UNIX 簡介

A Very Brief History of UNIX

UNIX 是一九六〇年代末期在美國 Bell Labs 由 Ken Thompson and Dennis Ritchie 以及其他同事合作發展出來的一個多人多工的作業系統。由於七〇年代中比較低價的 mini-computer (~US $ 200,000) 正開始普及，美國各大學的電腦系開始買的起自已的電腦進行作業系統方面的實驗及研究，一般商用的作業系統不能滿足他們的需要，而 Bell Labs 卻以幾乎免費的條件提供 UNIX 給學術研究機構使用。尤其特別的是 UNIX 的授權方式不但包含了它完整的原始碼(source code)，而且同意使用者可以依需要修改作業系統。因此在七〇年代起，許多大學開始培養出大量對 UNIX 從裡到外都非常精通的高手，這些人對日後 UNIX 的普及有決定性的影響。再者這種無私開放的精神慢慢變成了 UNIX 的傳統，對後續多種軟體及系統方面的研究發展有很大的貢獻。不過在另一方面這也造成了多種 UNIX 版本的出現，偶而也產生一些困擾。

影響 UNIX 普及率的另外一件重要大事發生在八〇年代初期，當時 Internet 各項標準正開始成形，但因支援 Internet 的軟體不多並沒有太受到重視。但在八〇年初期，一個叫做 BSD 的 UNIX 版本把 Internet 的軟體納入成為其基本作業系統的一部份，也就是說一部電腦只要安裝了 BSD UNIX，它馬上就可以上 Internet 了。這種情況就好像早期 Windows 3.1 沒有內建 Internet，所以上網並不是很方便，但到了 Windows 95 內建了 Internet，上網就容易了多了一樣。這件事的影響是早期使用 Internet 的人一定需要使用 UNIX，幾乎所有的 Internet 軟體在早期都是在 UNIX 上發展的，這情況一直到了九〇年中期隨著 Windows 95 及 Windows NT 的普及才慢慢改變。

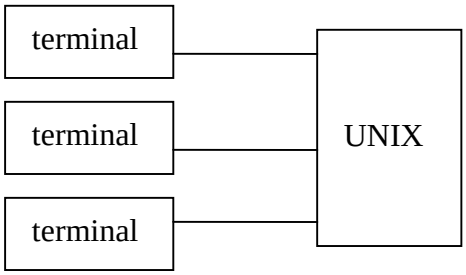
UNIX 還有一些重要或有趣的特性，例如它是第一個大部份（90%以上）由高階語言寫成的作業系統等等。其他的特性在後面章節遇到時再做說明。

對學資工的人來講，UNIX 是一個非常重要的工具，未來有許多課要在 UNIX 上講，因此我們在此捨容易使用的 Windows 介面改用 UNIX 上 Internet，因為這樣

可以一方面學 UNIX，二方面也可學到最基本、不帶任何花俏裝飾的 Internet services，對同學們應有很大的幫助。

Access UNIX Through Terminals

現下同學們熟悉的個人電腦（PC）是八十年代後的產物，在主機殼內包含有 CPU，硬碟，記憶體等元件，都是一些設計精密、功能強大的電子產品，只不過因為科技的進步，價格夠低，所以設計成一個人專用的使用方式。早期的電腦都十分昂貴，必需很多人分享才符合經濟效益。而且每個想用電腦的人也只能用一種以現在角度看來十分原始的設備上機。這種設備雖然也有 monitor 和 keyboard，乍看之下很像 PC，但和 PC 相比可差多了，因為它既沒有 CPU，也沒有硬碟，記憶體就是有也少的可憐。這樣的上機設備叫做終端機（terminal）。不但如此，大多數的 terminals 都是以直接連接的方式掛在主機上，往往有數十台 terminals 放在一間教室內供人使用，叫做 terminal room。由於這種連線方式有距離的限制，terminal room 通常離主機也不能太遠。

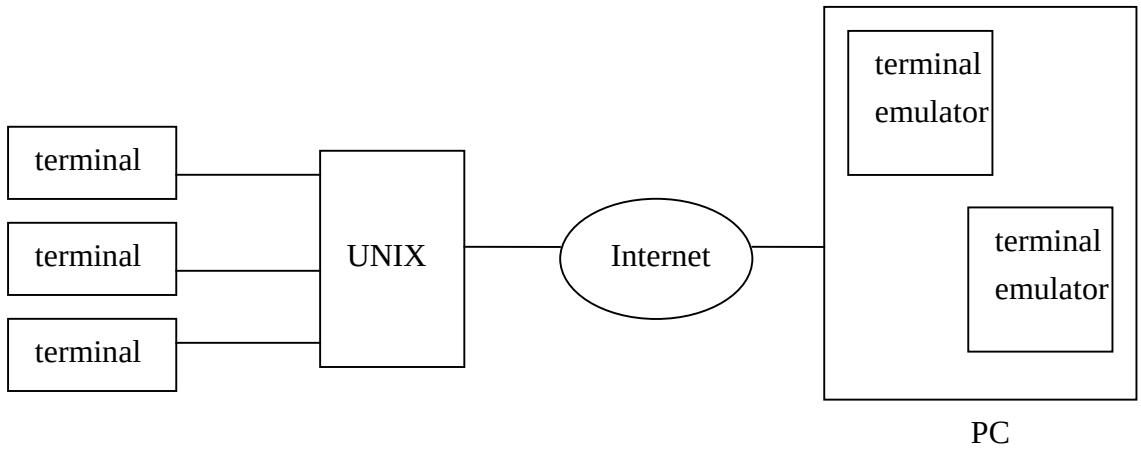


早期的 UNIX 使用 terminal 直接連上主機

因為 terminal 是很原始的設備，連 mouse 都沒有，所以時下流行的視窗介面不能使用。換言之，想用 terminal 上機，只有一種模式，就是文字模式。在此模式下，使用者在 keyboard 下按下鍵，這個動作會導致一些文字訊號傳到主機。主機有任何回給使用者的訊息也必須是利用傳送一些文字訊號的方式傳回 terminal，terminal 再將這些訊號以適當方式呈現在 monitor 上。使用 UNIX 正是這種方式。

Terminal Emulator

Terminal 既是那麼古早以前的產品，我們現在難到還得去古董店找 terminal 才能上 UNIX 麼？當然不是。前面提到現在的 PC 功能十分強大，只要我們能夠跑一種軟體讓一台 PC 能夠模擬成一台 terminal，就可以用和以前一模一樣的方法用 UNIX 了。這種軟體就叫做 terminal emulator。市面上的 terminal emulator 有許多種，本校電腦教室中可用的有 Windows 95 內建的 telnet 程式以及一種叫 netterm 的軟體。這些軟體的使用方式不難，上機上課時亦有提到，請自行學習。



Internet 環境下連上 UNIX 是使用 PC and terminal emulator

此地有兩點要特別注意：第一、雖然 terminal emulator 模擬一個 terminal，但現在上 UNIX 比以前的花樣多的多了。例如現在只要你連上 Internet 這朵雲，你就可以連上 Internet 上的任何一台 UNIX，而不是像以前一樣只能連上 terminal room 旁的那一台。更何況你可以一次跑上兩個以上的 terminal emulators，那你的一部 PC 就等於是兩部以上的 terminal 了。第二、雖然現在用 PC 連 UNIX，但是基本的操作原則並沒有變，我們仍是使用 terminal（只不過是模擬出來的 terminal）上機，也就是說，我們仍然是使用文字模式。

這種從遠端透過 Internet 使用 UNIX 的方式自然也使用了網路上 client and server 的觀念。執行 terminal emulator 的 PC 是 client，UNIX 主機是 server。使用到的服務叫做 remote login。

Logging in to UNIX

講到這裡我們知道如何用 terminal emulator 上 UNIX 了。其實這時我們已經用到了另一種 Internet service，叫做 remote login。上 UNIX 必需要經過身份確認的步驟，你必需告訴 UNIX 你的帳號（account 也叫做 user name）及正確的密碼，UNIX 檢查無誤後才讓你進入，這個動作叫做 login。當 login 的動作是透過 Internet 從遠端進行的時候，就叫做 remote login。

無論是近端或遠端，login steps 是一樣的。你先看到一小段 message，然後是如下一行的 login prompt（prompt 是提示的意思）：

login:

輸入你的帳號後，UNIX 會問你密碼：

Password:

輸入你的密碼，如果檢查無誤，恭喜你！你已經上 UNIX 了。

Two Types of Text Mode – Command Mode and Menu Mode

正式使用 UNIX 前先提一個觀念：UNIX 操作使用文字模式，但一般而言，文字模式的操作方式又可分為指令模式及選單模式兩種。在指令模式下，使用者需記住要用的指令，並在適當的時機下達指令給電腦。在選單模式下，所有能下的指令都已逐項列在螢幕上供你選擇，你不必背指令的名字，但你也無法隨心所欲的做未列入選單中的指令。顯然指令模式難學但功能不受限制，所以操作 UNIX 主要靠指令模式。

UNIX Command Format

進入個別的 UNIX 指令之前，先就 UNIX 指令的格式做一一般性的討論。大部份的 UNIX 指令都依據以下的傳統格式：

command = command name + options + arguments

command name 指明使用者要執行什麼指令，也就是他要做什麼動作，杰相當於

一個動詞。
options 是選項，用來控制指令的細節。
arguments 通常是動作的對象，也常常是檔案或目錄的名字。
請注意 options and arguments 為複數，可以為零或多個。

例如，sort 是一個 UNIX 排序的指令，可以用來將一個或多個檔案的內容依順序排好。在此 arguments 指明要排序的檔案是那幾個，options 則指明是要先大後小或先小後大排；或是依第一欄或第二欄排等等的變化。

Three Ways to Specify Path in UNIX

不論一部 UNIX 有幾顆硬碟，所有的硬碟空間都安排在同一個檔案系統之內。（Windows 則分 A:, B:, C:, D:, etc.。）這樣的檔案系統有一個最上層的 root。其次，每個使用者都有一個自己的家叫做 home directory。每次 login，UNIX 一定把這個使用者放到他的 home directory。

UNIX 有三種指定檔案位置（也叫 path）的方法，分別是：
1. absolute path，第一個字母是一個『/』。
2. relative path to home directory，第一個字母是一個『/』。
3. relative path to current directory，第一個字母不是『/』也不是『/』。

Absolute path 從 root 開始算，如『/abc/xyz/qqq』是 root 下的 abc 下的 xyz 下的 qqq。Relative path to home directory 是從 home directory 開始算，例如 b8703184 的 home directory 如果為『/home4/underg/87a/b8703184』則『~/qqq』是『/home4/underg/87a/b8703184/qqq』。Relative path to current directory 是從目前的位置（叫 current directory 或 working directory）開始算，如果目前的位置是『/home4/underg』則『85b/qqq』指的是『/home4/underg/85b/qqq』。

在這三種方式中也可以用『..』代表向上走一層。另外，『.』表示停留在這一層。所以『/abc/xyz/../pqr/./qqq/sss/..』指的是『/abc/pqr/qqq』。

順便提一下，UNIX 中大寫字母（uppercase）和小寫字母（lowercase）是不一樣的。這點和 DOS 不分大小寫是不同的。UNIX is case-sensitive and DOS is case-insensitive.

UNIX Session -- Command

一旦 login 上了 UNIX 之後，就會看到一些系統管理者留給使用者看的訊息，例如何時要關機做維修或備份等。然後就可看到一個叫做 command prompt 的提示，在我們的 csa500 上，這提示通常是

csa500.isu.edu.tw >

但也可能有其他的可能。command prompt 是 UNIX 系統用來告訴使用者：”I am ready, please enter your command”。你在這裡打入的提令很清楚的是給 UNIX 的指令。

從這時起你與 UNIX 的互動會是：你輸入一個 UNIX 指令，UNIX 執行這指令，這可能花 0.01 秒，但也可能花上一整天，螢幕上也有可能出現一些 output，指令做完後又可看到 command prompt，你又可以輸入下一個 UNIX 指令。如此週而復始直到你離開 UNIX 為止。離開 UNIX 的指令是 logout 或 exit，兩者皆可。從 login 到 logout 的這段時間，叫做一個 UNIX session。

Recording Your UNIX Session with Script

UNIX 上有個很好用的指令叫做 script。它的功能是完整的記錄從輸入 script 指令到用 exit 離開 script 這段時間內每一個螢幕及鍵盤的動作。我的建議是 login 上 UNIX 後立刻執行 script，要離開 UNIX 前才離開 script，這樣你在 UNIX 上所有的動作都留下記錄，以後可以回來慢慢的研究。

Script 通常把記錄留在一個叫 transcript 的檔案裡，舊的 transcript 則會被覆蓋過去。但是你可以用 options 來控制這個記錄檔的名字以及是否覆蓋。下一節教你如何找出如何下 option。

Man -- The Most Important UNIX Command

不管你是初學者或是 UNIX 老手，毫無疑問的最重要的一個 UNIX 指令是 man。Man 是 manual page 的簡寫。UNIX 是第一個把使用者手冊編成線上手冊放到系統上的作業系統。假設你看到一個指令但不知如何用它，或是你雖知道該指令大致做什麼事但是記不得有那些 options 可用，這時就麼該用 man 來查一下這個指令的線上手冊，在 UNIX 的術語中這動作叫做查某一指令的 man page。

UNIX Manual Page Format

每個 UNIX man page 包含了以下各項標題及內含的資料，以 man ls 為例：

1. Name -- 指令名字及一段很簡短的說明解釋該指令的功能。

   ls - Lists and generates statistics for files

   ls 就是 DOS 裡的 dir 指令。
2. Synopsis -- 指令的使用法（包括有效的 options）及較詳細的說明。

   ls [-aAbcCdfFgilLmnopqrRstux1] [file ... | directory ...]

   The ls command writes …（下略）

   在此處說明一個資工常用的習慣用法，那就是在左右方括弧中的項目是可有可無（英文叫做 optional）。此處可知 ls 可以不帶任何 options，但也可以用 aAbcCdf…ux1 等十幾種 options。
3. Flags -- 說明各種 options 的用法。例如 ls -a 可印出隱藏檔，ls -l 印出多項檔案的相關資料，ls -t 用時間先後順序列出檔案。
4. Description -- 更詳盡的說明。
5. Examples -- 使用範例。
6. Files -- 相關檔案。
7. Related Information -- 其他相關的指令。

由於所有的 UNIX man page 都是英文寫的，你必須提升你的英文能力，而且越快越好。

Reverse Man Page Lookup – man -k

有 man 的幫助，即使是新手也可以學會用許多 UNIX 的指令。但是有的時候，我們知道想叫 UNIX 做什麼事但卻不知適當的指令名字。這事是由 man -k 完成。也就是說-k 是一個與 man 方向相反的查尋指令 option：man 告訴我們什麼指令名字做什麼事，man -k 反過來從動作回頭來查指令的名字。

man -k 是利用關鍵字（keyword）來查指令名字。例如我們想知道刪除檔案的 UNIX 指令，我們可以用下列指令：

man -k file

這個指令會列出一長串與檔案有關的指令，其中可以找到刪除檔案的指令 rm。

離開 man 之前最後一個問題：請問 man man 這個指令在做什麼？

Some Frequently Used UNIX Command

以下是一些常用的 UNIX 指令，打底線的會在後面講到，其餘的在此不一一解釋，你可以利用 man 查出如何使用這些指令：

1. File and directory: cd, pwd, mkdir, rmdir, cp, mv, rm,
2. Examine file content: cat, more, head, tail, od
3. File comparison and searching: diff, cmp, grep
4. Read manual page: man, man -k
5. Get system information: who, w, finger, uptime, date
6. Change password: passwd
7. Text editor: vi, emacs, pico
8. Job control: ps, jobs, ^Z, fg, bg, &
9. Internet services: cpine, mail, mailx, telnet, ftp, archie,
10. Network diagnostic: ping, traceroute
11. Miscellaneous: wc, du, df, script, cal, which, sort, echo

Pico – A Simple UNIX Editor

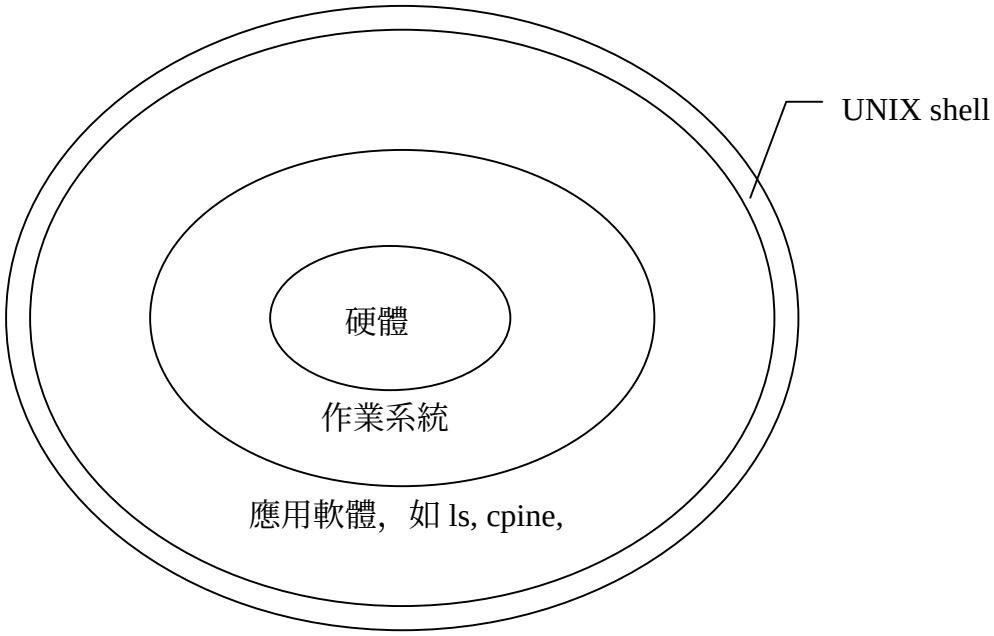
UNIX 上最傳統的文字編輯器是 vi，許多 UNIX 高手喜歡用另外一個叫做 emacs

的 text editor。這兩個 editors 都是功能非常強大但需要花一段時間才學得精的程式。這門課對 editor 的要求並不多, 因此我們建議使用一個較易學的簡單 editor -- pico。

Pico 和 Windows 中的筆記本很像, 也很好用。螢幕的最下方有兩行指令的 summary。值得一提的是它用標準的 UNIX notation ^X, 表示 control-X, 也就是按住 control 鍵不放再按 X 鍵。

Layers of A UNIX Machine

從這節起我們討一些比較深的 UNIX 觀念。一部 UNIX 主機內的軟體及硬體通常可以用下圖來描述出不同的層次, 藉以了解 UNIX 內部的結構：



UNIX machine 可以分成四層

最內部的一層的電腦的硬體。硬體外是作業系統, 負責管理硬體使其好用且更有效率。再外層是應用軟體, 我們學的絕大多數的 UNIX 指令都是屬於這類應用軟體。最外層也是最接近使用者的是 UNIX shell。

Shell – Command Interpreter

Shell 是 UNIX 自創的專有名辭，描寫它位在其餘軟硬體之外，就像一層殼。不過 shell 的基本原理倒不是新的觀念，在其他機器上也已行之有年，不同的是在他處不叫做 shell 而叫做 command interpreter, 意思是一個專以解釋及處理使用者輸入的指令為主要工作的軟體。這兩者可視為同義字，都是使用者直接對話的對象。

如果我們把一部 UNIX 看成一家公司，shell 就是公司大門口專門負責接待顧客的接待人員。任何顧客進入公司第一個看到的就是他。顧客的問題如果他能解答就直接做答，如果他不能自已解答就會引導顧客給其他適當的員工解決問題。

以上的比喻其實就是 shell 的基本工作原理。我們可以把 shell 的基本工作寫成以下的幾個步驟。請注意這裡給的是最基礎的工作，我們後面會指出許多 shell 更複雜的功能，這些步驟將會跟著變的更複雜。

1. 在螢幕上印出 command prompt，等使用者輸入指令。
2. 把輸入的指令分成 tokens 並做一些額外的處理。Token 是資工裡常用的專有名辭，做為一個不能再分割的小單位解釋。例如輸入 man -k directory，則 man, -k,以及 directory 各是一個不能再割的更小的單位，所以這行指令分出三個 tokens。額外的處理有非常多種，下面會擇簡單的做些介紹。
3. 如果指令是 shell 自已就能處理的指令（叫內部指令），shell 自已執行該指令。否則 shell 就會試著找到負責該指令的程式（叫外部指令），做出一個合適的工作環境（術語叫做 create a process）供該指令執行。如果找不到指令，印出一行錯誤訊息 – Command not found。
4. 回到 step 1。

UNIX 就是如此週而復始的重複這些步驟，直到第三步待執行的指令是 exit 或 logout，那時 shell 才結束工作。

在這裡我們給幾個例子用以說明上述的步驟。

例一、在第一步 shell 印出 csa500>後，使用者輸入『ls』再按 enter，進入第二步。Shell 分割 token 的結果只有一個 token，ls 不是內部指令所以 shell 找出專門負責 ls 工作的應用程式（這個程式依 UNIX 的規定就叫做 ls），shell 接著 create a process，然後讓 ls 在這個環境裡執行。使用者看到的是一些檔案的資料出現在螢幕上。

例二、shell 印出 csa500>後，使用者輸入『cd /usr/bin』。Shell 發現共有兩個 tokens，分別為 cd 及/usr/bin。Shell 沒有需要做任何額外的處理。cd 剛好是一個內部指令，shell 並不需要 create a process 而是直接就自己把 cd 的動作完成了。螢幕上沒有任何輸出，但是 current directory 的位置已經換到/usr/bin 下面去了。

例三、使用者本意是要輸入『ls -l』，但卻漏了空白鍵，變成了『ls-l』。Shell 找到一個叫 ls-l 的 token。Shell 沒有需要做任何額外的處理。ls-l 不是內部指令，所以 shell 試著找找有沒有 ls-l 這個外部指令，找的結果是沒有，shell 印出 Command not found 的錯誤訊息。

First Example of Shell Processing – Filename Expansion

現在我們用兩個具代表性的例子來看看 shell 分解完 token 之後再如何續做處理。我們的第一個例子對會用 DOS 的同學不會陌生，在 UNIX 及 DOS 裡，指令中出現的『*』可以代表檔案名字內的零個或多個的任何字元，這叫做 filename expansion 或 filename substitution。它處理的方式如下：假設 current directory 內共有五個檔案，分別是 aa, ab, acd, wyz, xyz。使用者輸入『ls -s a*』。Shell 發現共有三個 tokens，分別為 ls, -1 及 a*。Shell 一旦看到了『*』就知道這是指揮它做 filename expansion 的命令，於是它把 current directory 內所有名字長的像 a*（a 後面接零個或多個的任何字元）的檔案找出來，一共有三個 – aa, ab 和 acd。Shell 把『a*』換成 aa, ab 和 acd 三個 tokens，連前面的 ls 與-l 總共是五個 tokens 一起送給外部指令 ls 在一個新的 process 裡執行。最後的結果是三行 output 出現在螢

幕上，分別是 aa, ab 和 acd 的檔案資訊。

Filename expansion 功能中還有其他的字元可用，例如『?』可以代表任何一個單一的字元，『[4kp]』可以代表 4, k, p 這三個字元中的任何一個。所以在上例中，『a?』代表 aa 與 ab，但不代表 acd；『[uvwx]yz』代表 wyz 及 xyz。

Second Example of Shell Processing – Command Aliasing

Shell 處理的第二個例子叫 command aliasing。Alias 是別名的意思。有時我們習慣了以某種指令名字，但偏偏 UNIX 指令名字卻不一樣，造成使用上的不方便。這時我們可以為該 UNIX 指令另取一別名，下次講到這個別名時 UNIX 會自動把它換成正式的指令名字。

例如，DOS 中刪除檔案的指令是 delete，但在 UNIX 上卻是 rm。我們如果用下面的指令

```
alias delete rm
```

Shell 會在一個叫做 alias table 的表裡記錄 delete 是 rm 的 alias。如果使用者下達 delete abc 的指令，shell 會在第二步時分解出兩個 tokens，分別是 delete 和 abc。第一個 token 剛好在表中查到是 rm 的別名。Shell 此時會把 delete 這個 token 換成 rm，create a process，將 rm 及 abc 這兩個 tokens 交給外部指令 rm 在一個新的 process 內執行，結果是 abc 這個檔案將被刪除，正如當初的期待一樣。

要檢查 shell 內部的 alias table，只需在 command prompt 後輸入 alias 即可。

Choices of UNIX Shells

UNIX 上的 shells 不至一種，每種 shell 的基本作業原則相似，但功能有差異，特別是在第二步額外處理的部份。常用的有 Bourne shell, C shell, TC shell。Bourne shell 是 UNIX 上歷史最悠久的 shell，功能較簡單。C shell 比 Bourne shell 好用，也是 csa500 為一般使用者內定的 shell。TC shell 有一些更方便的功能，我的建議

是先花點時間學好 C shell，半年一年後再換成 TC shell。

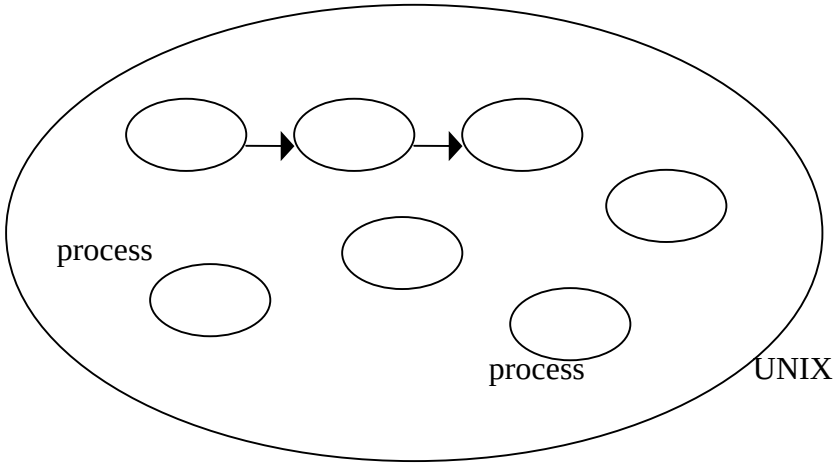
For Further Reading – C Shell Man Page

我特別建議同學們把 C shell 的 man page 用 man csh 的指令看一看，甚至應該印出來慢慢看上五遍十遍，對你學 UNIX 會有非常大的幫助。如何列印 UNIX man page 將在以後說明。

What Is a Process

對資工系的同學來說，process 是一個非常重要的觀念，雖然目前我們還沒有足夠的準備及能力深入這個題目，但是先有一些正確的了解對下面要講的 shell 的功能會有幫助。

日常生活裡想要做好一件事必須要先有某些先決條件或環境。比方說要考試了就得帶著課本或講義找個安靜的地方看書，想練球須要有球場及球具。同樣的要執行一個指令，必須要有一些資源，例如 CPU 及 memory。現代的作業系統的責任是把這些資源妥善保管運用，不讓多工系統中的某一項工作影響到另一項工作。為達到此目的，作業系統會限制每件工作（其實就是一個單一的指令）一定要在滿足某些條件的特殊環境裡才能執行。這樣的環境就叫做一個 process。我們將用一個橢圓來表示一個 process。現代的作業系統都是多工的系統，也就是可以同時有多個 process 存在於系統之中，各 process 做自己的事，互不干擾。



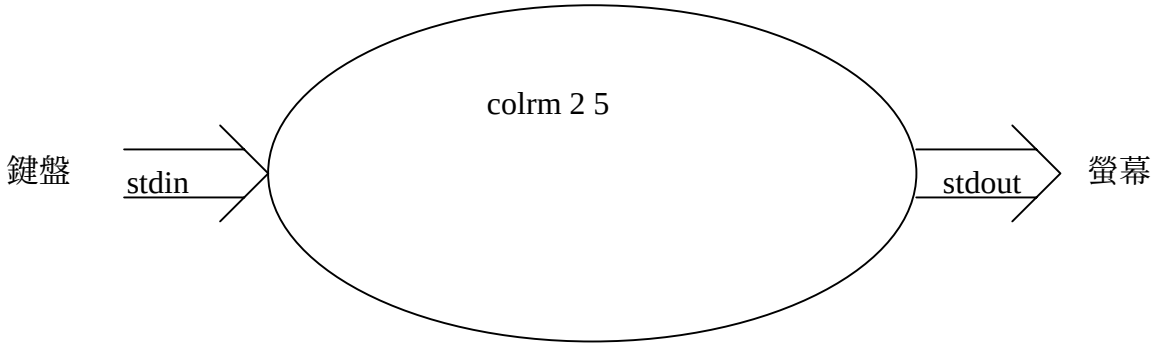
多工的系統中可以同時有多個 process 存在

前面講到 shell 時提到，shell 基本步驟裡的第三步就是做一個 process 出來供指令執行之用。不過也有其他的程式可以做 process 出來，我們在稍後講 ftp 時將會提到。有些 processes 之間有某種關係，這在圖中用線連在一起來表示。

Standard Input and Output

每個 process 都有一個號碼叫做 process ID(PID)，這個號碼唯一代表該 process，所以絕不會同號。其他有關 process 的一些特性超出大一的程度，在此不多提。與我們下面要講的有關的一項特性是每個 process 在創始之時就有一個輸入及一個輸出的管道，分別叫做 standard input and standard output。Standard input 在一般情形下是接在鍵盤上，任何時候這個 process 從 standard input 讀東西時就是從鍵盤上讀。Standard output 在一般情形下是接在螢幕上，任何時候這個 process 送東西給 standand output 時會出現在螢幕上。Standard input 在 UNIX 的習慣裡常寫成 stdin，standard output 寫成 stdout。

我們用一個 UNIX 指令叫 colrm 的為例，colrm 的作用是將輸入的每一行中的某些列刪除，例如 colrm 2 5 會把輸入的每一行中的第二列到第五列刪除。這個指令可用下圖表示：



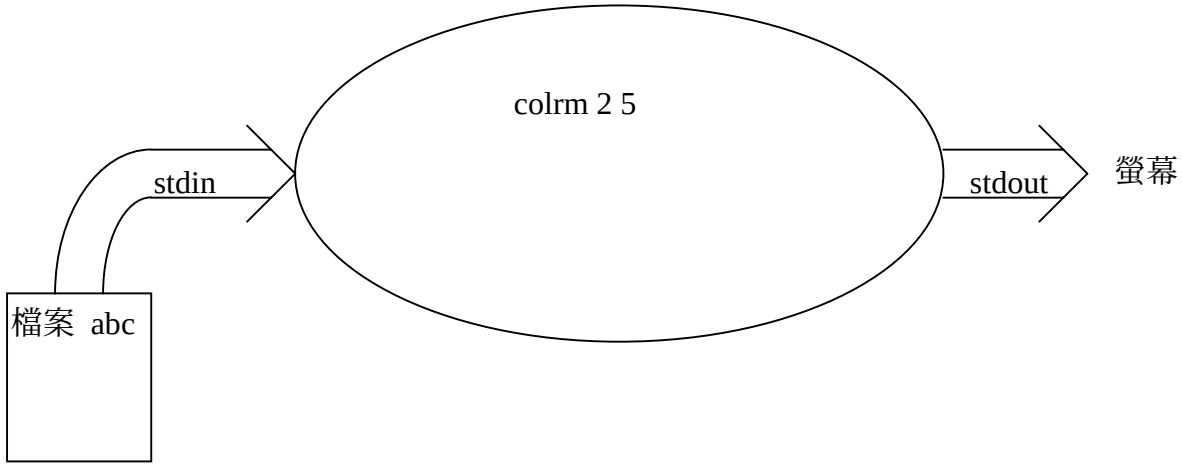
這張圖清楚的顯示，stdin 是由鍵盤輸入，stdout 由螢幕輸出。如果你鍵入兩行資料，第一行是 123456789，第二行是 abcdefghijklmnopqrstuvwxyz，colrm 會刪去每一行的第二到第五列，你會在螢幕上看到兩行輸出，第一行是 16789，第二行是 afghijklmnopqrstuvwxyz。

值得特別指出的是在 UNIX 的觀念裡，colrm 知道的是它應由 stdin 讀資料，至於 stdin 是連到鍵盤這件事，colrm 完全不需要知道。

除了 stdin and stdout 之外，你們大二學 C/C++時還會學到另一個輸出管道叫 standard error。

I/O Redirection

但是有些時候，我們不打算直接由鍵盤輸入，例如要輸入的資料量太大，已分由二十個人各負責一部份打好且整合成一個檔案叫 abc 了。這時是否能夠叫 colrm 去檔案裡讀資料呢？在 UNIX 之前，這種小改變常常需要修改程式，但是在 UNIX 裡不需要修改程式-- colrm 已經能夠從 standard input 讀資料，只要我們能想法子讓執行 colrm 的 process 把 stdin 接上檔案 abc 就好了。這情形可用下圖來表示：



standard input is redirected to read from a file

如何讓 colrm 這個 process 改變 stdin 方向呢？回憶一下這個 process 是誰做出來的。答案是 shell，所以如果在下指令給 shell 時就傳達了希望 stdin 改向的意思，shell 不就可以在做出 process 時加一點工製造出改向的效果嗎？的確如此。我們如果下達以下的指令給 shell：

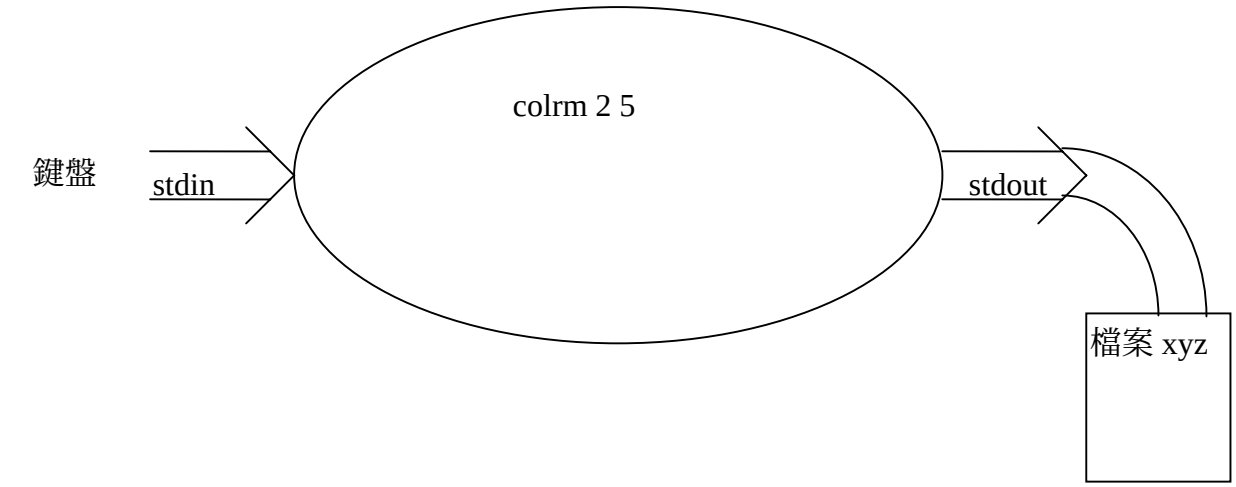
```
colrm 2 5 < abc
```

就可以達到上圖的效果，由 stdin -- 這次是檔案 abc -- 讀取資料，刪去每行的二至五列，再將結果送到 stdin -- 螢幕上去。

同樣的道理，我們可以請 shell 將一 process 的輸出改向到一個檔案，例如

```
colrm 2 5 > xyz
```

會將鍵盤輸入刪去二至五列後將答案送去檔案 xyz，如下圖所示：

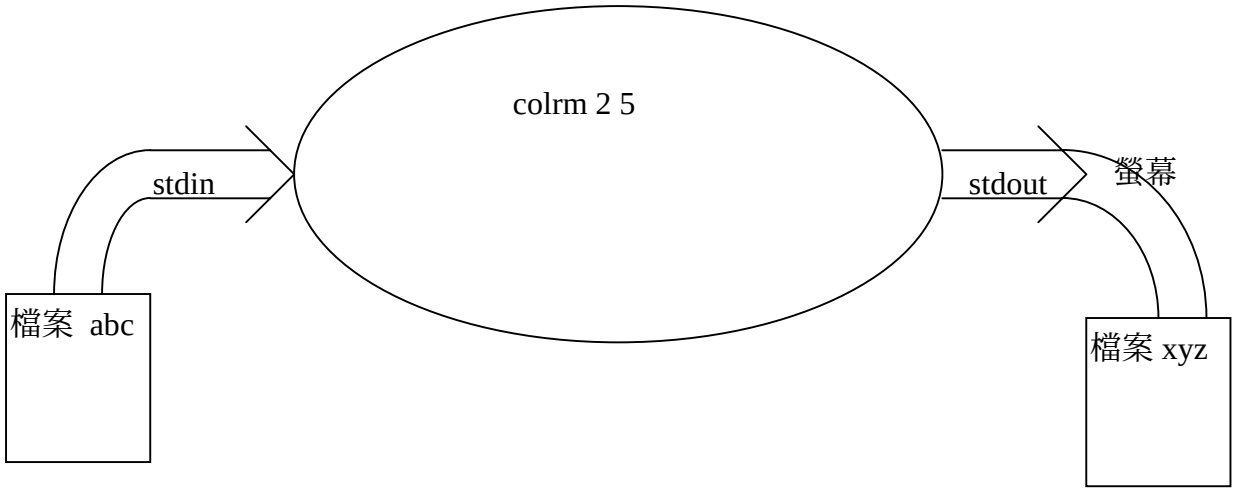


standard input is redirected to write to a file

當然你也可以要求 stdin and stdout 同時改向，指令寫法如下：

colrm 2 5 < abc > xyz

改向的效果顯示如下圖：



Both stdin and stdout are redirected

前面提到你應該將 csh 的 man page 印出來，現在你應該知道如何將 man page 存成檔案了，指令很簡單：man csh > filename。下面在 ftp 節裡會教你如何將這個檔案傳到 PC 上以便列印。

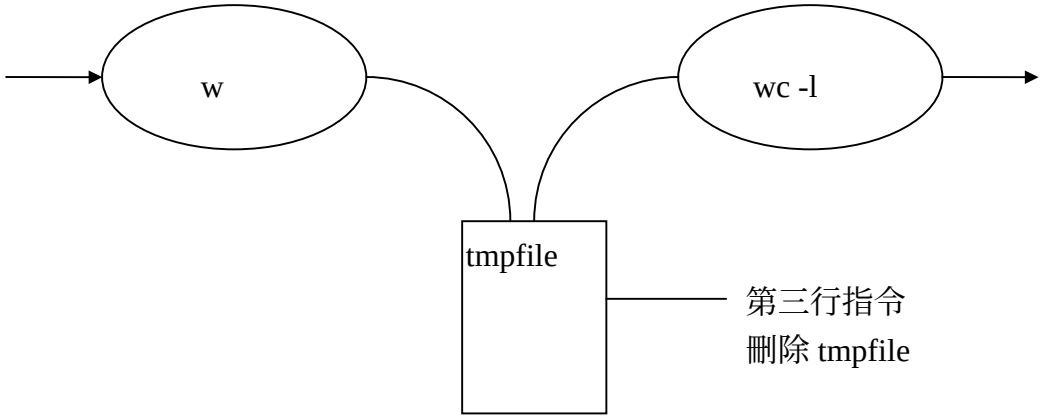
最後提一下 output redirection 的一個變種。一般的 output redirection 下如果輸出的檔案已經存在，舊的資料會被覆蓋過去，但如果使用>>代替>的話，檔案內原有的內容會被保留，新寫入的資料會被接在舊資料的後面，英文叫做 append。

Pipe

回憶一下有一個指令叫 w 可以列出目前上機的人，如果上機的人太多，螢幕會很快的捲過去，很難算出有幾行。但是又有一個指令叫 wc -l 可以數出 stdin 有幾行。我們如果想知道現在有幾人上機，可以把 w 的結果存入一個暫時的檔案，用 wc -l 數這個檔案有幾行，再把暫時檔刪除，也就時用下面三個指令：

```
w > tmpfile
wc -l < tmpfile    （在此可知有幾人上機）
rm tmpfile
```

這種做法可以用下圖表示。第一個指令做出 w 的 process，第二個指令做出 wc -l 的 process，第三個指令刪除 tmpfile。

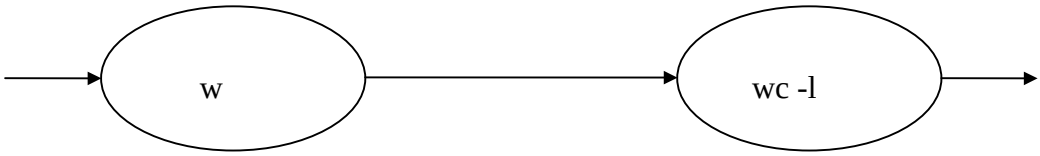


這做法有幾個缺點，指令太多，還需要暫時檔等等。UNIX shell 有個好用的功能叫做 pipe 正好合用。我們如果下達下面的指令：

```
w | wc -l
```

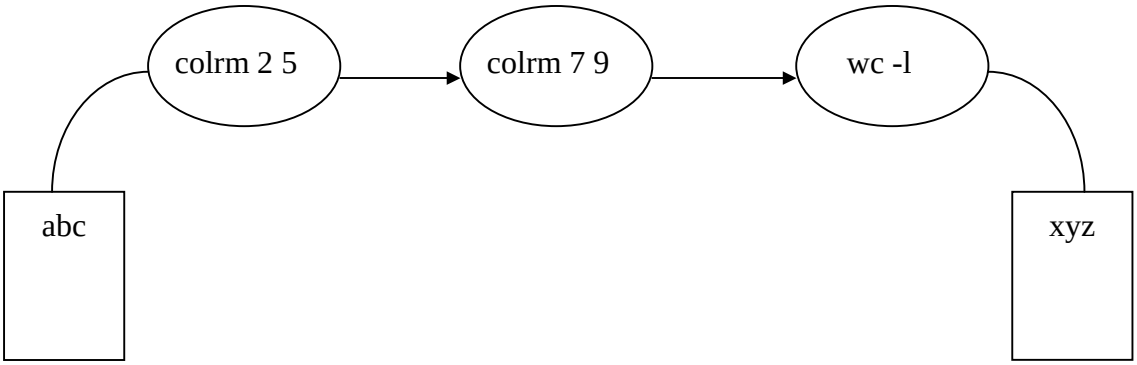
Shell 會做出兩個 processes 出來，第一個執行 w，第二個執行 wc -l ，而且 shell 會把第一個 process 的 stdout 和第二個 process 的 stdin 連接在一起。任何從第一個 process stdout 輸出的資料會被第二個 process 當做 stdin 使用，這種做法可以

達到如前例相同的效果，既簡捷又不需要 tmpfile，是 UNIX 裡很有用的發明。（會用 DOS 的同學們也許學過 I/O redirection 及 pipe，這是因為 DOS 打從 2.0 版後就從 UNIX 借了許多好用的觀念，I/O redirection 及 pipe 也在內。



Pipe

Pipe 除了可以一次連接多個指令外，還可以和 I/O redirectio 合著一起用，例如 colrm 2 5 < abc | colrm 7 9 | wc -l > xyz 的作用可以用下圖表示：



Combining pipe and I/O redirection

這裡要說明一下的是以前提到的 shell 的基本步驟是不考慮到 I/O redirection 及 pipe 的簡化的說明，現在加上了這些新功能，shell 處理步驟自然會複雜的多，例如第二步打散成 tokens 就不能只用空白字元，因為>,<,| 這三個字元都可以有分隔 token 的作用。同是第二步的 command aliasing 可能會因為有一行中有多個指令而必需重複為每個指令做一次。再如第三步中的 create process 步驟，如一行中有多個指令就必需做出多個 processes，而且要適當的連接這些 processes 的 stdin and stdout。但這些變化並不會影響到 shell 處理指令的基本結構，在此就不詳細說明了。

Job Control

如果我們使用了 pipe，那麼一行指令中會出現多個個別的指令，需要多個 processes 才能順利執行。這些相互有關連的 processes 在使用者眼中是一個整個的動作，叫做一個 job。我們到目前為止都是在 command prompt 後鍵入指令，再等它執行完畢。如果這 job 要花很多時間我們也只好等，如果這 job 一直跑不出來，那會一直等下去，顯然一定需要有控制 job 的方法才行。

第一個控制 job 的方法是把它殺掉，殺掉後相關的 process 就不復存在於系統中，command prompt 也會跳出來接受下一個指令。要殺正在執行的 job 用^C（Control-C）。

第二個控制 job 的方法是讓它暫時停止，command prompt 跳出來接受下一個指令。暫停的 job 可以稍後再恢復執行。暫停正在執行的 job 用^Z（Control-Z）。

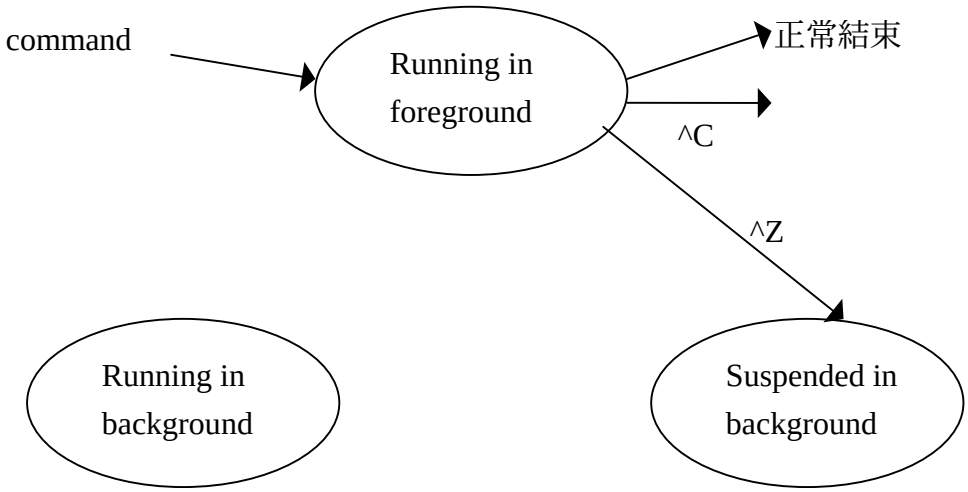
看到這裡我們知道一個有兩種可能的狀態：正在執行及暫停執行。其實真正的情況比這要稍微復雜一點，不過我們先要解釋兩個觀念：foreground and background。

Foreground and Background Jobs

我們如果把螢幕看做是一個舞台，一般正在執行的 job 可視為正在台前或幕前表現，所以我們稱這個 job 目前在 foreground。這個幕前的 job 就是你輸入要 UNIX 執行的指令，而在指令與指令之間 shell 會短暫的出現在幕前一下接受下一個指令。前面提到被暫時停止的 job 可以看成被 shell 暫時擠到幕後或 background。不過在幕後的 jobs 有兩種可能性：暫停或繼續執行。這好比表演欲高的人在舞台後依舊可以繼續表演一樣。在 background 執行的 job 會繼續使用 CPU，會繼續有進展，該做的工作做完了它就會正常結束。相反的，暫停的 job 就算只差幾個 microseconds 就結束，但一旦停下來就不會進行，除非再啟動起來否則不能正常結束。

State Transition Diagram for Job Control

這三種狀態正好用以前講過的 state transition diagram 表示。三個圈代表三個可能的狀態，箭頭代表改變狀態的指令。我們正常啟動一個 job 後它會在幕前執行，再加上^C and ^Z，我們目前得到的 state transition diagram 如下圖：



Incomplete state transition diagram for job control

本圖目前有許多 missing links，使各狀態間之切換無法進行，以下我們逐步討論多個相關指令，你應該每學一個指令就自已在圖上加上相對的箭頭，這樣對你了解 state transition diagram 的意義會有很大幫助。

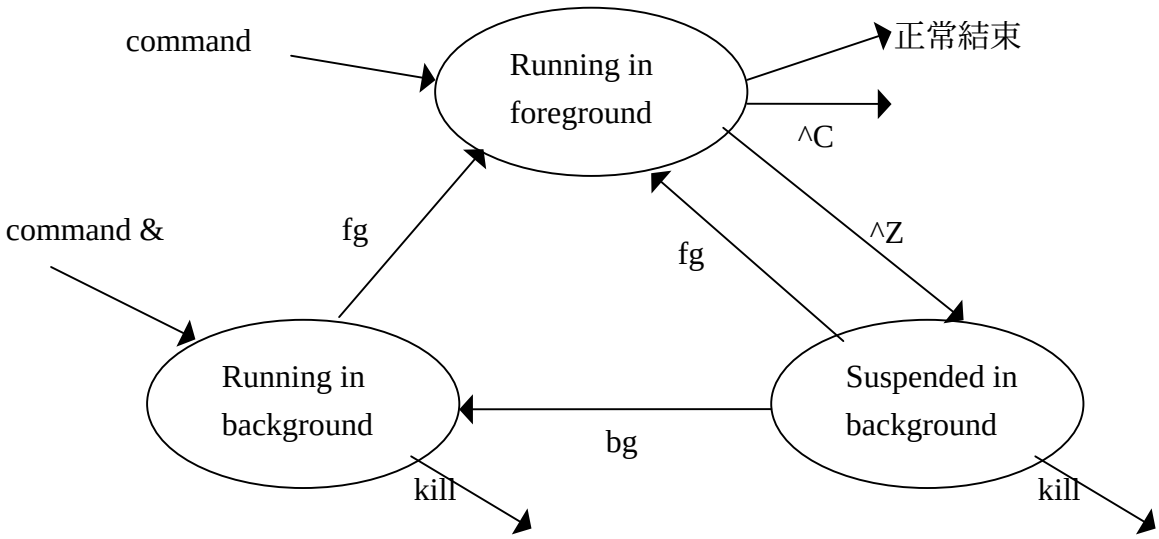
以下是相關的指令：

- jobs：本指令只有查詢功能，不改變狀態。jobs 會列出目前你有多少 jobs 以及它們的狀態。任何時候只能有一個 job 在 foreground，但可以有任何數目的 background jobs。每個列出的 job 都有一個號碼，供其他指令中指定個別的工作用。
- bg：將一個 job 由 suspended in background 改成 running in background，這種 jobs 會繼續進行，一直到它正常結束或是需要由鍵盤輸入時才停止。Background jobs 沒有使用鍵盤的機會因為鍵盤是給 foreground job 用的。例如 bg %3 會指定第三號的 job 把它改為 running in background。

如果只下 bg，那麼最新被暫停的 job 會被指定。

- fg：將一個 background job（無論是 running or suspended）丟回 foreground 執行，shell 暫時休息。
- kill：將一個 background job 殺掉。kill %3 會指定第三號的 job 把它殺掉。kill 亦可以殺掉各別的 process。
- command &：當我們想在執行一個指令之初就直接讓它進入 background 執行，shell 馬上回來接受下一個指令，我們可以如以前一樣的下指令，但在指令的最後加上&這個符號（英文讀做 ampersand）。所有以前講的 I/O redirection, pipe, etc 都依然可用。

你的 state transition diagram 是否正確，如下圖？



Completed state transition diagram for job control

上圖說明了一個 job（以及其中的 processes）從生到死的過程，任何時間只能在一個狀態內，不能多也不能少。第十七頁的圖和本圖的關係則是：當一個 job 是在圖內三種狀態時，job 裡的一個或多個 processes 會出現在 17 頁圖中。反過來說，任何一個 17 頁圖中的 process 或 job 一定是在上述的三種狀態之中的一種。

UNIX 的部份大致到此結束，你應該由多用中學習各種常用的指令，讓自已用的

更熟，別忘了找時間看 csh man page。

File Transfer

現在我們來學如何在 UNIX 上使用檔案傳輸這個 Internet service，英文叫 ftp，是 File Transfer Protocol 的簡寫。（Well，DOS 裡也有同樣的指令，也叫 ftp。我們在這裡講的在 UNIX 及 DOS 裡都有效。）雖然在圖形介面上也可以傳輸檔案，但文字模式的 ftp 讓我們接觸到 ftp 裡非常低層的細節，是資工系學生想充分了解 ftp 不可或缺的過程。

過去提到的 Internet 基本觀念在此依舊有效：要使用服務必需啟動服務的 client 軟體，這個軟體就叫做 ftp，有一個文字模式的使用介面。從進入 ftp 到用 bye 或 quit 指令離開之前，你都是在和 ftp 對話。為了讓使用者不至於弄混起見，ftp 會印出自已的 command prompt，就是 ftp>。你在 ftp>後面打的必需是 ftp 看的懂的指令，而不是 UNIX 或 DOS 的指令。（雖然有些 ftp 指令像 ls 和 cd 和 UNIX 或 DOS 指令同名，但也只是用同名之便讓使用者少背些指令而已，真正做出 ls 和 cd 的效果的不是 UNIX 或 DOS，是 ftp。尤其要先說明的是所有的 ftp 指令都是下給 client 端的，包括一些要 server 做的工作也是先下指給 client，client 再透過網路要求 server 執行適當的動作才能夠完成的。在這個層次使用 ftp 可以對它有更深的了解。

要使用 ftp，在 UNIX 或 DOS command prompt 後鍵入 ftp，看到 ftp>。使用？或 help 指令可以看到所有 ftp 看的懂的指令，我們在講解使用原理時會介紹一些指令，其餘的部份可以參考 ftp 的 man page。（用 man ftp。）

啟動 client 之後必須指定 server，這裡用 open csa500.isu.edu.tw 可以指定 csa500 為 server，ftp 會印出一些資訊，然後問你要用什麼身份 login。鍵入你在 csa500 上正確的帳號和密碼你就已經 login 可以使用檔案傳輸的功能了。

我們這裡是用類似分解動作的方式來使用 ftp：先啟動程式，再連上 server，再告

訴 server 你的帳號及密碼。這樣做的好處是對 ftp 的作業能有比較清楚的了解，例如，你可以用 user 這個 ftp 指令告訴 client 你現在要用另一個帳號 login，ftp 會重新問你一遍帳號及密碼，這時你與 csa500 的連線並沒有斷掉。又如你可以用 close 指令叫 ftp 切斷與目前 server 的連線，再用 open 指令連上另一台 server，這時你仍在 ftp 裡沒有離開它。一般不了解這些關係的人通常會用 ftp csa500.isu.edu.tw 的指令直接完成啟動軟體、連 server、login 等動作，固然也可以達到傳檔案的效果，但他對 ftp 的了解顯然是不如你了。

要知道你現在有沒有連到 server，以及其他的一些資料，可以用 status 來查看。回答中的某些狀態等下還會提到。

Login 進入 ftp server 之後你可以用 cd 移動 current directory 也可以用 ls 查看檔案名稱及資料，但是這些檔案都是在 server 上的 file system 上的檔案，而不是在你自已的 client 端的系統上。Internet 習慣上用 remote 或遠端這個字來表示 server，用 local 或近端這個字來表示 client。例如要移動近端的 current directory 不能用 cd 而要用 lcd。瀋檔案傳輸的主要功能也可以用 remote and local 的角度來看：如果要將檔案從 remote 傳到 local，用的指令是 put，如果要將檔案從 local 傳到 remote，用的指令是 get。

Put and get 有幾種可能的指令格式。如果你只鍵入 get，ftp 會問你 remote-file 及 local-file，你也可以把 remote-file 及 local-file 的名字加入指令內，ftp 就不會再問；你也可以不給 local-file 的名字，意思是 local-file 和 remote-file 同名。除了用 get 和 put 一次一個檔案慢慢傳之外，你也可以用 mput 及 mget 指令一次要求傳多個檔案。（m 在此是 multiple 的意思。）例如，mget * 是要求 server 把 remote 端所有 current directory 裡的檔案全部傳回來。

在使用 mget 或 mput 的時候你可能會發現有時 ftp 會一遍又一遍的問你這個檔案要不要傳，有的時候又不會問。這個是否用交談式的提示方法是由一個叫做 prompt 的指令控制的。如果現在的狀態是不問（prompting is off），打一次 prompt 會將 prompting 變成 on，再打一次又會將 prompting 變成 off，如此週而復始。目

前的狀態則可用 status 來查。

前述的這種做一遍變成 on，再一遍變回 off，再一遍變回 on 的切換方式有個專有名辭叫做 toggle。ftp 裡有好幾個其他的 toggles：

- verbose：是否會告訴使用者許多有關目前狀況的訊息。如 on，螢幕上會出現很多此類訊息，對初學 ftp 的人可以有幫助學習的效果。
- hash：是否在每傳送一些數量的資料後在螢幕上印一個#字元（英文叫做 hash mark）。它可以動態的表示出目前傳輸速度，如果###一直不斷的出來那麼檔案傳的很快；如果久久不見一個#，不是傳送的速度極慢就是根本沒有在傳。
- 其他的 toggles 還有 bell, debug, glob 等等。這些 toggles 的狀態都可用 status 來查。

另一件在傳檔案時非常重要的設定是選擇用 ascii 方式來傳送或是用 binary 方式來傳。大原則是如果檔案是純文字檔，則用 ascii 傳，其他的都用 binary 來傳。ascii 與 binary 指令分別將傳送方式設為 ascii or binary。ASCII 是目前電腦中大小寫英文字母、數字及符號的標準編碼方式。Binary 是二進位的意思，此處指的是非文字的資料型態。

為什麼要有這種麻煩？原因出在文字檔中的換行。以點矩陣印表機為例，換行的動作其實是兩個動作合成的，一個動作是印頭回到最左邊、叫 carriage return 或 CR，另一個是捲動紙張到下一行、叫 linefeed 或 LF。但是不同的系統用不同的方式表示換行。例如 UNIX 用一個 byte，但 DOS 上需要兩個 bytes 來表示。如果不做適當的轉換，一個文字拿到另一個系統上樣子就變了。這似乎是件不應該發生的事。另一方面，非文字檔（圖形檔、執行檔、壓縮檔等）極有可能碰巧出現換行的字元，如果被做了轉換則原本的檔案格式就不對了，會造成執行檔不能執行，壓縮檔解不回來等狀況。所以 ftp 完全靠使用者告訴它要傳的檔案是否應做轉換，文字檔（ascii）就做轉換，非文字檔（binary）就不做。最後要注意的是 ascii 只適用於純文字檔，Microsoft Word 的 doc 檔（例如這份講義）因為包含了字型、分段等等的其他資料，並不算是純文字檔，傳送時需要用 binary 來傳。

Escape to Shell

相信你在使用 ftp 一陣子後會覺得好像和 shell 的操作原理蠻相似的。不錯，兩者都是指令模式，所以也都有一個指令的 loop（讀指令，處理指令）。不同的是 shell 會 create process 來執行外部指令，而 ftp 指令都是內部指令由它自已獨力完成。不過 ftp 也可以 create process，而且它做出來的 process 只用來執行一種程式—shell。

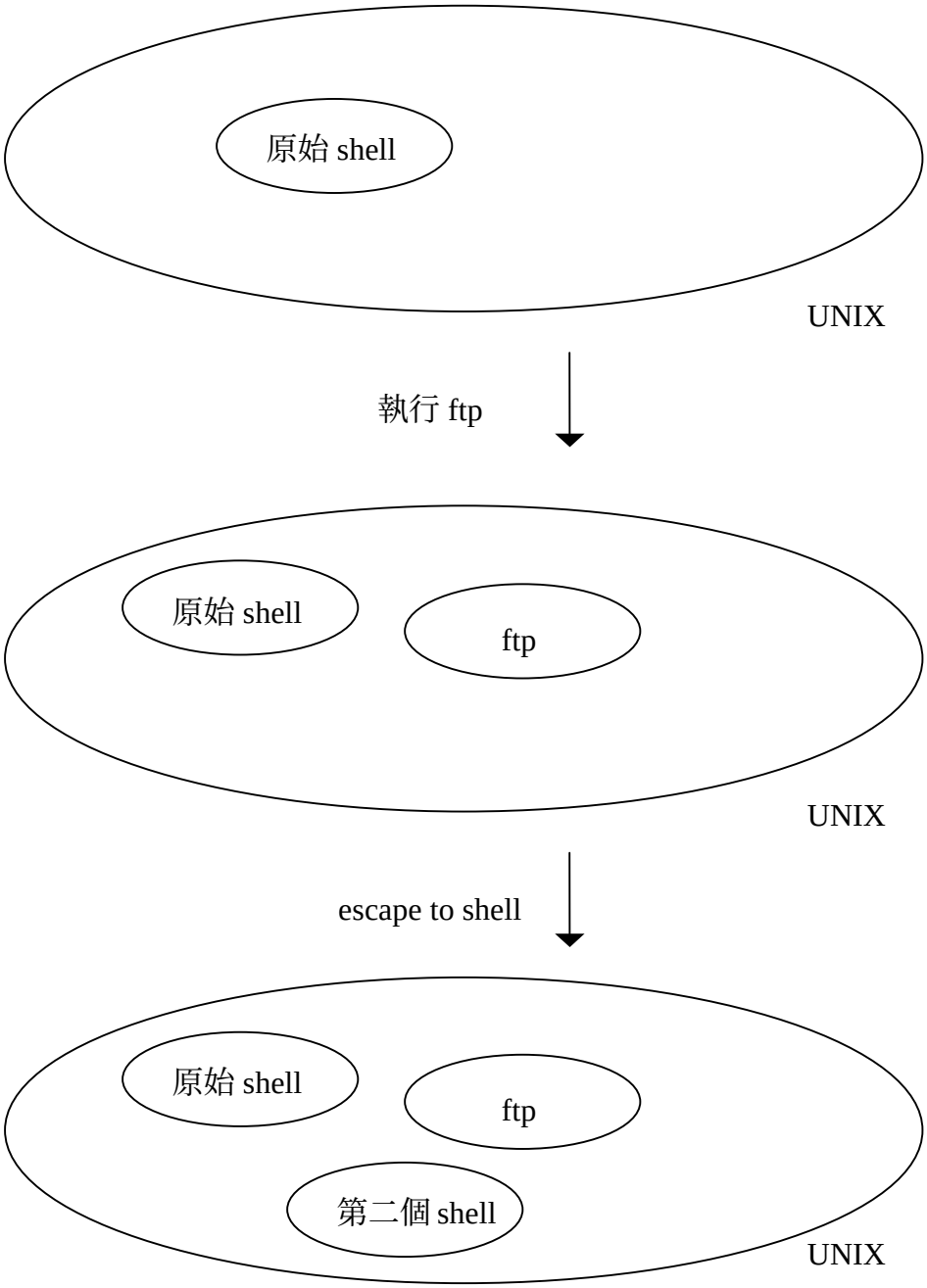
這話初聽有點奇怪但一解釋就不怪了。第一、任何 process 都可以 create process，所以 ftp 自然也可以 create process。第二、任何程式都必須在一定環境下才能執行，這個環境就是一個 process，shell 也是一個程式，要執行 shell 也不例外，需要在一個 process 裡才能執行。第三、ftp 其實大可不用擔心如果使用者要做的事不是 shell 而是其他的指令，因為一旦 shell 跑起來，任何其他的指令都可以透過 shell 來執行。

從一個程式中跳出來執行一個 shell 叫做 escape to shell，在 ftp 裡 escape to shell 的指令是一個驚嘆號（！）。

下面是在 csa500 上從 ftp escape to shell 的使用例子：

```
csa500> ftp
ftp> !
csa500>  （在此可下任何 UNIX 指令，再者 command prompt 也許會不同）
csa500> …
csa500> exit
ftp>    （跳出去做出來的 shell 結束，回到 ftp）
ftp> quit
csa500>  （ftp 結束，回到最早的 shell）
```

這裡用一張圖可以清楚的顯示出事情發生的狀況：

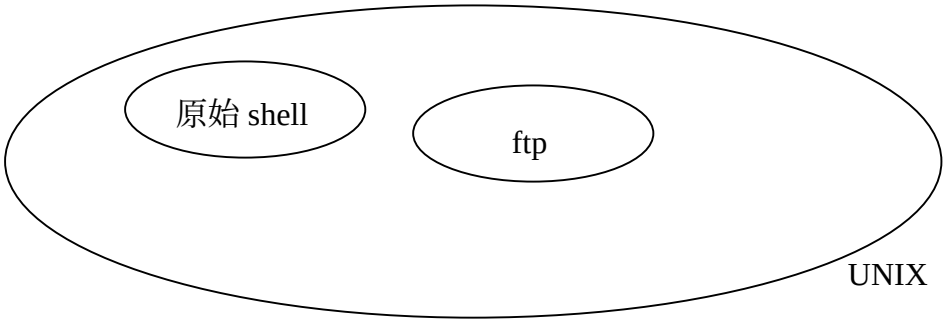


要驗証這張圖很簡單，在第二個 shell 的 command prompt 裡打人個叫 ps 的指令，這個指令告訴你系統裡現在有那些和你有關的 processes。ps 本身不算的話，你會看到三個 processes，原始 shell，ftp，及第二個 shell。另外藉機複習一下以前講的 foreground and background，當第二個 shell 在執行的時候，它在 foreground，原始的 shell 以及 ftp 都在幕後睡覺。

為了讓事情更有意思，看看下面的情況：

```
csa500> ftp
ftp> ^Z
csa500>
```

看起來和上一例沒什麼兩樣，是不是？但是如果此時做個 ps，你會發現只有兩個相關的 processes，一個 shell，一個 ftp，如下圖。



這次你並沒有 escape to shell，也就是說你沒有再多做一個 process 出來執行 shell。你用的是 job control 裡的^Z，把現行的工作（ftp）暫停，把在休息的原始 shell 叫出來為你服務。而你要回 ftp 也不能像前例一樣用 exit，這裡你怎麼出來的就要用相對的方式回去，用 fg 指令。

Anonymous FTP Site

很早時提過 ftp 是極佔 Internet 頻寬的一項服務。前面上自已有帳號的情況其實反而是少數，大多數的 ftp 都是去自已沒有帳號的 server。這種 ftp server 經常是以供任何 Internet 上的使用者免費下載檔案為唯一目的而存在的主機，這種主機通常稱之為 anonymous ftp site，而這種服務叫做 anonymous ftp。Anonymous ftp server 依舊需要 login，但是用的帳號是 anonymous（anonymous 就是匿名的意思）很多學生記不得 anonymous 的拼法，這裡教你們一招，帳號用 ftp 也可以通。（不過你還是得知道 anonymous ftp 是怎麼一回事。使用 anonymous ftp 時習慣上在密碼部份打入你的 email address，雖然亂打或不打也可以，但這不是 Internet 上的禮節。Email address 可以讓 server manager 知道誰來過他的站。