**Linux 的開機流程分析**

開機流程一覽



開機的過程中，開機管理程式(Boot Loader)使用的軟體可能不一樣，例如目前各大Linux distributions的主流為grub，但早期Linux預設是使用 LILO。開機管理程式可以指定使用哪個核心來開機，並實際載入核心到記憶體當中解壓縮與執行，此時核心就能夠開始在記憶體內活動，並偵測所有硬體資訊與載入適當的驅動程式來使整部主機開始運作，等到核心偵測硬體與載入驅動程式完畢後，一個最陽春的作業系統就開始在PC上跑了。主機系統開始運作後，Linux才會呼叫外部程式開始準備軟體執行的環境，並且實際的載入所有系統運作所需要的軟體程式！最後系統就會開始等待你的登入與操作！簡單來說，系統開機的經過可以彙整成底下的流程的：

1. [載入BIOS的硬體資訊與進行自我測試，並依據設定取得第一個可開機的裝置；](http://linux.vbird.org/linux_basic/0510osloader.php#process_1)
2. [讀取並執行第一個開機裝置內MBR的boot Loader(亦即是grub, spfdisk等程式)；](http://linux.vbird.org/linux_basic/0510osloader.php#process_2)
3. [依據boot loader的設定載入Kernel，Kernel會開始偵測硬體與載入驅動程式；](http://linux.vbird.org/linux_basic/0510osloader.php#process_3)
4. [在硬體驅動成功後，Kernel會主動呼叫init程式，而init會取得run-level資訊；](http://linux.vbird.org/linux_basic/0510osloader.php#startup_init)
5. [init執行/etc/rc.d/rc.sysinit檔案來準備軟體執行的作業環境(如網路、時區等)；](http://linux.vbird.org/linux_basic/0510osloader.php#startup_sysinit)
6. [init執行run-level的各個服務之啟動(script方式)；](http://linux.vbird.org/linux_basic/0510osloader.php#startup_runlevel)
7. [init執行/etc/rc.d/rc.local檔案；](http://linux.vbird.org/linux_basic/0510osloader.php#startup_local)
8. [init執行終端機模擬程式mingetty來啟動login程式，等待使用者登入；](http://linux.vbird.org/linux_basic/0510osloader.php#startup_term)

BIOS, boot loader 與 kernel 載入



* Boot Loader 的功能

Loader最主要功能是認識作業系統的檔案格式並據以載入核心到主記憶體中去執行。boot loader的功能就是『載入kernel檔案』！

* 載入核心偵測硬體與initrd的功能

當藉由boot loader的管理而開始讀取核心檔案後，接下來Linux就會將核心解壓縮到主記憶體當中，並利用核心的功能，開始測試與驅動各周邊裝置。Linux核心會自己重新偵測一次硬體，不一定會使用BIOS偵測到的硬體資訊！核心檔案一般來說會被放置到/boot裡面，並取名為/boot/vmlinuz！Linux核心可透過動態載入核心模組的(驅動程式)，核心模組放置在/lib/modules/目錄內。由於模組放置到磁碟根目錄內(要記得/lib不可以與/分別放在不同的partition)，因此開機的過程中核心必須要掛載根目錄，才能夠讀取核心模組提供載入驅動程式的功能。一般來說，非必要的功能且可以編譯成為模組的核心功能，目前的Linux distributions都會將他編譯成為模組。

核心不認識SATA磁碟，所以要載入SATA磁碟的驅動程式，否則無法掛載根目錄。但SATA的驅動程式在/lib/modules內，無法掛載根目錄又怎麼讀取到/lib/modules/內的驅動程式？可透過虛擬檔案系統來處理。虛擬檔案系統(Initial RAM Disk)一般使用的檔名為/boot/initrd，透過boot loader載入到記憶體中，然後這檔案會被解壓縮並且在記憶體當中模擬成一個根目錄，此模擬在記憶體當中的檔案系統能夠提供一支可執行的程式，透過該程式來載入開機過程中所最需要的核心模組，通常是USB, RAID, LVM, SCSI等檔案系統與磁碟介面的驅動程式！載入完後會幫助核心重新呼叫/sbin/init開始後續的正常開機流程。boot loader可載入kernel與initrd，然後在記憶體中讓initrd解壓縮成為根目錄，kernel能夠藉此載入適當的驅動程式，最終釋放虛擬檔案系統，並掛載實際的根目錄檔案系統，就能夠開始後續的正常開機流程。

|  |
| --- |
| 例題：是否沒有initrd就無法順利開機？  答：不見得的！需要initrd最重要的原因是，當開機時無法掛載根目錄的情況下，此時就一定需要 initrd，例如根目錄在特殊的磁碟介面(USB, SATA, SCSI)，或是檔案系統較為特殊(LVM, RAID)等，才會需要 initrd。如果Linux是安裝在IDE介面的磁碟上，並且使用預設的ext2/ext3 檔案系統，那麼不需要initrd也能順利的開機進入Linux！ |

在核心完整的載入後，您的主機應該就開始正確的運作了，接下來就是要開始執行系統的第一支程式：/sbin/init。

第一支程式 init 及設定檔 /etc/inittab 與 runlevel



在核心載入完畢、進行完硬體偵測與驅動程式載入後，主機硬體應該已經準備就緒了，此時核心會主動的呼叫第一支程式/sbin/init。init的PID號碼是一號。/sbin/init 最主要的功能就是準備軟體執行的環境，包括系統的主機名稱、網路設定、語系處理、檔案系統格式及其他服務的啟動等。而所有的動作都會透過init的設定檔，亦即是/etc/inittab來規劃，而 inittab內還有一個很重要的設定項目，那就是預設的 runlevel (開機執行等級)！

* Run level：執行等級有哪些？

Linux就是藉由設定run level來規定系統使用不同的服務來啟動，讓Linux的使用環境不同。基本上，依據有無網路與有無X Window而將run level分為7個等級，分別是：

* 0 - halt (系統直接關機)
* 1 - single user mode (單人維護模式，用在系統出問題時的維護)
* 2 - Multi-user, without NFS (類似底下的 runlevel 3，但無 NFS 服務)
* 3 - Full multi-user mode (完整含有網路功能的純文字模式)
* 4 - unused (系統保留功能)
* 5 - X11 (與 runlevel 3 類似，但加載使用 X Window)
* 6 - reboot (重新開機)

由於run level 0, 4, 6不是關機、重新開機就是系統保留的，所以：『不能將預設的 run level 設定為這三個值』，則系統就會不斷的自動關機或自動重新開機....開機時到底是如何取得系統的run level的？當然是 /etc/inittab 所設定的囉！那麼/etc/inittab 到底有什麼資訊呢？我們先來看看這個檔案的內容好了：

* /etc/inittab 的內容與語法

|  |
| --- |
| [root@www ~]# vim /etc/inittab  id:5:initdefault: <==預設的 runlevel 設定, 此 runlevel 為 5  si::sysinit:/etc/rc.d/rc.sysinit <==準備系統軟體執行的環境的腳本執行檔  # 7 個不同 run level 的，需要啟動的服務的 scripts 放置路徑：  l0:0:wait:/etc/rc.d/rc 0 <==runlevel 0 在 /etc/rc.d/rc0.d/  l1:1:wait:/etc/rc.d/rc 1 <==runlevel 1 在 /etc/rc.d/rc1.d/  l2:2:wait:/etc/rc.d/rc 2 <==runlevel 2 在 /etc/rc.d/rc2.d/  l3:3:wait:/etc/rc.d/rc 3 <==runlevel 3 在 /etc/rc.d/rc3.d/  l4:4:wait:/etc/rc.d/rc 4 <==runlevel 4 在 /etc/rc.d/rc4.d/  l5:5:wait:/etc/rc.d/rc 5 <==runlevel 5 在 /etc/rc.d/rc5.d/  l6:6:wait:/etc/rc.d/rc 6 <==runlevel 6 在 /etc/rc.d/rc6.d/  # 是否允許按下 [ctrl]+[alt]+[del] 就重新開機的設定項目：  ca::ctrlaltdel:/sbin/shutdown -t3 -r now  # 底下兩個設定則是關於不斷電系統的 (UPS)，一個是沒電力時的關機，一個是復電的處理  pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"  pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"  1:2345:respawn:/sbin/mingetty tty1 <==其實 tty1~tty6 是由底下這六行決定的。  2:2345:respawn:/sbin/mingetty tty2  3:2345:respawn:/sbin/mingetty tty3  4:2345:respawn:/sbin/mingetty tty4  5:2345:respawn:/sbin/mingetty tty5  6:2345:respawn:/sbin/mingetty tty6  x:5:respawn:/etc/X11/prefdm -nodaemon <==X window 則是這行決定的！ |

解析一下這個檔案吧！首先，這個檔案的語法是利用冒號 (:) 將設定分隔成為四個欄位，每個欄位的意義與說明如下：

|  |
| --- |
| [設定項目]:[run level]:[init 的動作行為]:[指令項目] |

1. 設定項目：最多四個字元，代表 init的主要工作項目，只是一個簡單的代表說明。
2. run level：該項目在哪些run level底下進行的意思。如果是 35 則代表 runlevel 3與5都會執行。
3. init 的動作項目：主要可以進行的動作項目意義有：

|  |  |
| --- | --- |
| inittab 設定值 | 意義說明 |
| initdefault | 代表預設的run level設定值 |
| sysinit | 代表系統初始化的動作項目 |
| ctrlaltdel | 代表 [ctrl]+[alt]+[del] 三個按鍵是否可以重新開機的設定 |
| wait | 代表後面欄位設定的指令項目必須要執行完畢才能繼續底下其他的動作 |
| respawn | 代表後面欄位的指令可以無限制的再生(重新啟動)。舉例來說， tty1 的 mingetty 產生的可登入畫面，在你登出而結束後，系統會再開一個新的可登入畫面等待下一個登入。 |

* init 的處理流程

/etc/inittab的設定也有點類似shell script，因為該檔案內容的設定也是一行一行的從上往下處理的，因此inittab設定的處理流程會是：

1. 先取得 runlevel 亦即預設執行等級的相關等級 (以鳥哥的測試機為例，為5號)；
2. 使用/etc/rc.d/rc.sysinit進行系統初始化
3. 由於runlevel是5，因此只進行『l5:5:wait:/etc/rc.d/rc 5』，其他行則略過
4. 設定好 [ctrl]+[alt]+[del] 這組的組合鍵功能；
5. 設定不斷電系統的 pf, pr 兩種機制；
6. 啟動 mingetty的六個終端機(tty1 ~ tty6)；
7. 最終以 /etc/X11/perfdm -nodaemon 啟動圖形介面啦！

現在你可以知道為啥 [ctrl]+[alt]+[del] 可以重新開機而我們預設提供6個虛擬終端機 (tty1~tty6) 給你使用了吧！由於整個設定都是依據 /etc/inittab 來決定的，因此如果你想要修改任何細節的話，可以這樣做喔：

* 如不想讓使用者用 [crtl]+[alt]+[del] 重啟系統，可將『ca::ctrlaltdel:/sbin/shutdown -t3 -r now』加上註解(#)取消設定
* 規定開機的預設run level是純文字的3號或者是具有圖形介面的5號，可經由『id:5:initdefault:』那個數字來決定！以這個檔案為例，我是使用預設的圖形介面。如果想要關閉圖形介面，將5改成3即可。
* 如果不想啟動六個終端機 (tty1~tty6)，那麼可以將『 6:2345:respawn:/sbin/mingetty tty6』關閉數個。但至少啟動一個喔！

一般是3或5來作為預設的run level。但有時後可能需要進入run level 1，也就是單人維護模式的環境。run level 1有點像Windows系統當中的『安全模式』，專門用來處理當系統有問題時的操作環境。

init 處理系統初始化流程 (/etc/rc.d/rc.sysinit)



/etc/inittab裡『si::sysinit:/etc/rc.d/rc.sysinit』表示：開始載入各項系統服務之前，得先做好整個系統環境，利用/etc/rc.d/rc.sysinit這個shell script來設定好系統環境。』開機的過程當中進行了什麼動作，得分析/etc/rc.d/rc.sysinit。他主要的工作大抵有這幾項：

1. 取得網路環境與主機類型：網路設定檔/etc/sysconfig/network，取得主機名稱與預設通訊閘(gateway)等網路環境。
2. 測試與掛載記憶體裝置/proc及USB裝置/sys：除掛載記憶體裝置/proc之外，還會主動偵測系統上是否具有usb的裝置，若有則會主動載入usb的驅動程式，並且嘗試掛載 usb的檔案系統。
3. 設定終端機 (console) 字形：
4. 設定顯示於開機過程中的歡迎畫面 (text banner)；
5. 設定系統時間 (clock) 與時區設定：需讀入 /etc/sysconfig/clock 設定值
6. 周邊設備的偵測與Plug and Play(PnP)參數的測試：根據核心在開機時偵測結果(/proc/sys/kernel/modprobe )開始進行ide/scsi /網路/音效等周邊設備的偵測，以及利用以載入的核心模組進行 PnP 裝置的參數測試。
7. 使用者自訂模組載入：使用者可以在/etc/sysconfig/modules/\*.modules加入自訂模組，此時會被載入到系統當中
8. 載入核心的相關設定：統會主動去讀取 /etc/sysctl.conf 這個檔案的設定值，使核心功能成為我們想要的樣子。
9. 設定主機名稱與初始化電源管理模組 (ACPI)
10. 初始化 LVM 的檔案系統功能
11. 以 fsck 檢驗磁碟檔案系統：會進行 filesystem check
12. 重新以可讀寫模式掛載系統磁碟：
13. 清除開機過程當中的暫存檔案：
14. 將開機相關資訊載入 /var/log/dmesg 檔案中。

/etc/rc.d/rc.sysinit將基本的系統設定資料都寫好了，也將系統的資料設定完整！dmesg可知道開機的過程中發生什麼事情。這檔案進行的很多工作的預設設定檔，都在/etc/sysconfig/中！經過/etc/rc.d/rc.sysinit處理後，主機系統等著系統相關的服務與網路服務啟動！

啟動系統服務與相關啟動設定檔 (/etc/rc.d/rc N & /etc/sysconfig)



載入核心讓整個系統準備接受指令來工作，經過/etc/rc.d/rc.sysinit的系統模組與相關硬體資訊的初始化後，還要啟動系統所需要的各項服務，/etc/inittab裡的run level設定值可決定啟動的服務項目。不同的run level服務啟動的各個shell script 在/etc/inittab 裡提到：

|  |
| --- |
| l0:0:wait:/etc/rc.d/rc 0  l1:1:wait:/etc/rc.d/rc 1  l2:2:wait:/etc/rc.d/rc 2  l3:3:wait:/etc/rc.d/rc 3  l4:4:wait:/etc/rc.d/rc 4  l5:5:wait:/etc/rc.d/rc 5 <==本例中，以此項目來解釋  l6:6:wait:/etc/rc.d/rc 6 |

上面提到的就是各個run level要執行的各項腳本放置處！透過/etc/rc.d/rc來處理相關任務！/etc/rc.d/rc 5的意義是這樣的：

* 透過外部第一號參數 ($1)來取得想要執行的腳本目錄。亦即由/etc/rc.d/rc **5**可取得/etc/rc**5**.d/這目錄來處理相關的腳本程式；
* 找到 /etc/rc5.d/K??\* 開頭的檔案，並進行『 /etc/rc5.d/K??\* stop』的動作；
* 找到 /etc/rc5.d/S??\* 開頭的檔案，並進行『 /etc/rc5.d/S??\* start』的動作；

透過上面的說明我們可以知道所有的項目都與 /etc/rc5.d/有關，那麼我們就來瞧瞧這個目錄下有些什麼玩意兒吧！

|  |
| --- |
| [root@www ~]# ll /etc/rc5.d/  lrwxrwxrwx 1 root root 16 Sep 4 2008 K02dhcdbd -> ../init.d/dhcdbd  ....(中間省略)....  lrwxrwxrwx 1 root root 14 Sep 4 2008 K91capi -> ../init.d/capi  lrwxrwxrwx 1 root root 23 Sep 4 2008 S00microcode\_ctl -> ../init.d/microcode\_ctl  lrwxrwxrwx 1 root root 22 Sep 4 2008 S02lvm2-monitor -> ../init.d/lvm2-monitor  ....(中間省略)....  lrwxrwxrwx 1 root root 17 Sep 4 2008 S10network -> ../init.d/network  ....(中間省略)....  lrwxrwxrwx 1 root root 11 Sep 4 2008 S99local -> ../rc.local  lrwxrwxrwx 1 root root 16 Sep 4 2008 S99smartd -> ../init.d/smartd  ....(底下省略).... |

服務是以『/etc/init.d/服務檔名 {start,stop}』來啟動與關閉的，那麼/etc/rc5.d/[SK]xx 其實就是跑到/etc/init.d/去找到相對應的服務腳本，然後分別進行start (Sxx)或 stop (Kxx)的動作！舉例來說，表格內的K91capi及S10network，透過/etc/rc.d/rc 5的執行，這兩個檔案會這樣進行：

* /etc/rc5.d/K91capi stop --> /etc/init.d/capi stop
* /etc/rc5.d/S10network start --> /etc/init.d/network start

你有想要啟動該runlevel時就執行的服務，那麼利用 Sxx 並指向 /etc/init.d/ 的特定服務啟動腳本後，該服務就能夠在開機時啟動啦！那麼為什麼K與S後面要有數字呢？因為各不同的服務其實還是互有關係的。舉例來說，如果要啟動WWW服務，總是得要有網路吧？所以 /etc/init.d/network 就會比較先被啟動啦！那麼您就會知道在S或者是K後面接的數字是啥意思了吧？那就是執行的順序啦！

使用者自訂開機啟動程序 (/etc/rc.d/rc.local)



在完成預設runlevel指定的各項服務的啟動後，如果還有其他的動作想要完成，舉例來說，還想要寄一封mail給某個系統管理帳號，通知他系統剛剛重新開機完畢，那麼是否應該要製作一個shell script放置在 /etc/init.d/ 裡面，然後再以連結方式連結到 /etc/rc5.d/裡面呢？/etc/rc.d/rc.local這個檔案就可以執行您自己想要執行的系統指令了。也就是說，有任何想要在開機時就進行的工作時，直接將他寫入 /etc/rc.d/rc.local，那麼該工作就會在開機的時候自動被載入喔！一般來說，自己製作的 shell script 完整檔名寫入 /etc/rc.d/rc.local ，如此一來，開機就會將我的shell script執行過！

根據/etc/inittab之設定，載入終端機或 X-Window 介面



在完成系統所有服務的啟動後，Linux 會啟動終端機或者是X Window來等待使用者登入！實際參考的項目是/etc/inittab的這一段：

|  |
| --- |
| 1:2345:respawn:/sbin/mingetty tty1  2:2345:respawn:/sbin/mingetty tty2  3:2345:respawn:/sbin/mingetty tty3  4:2345:respawn:/sbin/mingetty tty4  5:2345:respawn:/sbin/mingetty tty5  6:2345:respawn:/sbin/mingetty tty6  x:5:respawn:/etc/X11/prefdm –nodaemon |

在run level 2, 3, 4, 5時，都會執行/sbin/mingetty，而且執行六個，這也是為何Linux 會提供『六個純文字終端機』的設定！因為 mingetty就是在啟動終端機的指令說。respawn的init動作項目，代表『當後面的指令被終止(terminal)時，init會主動的重新啟動該項目。』這也是為何登入tty1終端機介面後，以exit離開後， 系統還是會重新顯示等待使用者輸入的畫面的原因啊！例如不想要tty5與tty6，就將那兩行註解，則下次重新開機後，您的Linux 就只剩下『 F1 ~ F4』有效而已！如果使用的是run level 5呢？那麼除了這六個終端機之外，init還會執行/etc/X11/prefdm -nodaemon那個指令喔！他主要的功能就是在啟動 X Window！

核心與核心模組



模組化可想成是一個『外掛程式』，可能由硬體開發廠商提供，也可能我核心本來就支援～不過，較新的硬體，通常都需要硬體開發商提供驅動程式模組！那麼核心與核心模組放在哪？

* 核心：/boot/vmlinuz或/boot/vmlinuz-version；
* 核心解壓縮所需RAM Disk：/boot/initrd (/boot/initrd-version)；
* 核心模組：/lib/modules/version/kernel或/lib/modules/$(uname -r)/kernel；
* 核心原始碼：/usr/src/linux 或/usr/src/kernels/ (要安裝才會有，預設不安裝)

如果該核心被順利的載入系統當中了，那麼就會有幾個資訊紀錄下來：

* 核心版本： /proc/version
* 系統核心功能： /proc/sys/kernel

問題來啦，如果我有個新的硬體，偏偏我的作業系統不支援，該怎麼辦？很簡單啊！

* 重新編譯核心，並加入最新的硬體驅動程式原始碼；
* 將該硬體的驅動程式編譯成為模組，在開機時載入該模組

核心模組與相依性



核心模組的放置處是在/lib/modules/$(uname -r)/kernel當中，裡面主要還分成幾個目錄：

|  |
| --- |
| arch ：與硬體平台有關的項目，例如 CPU 的等級等等；  crypto ：核心所支援的加密的技術，例如 md5 或者是 des 等等；  drivers ：一些硬體的驅動程式，例如顯示卡、網路卡、PCI 相關硬體等等；  fs ：核心所支援的 filesystems ，例如 vfat, reiserfs, nfs 等等；  lib ：一些函式庫；  net ：與網路有關的各項協定資料，還有防火牆模組 (net/ipv4/netfilter/\*) 等等；  sound ：與音效有關的各項模組； |

Linux會提供一些模組相依性的解決方案！/lib/modules/$(uname -r)/modules.dep記錄了在核心支援的模組的各項相依性。那麼這個檔案如何建立呢？利用depmod這個指令就可以達到建立該檔案的需求了！

|  |
| --- |
| [root@www ~]# depmod [-Ane]  選項與參數：  -A ：不加任何參數時， depmod 會主動的去分析目前核心的模組，並且重新寫入  /lib/modules/$(uname -r)/modules.dep 當中。若加入 -A 參數時，則 depmod  會去搜尋比 modules.dep 內還要新的模組，如果真找到新模組，才會更新。  -n ：不寫入 modules.dep ，而是將結果輸出到螢幕上(standard out)；  -e ：顯示出目前已載入的不可執行的模組名稱  範例一：若我做好一個網路卡驅動程式，檔名為 a.ko，該如何更新核心相依性？  [root@www ~]# cp a.ko /lib/modules/$(uname -r)/kernel/drivers/net  [root@www ~]# depmod |

Linux kernel 2.6.x版的核心模組副檔名一定是.ko結尾的，當使用depmod之後，該程式會跑到模組標準放置目錄/lib/modules/$(uname -r)/kernel，並依據相關目錄的定義將全部的模組捉出來分析，最終才將分析的結果寫入modules.dep檔案中！

核心模組的觀察



那你到底曉不曉得目前核心載入了多少的模組呢？粉簡單啦！利用 lsmod 即可！

|  |
| --- |
| [root@www ~]# lsmod  Module Size Used by  autofs4 24517 2  hidp 23105 2  ....(中間省略)....  8139too 28737 0  8139cp 26305 0  mii 9409 2 8139too,8139cp <==mii 還被 8139cp, 8139too 使用  ....(中間省略)....  uhci\_hcd 25421 0 <==底下三個是 USB 相關的模組！  ohci\_hcd 23261 0  ehci\_hcd 33357 0 |

使用 lsmod 之後，系統會顯示出目前已經存在於核心當中的模組，顯示的內容包括有：

* 模組名稱(Module)；
* 模組的大小(size)；
* 此模組是否被其他模組所使用 (Used by)。

模組有相依性！舉上表為例，mii模組會被8139too使用。就是『當要載入8139too時，需先載入mii模組才可以順利的載入8139too』。可以查閱每個模組的資訊嗎？舉例來說，我們知道8139too是螃蟹卡的驅動程式，那麼mii是什麼？就用 modinfo 來觀察吧！

|  |
| --- |
| [root@www ~]# modinfo [-adln] [module\_name|filename]  選項與參數：  -a ：僅列出作者名稱；  -d ：僅列出該 modules 的說明 (description)；  -l ：僅列出授權 (license)；  -n ：僅列出該模組的詳細路徑。  範例一：由上個表格當中，請列出 mii 這個模組的相關資訊：  [root@www ~]# modinfo mii  filename: /lib/modules/2.6.18-92.el5/kernel/drivers/net/mii.ko  license: GPL  description: MII hardware support library  author: Jeff Garzik <jgarzik@pobox.com>  srcversion: 16DCEDEE4B5629C222C352D  depends:  vermagic: 2.6.18-92.el5 SMP mod\_unload 686 REGPARM 4KSTACKS gcc-4.1  # 可以看到這個模組的來源，以及該模組的簡易說明！(是硬體支援函式庫)  範例二：我有一個模組名稱為 a.ko ，請問該模組的資訊為？  [root@www ~]# modinfo a.ko  ....(省略).... |

modinfo除了可查閱在核心內的模組，還可檢查某個模組檔案，因此如果想知道某檔案代表的意義為何，利用modinfo加上完整檔名！

核心模組的載入與移除



自行手動載入模組，使用modprobe指令來載入模組，因為modprobe會主動去搜尋modules.dep的內容，克服模組的相依性後，才決定載入的模組有哪些，至於insmod完全由使用者自行載入完整檔名的模組，並不會主動的析模組相依性！

|  |
| --- |
| [root@www ~]# insmod [/full/path/module\_name] [parameters]  範例一：請嘗試載入 cifs.ko 這個『檔案系統』模組  [root@www ~]# insmod /lib/modules/$(uname -r)/kernel/fs/cifs/cifs.ko  [root@www ~]# lsmod | grep cifs  cifs 212789 0 |

他立刻就將該模組載入囉～但是 insmod 後面接的模組必須要是完整的『檔名』才行！那如何移除這個模組呢？

|  |
| --- |
| [root@www ~]# rmmod [-fw] module\_name  選項與參數：  -f ：強制將該模組移除掉，不論是否正被使用；  -w ：若該模組正被使用，則 rmmod 會等待該模組被使用完畢後，才移除他！  範例一：將剛剛載入的 cifs 模組移除！  [root@www ~]# rmmod cifs  範例二：請載入 vfat 這個『檔案系統』模組  [root@www ~]# insmod /lib/modules/$(uname -r)/kernel/fs/vfat/vfat.ko  insmod: error inserting '/lib/modules/2.6.18-92.el5/kernel/fs/vfat/vfat.ko':  -1 Unknown symbol in module  # 無法載入 vfat 這個模組啊！傷腦筋！ |

使用insmod與rmmod的問題就是，你必須要自行找到模組的完整檔名才行，萬一模組有相依屬性的問題時，你將無法直接載入或移除該模組呢！所以近年來我們都建議直接使用 modprobe 來處理模組載入的問題，這個指令的用法是：

|  |
| --- |
| [root@www ~]# modprobe [-lcfr] module\_name  選項與參數：  -c ：列出目前系統所有的模組！(更詳細的代號對應表)  -l ：列出目前在 /lib/modules/`uname -r`/kernel 當中的所有模組完整檔名；  -f ：強制載入該模組；  -r ：類似 rmmod ，就是移除某個模組囉～  範例一：載入 cifs 模組  [root@www ~]# modprobe cifs  # 很方便吧！不需要知道完整的模組檔名，這是因為該完整檔名已經記錄到  # /lib/modules/`uname -r`/modules.dep 當中的緣故啊！如果要移除的話：  [root@www ~]# modprobe -r cifs |

modprobe是直接去搜尋modules.dep的紀錄，所以可以克服模組的相依性問題，而且還不需要知道該模組的詳細路徑呢！

|  |  |
| --- | --- |
| 例題：嘗試使用modprobe 載入vfat這個模組，並且觀察該模組的相關模組是哪個？  答：我們使用 modprobe 來載入，再以lsmod來觀察與grep擷取關鍵字看看：   |  | | --- | | [root@www ~]# modprobe vfat  [root@www ~]# lsmod | grep vfat  vfat 15809 0  fat 51165 1 vfat <==原來就是 fat 這個模組啊！  [root@www ~]# modprobe -r vfat <==測試完移除此模組 | |

核心模組的額外參數設定： /etc/modprobe.conf



如果修改某些模組的額外參數設定，就在這個檔案內設定吧！假設我的網路卡eth0是使用ne，但是eth1同樣也使用ne，為了避免同一個模組會導致網路卡的錯亂，可以先找到eth0與eth1的I/O與IRQ，假設：

* eth0 ： I/O (0x300) 且 IRQ=5
* eth1 ： I/O (0x320) 且 IRQ=7

則：

|  |
| --- |
| [root@www ~]# vi /etc/modprobe.conf  alias eth0 ne  alias eth1 ne  options eth0 io=0x300 irq=5  options eth1 io=0x320 irq=7 |

嘿嘿！如此一來，我的 Linux 就不會捉錯網路卡的對應囉！因為被我強制指定某個 I/O 咯嘛！