公路上停下來，你只好接受事實，然後重新發動，重新上路。
4. 有時候，你操控汽車右轉、左轉或倒車時，可能會造成汽車熄火，而且拒絕再度發動，唯一解決方法是重新安裝引擎。
5. 每一輛車只能有一名乘客，除非你買的是 汽車95/98 或 汽車NT/2000。即使如此，你還得加座椅，座椅的價格和汽車差不多。

6. 麥金塔(Macintosh，著名電腦製造商)製造一種非常耐用的汽車，由太陽能 (Sun，著名系統軟體供應商)推動，速度快五倍，使用方式易學易熟練，但它只使用百分之五的道路。

7. 原先儀表板上所有的指示燈，例如：油量、水量、溫度等，現在濃縮成一個，上面標示爲 一般汽車錯誤。

8. 新的座椅要求每個人的臀部要一樣大。

9. 安全氣囊彈出之前會問：「你確定嗎？」

10. 有時候你的汽車會毫無理由將你鎖在門外，唯一的進入方式是同一時間右手拉著車門開關、左手轉動車鑰匙、口內咬著汽車天線。

11. 不管車主願不願意，通用汽車會要求車主還必須購買由通用汽車子公司出版的導航地圖。如果車主不願意使用並將它丟掉，汽車的效能會銳減百分之五十，如果車主使用別家出版的導航地圖，汽車撞毀(Crash)的次數會增加。

12. 每一次通用汽車推出新車型時，車主必須重新學習駕駛方式，因爲新車和舊車 的所有控制方式都不相同。

13. 你必須按啓動才能關閉引擎。

什麼是 shell script

- script == 腳本 劇本...貝殼腳本
 - Like dos底下的批次檔
 - 很多指令集合而成的可執行文字檔
-

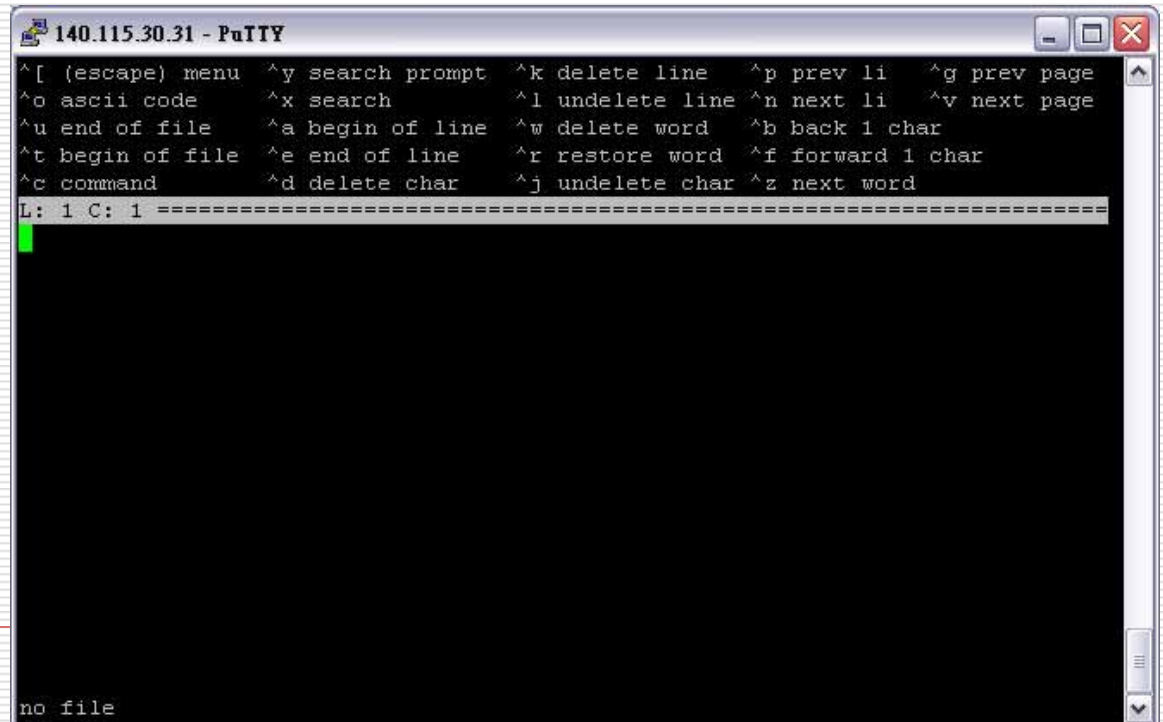
爲什麼要學shell script

- 自動化管理
 - 連續指令單一化
 - 用檔案把一堆指令包起來
 - 跨平台且學習歷程短
 - VS C
 - VS C++
-

EDITOR

□ ee

- 跟記事本很像，入門門檻比vi低
- 離開指令: `esc->a->a`



The screenshot shows a PuTTY window titled "140.115.30.31 - PuTTY". The editor interface displays a list of keyboard shortcuts at the top, organized in three columns. Below the shortcuts, the status bar shows "L: 1 C: 1" followed by a long line of equals signs. A green cursor is visible at the start of the first line. The bottom of the window displays the text "no file".

140.115.30.31 - PuTTY		
^[(escape) menu	^y search prompt	^k delete line
^o ascii code	^x search	^l undelete line
^u end of file	^a begin of line	^w delete word
^t begin of file	^e end of line	^r restore word
^c command	^d delete char	^j undelete char

^p prev li ^g prev page
^n next li ^v next page
^b back 1 char
^f forward 1 char
^z next word

L: 1 C: 1 =====

no file

第一支shell程式—hello world

- › mkdir cctraining; cd cctraining
- › ee first.csh

```
#!/bin/csh
#program:
#  this is show "hello world"
#date:2006/10/1
echo "Hello World!!"
```

離開ee : esc -> a -> a

- › csh first.csh
- › fetch <http://tinyurl.com/k2htr>
- › csh k2htr

解說

□ 用什麼來寫shell script

- vi(m)
- ee
- 任何文字編輯軟體皆可

□ 註解

- #
- 在'#'後面的文字將不會被解譯

□ 如何執行寫好的shell script

- `chmod +x filename.csh; ./filename.csh`
 - `csh filename.csh`
-

變數

□ 設值

- `set variable_name = value`

- 變數不能以數字做為開頭

□ 設定array variable

- `set name = (value1 value2 value3 value4)`

□ 使用者自行讀入變數內容

- `set name = $<`

□ 數值運算

- `@ x = $x + 1`

- `@ x = $x * 2`

變數—範例

```
% set color = red
```

```
% echo $color
```

```
red
```

```
% set color = ( red yellow blue )
```

```
% echo $color
```

```
red yellow blue
```

```
% set color = $<
```

```
orange
```

```
% echo $color
```

```
orange
```

變數—範例(2)

```
% set x = 1
```

```
% x = $x + 1
```

```
x: Command not found.
```

```
% @ x = $x + 1
```

```
% echo $x
```

```
2
```

第二個shell script

□ 利用csh 來輸出從1到100

```
% ee second.csh
#!/bin/csh
set x = 1
while($x < 100)
    echo $x
    @ x = $x + 1
end
```

- 離開ee編輯器請按esc -> a -> a
-

解說

□ 數值運算

■ $@ x = \$x + 1$

□ 設值時不用\$字號，取值時才需加\$字號

控制流程(1)

- foreach...end

```
$ ee foreach.csh
#!/bin/csh
foreach color (red yellow green)
    echo "one color is $color"
end
%csh foreach.csh
one color is red
one color is yellow
one color is blue
```

控制流程(2)

- Goto
- fetch <http://tinyurl.com/emf66>

```
$ ee goto.csh
```

```
#!/bin/csh
```

```
echo "firstLine"
```

```
goto second
```

```
echo "thirdLine"
```

```
second:
```

```
echo "secondLine"
```

```
% csh goto.csh
```

```
firstLine
```

```
secondLine
```

控制流程(3)

- if ...then ...else...endif
- **fetch <http://tinyurl.com/zh6mw>**

```
$ ee ifelse.csh
#!/bin/csh
echo "enter a number"
set number = $<
if ($number < 0) then
    echo "negative"
else if ($number == 0) then
    echo "zero"
else
    echo "positive"
endif
```

控制流程(4)

- switch...case...endsw
- fetch **<http://tinyurl.com/kzmbo>**

```
$ ee switch.csh
#!/bin/csh
echo -n "enter a number(1 for date,2 for who,3 for pwd, other for illegal
choise):"
set number = $<
switch($number)
  case 1:
    date
    breaksw
  case 2:
    who
    breaksw
  case 3:
    pwd
    breaksw
  default:
    echo "illegal choice"
    breaksw
endsw
```


控制流程(5)

- while...end
- fetch **<http://tinyurl.com/fwadp>**
- csh fwadp 9

```
% ee while.csh  
  
#!/bin/csh  
  
set x = 1  
while ($x <= $1)  
  set y = 1  
  while ($y <= $1)  
    @ v = $x * $y  
    echo -n "$v    "  
    @ y ++  
  end  
  echo ""  
  @ x ++  
end
```

進階條件指令

- -d \$var
 - 如果變數是目錄傳回true
 - -e \$var
 - 如果存在變數檔案傳回true
 - -f \$var
 - 如果存在變數為檔案傳回true (I.e., not a directory)
 - -o \$var
 - 如果檔案是由執行shell的使用者建立的傳回true
 - -r \$var
 - 如果檔案可由執行shell的使用者讀取傳回true
 - -w \$var
 - 如果檔案可由執行shell的使用者寫入傳回true
 - -x \$var
 - 如果檔案可由執行shell的使用者執行傳回true
 - -z \$var
 - 如果檔案是空檔案傳回true
-

Debug

- `$? , $status` – 觀看上個指令執行的結束值
 - `csh -x name.csh --Debug Option`
-

Debug -- example

```
% pwd
```

```
/usr/local
```

```
% echo $?
```

```
0
```

```
% pwd -h
```

```
pwd: illegal option -- h
```

```
usage: pwd [-L | -P]
```

```
% echo $?
```

```
1
```

Debug -- example

```
$ vim debug.csh
```

```
#!/bin/csh
```

```
set color = (red blue green)
```

```
set one = 1
```

```
set two = 2
```

```
echo "debug test start"
```

```
echo $color
```

```
echo $one
```

```
echo $two
```

```
% csh -x debug.csh
```

```
set color = ( red blue green )
```

```
set one = 1
```

```
set two = 2
```

```
echo "debug test start"
```

```
debug test start
```

```
echo red blue green
```

```
red blue green
```

```
echo 1
```

```
1
```

```
echo 2
```

```
2
```

習題

- 讓使用者輸入一個數字，程式可以由 $1 + 2 + 3 \dots$ 一直累加到使用者輸入的數字為止。
 - **fetch** <http://tinyurl.com/edzqo>
 - **csh edzqo**
 - 用shell script來判斷是否有/etc/passwd 這個檔案，如果有的話則把有你帳號的那行字串存入 ~/test.txt，如果沒有則輸出no data
 - **fetch** <http://tinyurl.com/pftz8>
 - **csh pftz8**
-

參考資料

- <http://netlab.cse.yzu.edu.tw/~statue/freebsd/docs/completed/84.shtml.htm>
- <http://netlab.cse.yzu.edu.tw/~statue/freebsd/>
- <http://www.twbsd.org/cht/book/index.php>
- <http://netlab.cse.yzu.edu.tw/~statue/freebsd/docs/csh/>
- <http://www.freebsd.org.hk/html/other/shell1.html>

-
- ❑ <http://www.mgt.ncu.edu.tw/~dino/unix/shell03.htm>
 - ❑ http://ccca.nctu.edu.tw/~chliao/cgi-bin/archives/2004/10/unix_tcsh.html
 - Differences between csh and tcsh
 - ❑ <http://www.eng.hawaii.edu/Tutor/csh.html#scripts>
 - The C Shell tutorial
-

freebsd Pink forever

- <http://www.youtube.com/watch?v=ND-EkYxnFtI>