LINUX 作業系統實務
05. Install/update Software

2020 TKU

Sherry Yin

Richard Mathew Stallman

- 生於1953年， 網路上自稱的ID為RMS. 1971年的時候，進入駭客圈中相當出名的人工智慧實驗室(AI Lab.)
- 1984年，史托曼開始GNU(GNU not Unix)計畫， 這個計畫的目的是：建立一個自由、開放的Unix作業系統(Free Unix)。
- 開始撰寫C語言的編譯器，那就是現在相當有名的GNU C Compiler(gcc)
- 將他原先就已經寫過的Emacs編輯器寫成可以在Unix上面跑的軟體，並公布原始碼。
- 史托曼便藉著Emacs以磁帶(tape)出售，成立自由軟體基金會(FSF, Free Software Foundation) 。終於在1990年左右完成了GCC以及可以被使用來操作作業系統的基本介面BASH shell
- 1985年，為了避免GNU所開發的自由軟體被其他人所利用而成為專利軟體， 他與律師草擬了有名的通用公共許可證(General Public License, GPL) 。

GNU所開發的幾個重要軟體

- Emacs
- GNU C (GCC)
- GNU C Library (glibc)
- Bash shell
- Stallman先生認為，如果使用方撰寫程式的能力比自己強，那麼當對方修改完自己的程式並且回傳修改後的程式碼給自己，那自己的程式撰寫功力無形中就更往上爬

Assignment I

- 既然已經有團隊, 請按照姓名首字母順序, 從第一個人開始, 把自己寫的比較得意的一段code發給下一個人, 請他給出意見或者修改, 然後給下一個. 所有人的修改跟意見彙總成一個檔案, Submit到iclass或MS teams.
- 如果沒有團隊的怎麼辦? – 請在標題註明請老師看

伺服器有甚麼用途

- 自己拥有一台服务器可以做哪些很酷的事情？ - 小游的回答 - 知乎 https://www.zhihu.com/question/40854395/answer/728324567
- Home Server PC Build: https://www.youtube.com/watch?v=AWJEHfV-pmA
- How to install software on Linux: https://www.youtube.com/watch?v=19O5kFdtKb0

Red Hat

- Red Hat Linux1.0版本於1994年11月3日發行。
- 以桌面用戶為目標的Mandrake Linux，Yellow Dog Linux和ASPLinux
- 2004年4月30日，Red Hat公司正式停止對Red Hat 9.0版本的支援，標誌著Red Hat Linux的正式完結。
- 原本的電腦版Red Hat Linux發行套件則與來自民間的Fedora計劃合併，成為Fedora Core發行版本。
- Red Hat Linux中的RPM軟體包格式可以說是Linux社區的一個事實標準，被廣泛使用於其他Linux發行套件中。

SuSE

- 源自德國。現時SUSE屬於Novell旗下的業務
- SuSE於1992年末創辦，目的是成為UNIX技術公司，專門製為德國人推出量身訂作的SLS/Slackware軟體及UNIX/Linux說明文件。
- "S.u.S.E."後來改稱/簡短為"SuSE"，意思為"Software-und System-Entwicklung"，那是一句德文，英文為"Software and system development"。

Ubuntu

- Ubuntu由Canonical公司發布，是目前最多使用者的Linux版本，用戶數超過10億人(含伺服器、手機與其分支版本)。
- Ubuntu在Ubuntu 12.04的發布頁面上使用了「友幫拓」作為官方譯名。之前一些中文使用者曾使用班圖、烏班圖、烏斑兔、烏幫圖、笨兔等作為非官方譯名。

Ubuntu新版發布週期

- https://zh.wikipedia.org/wiki/Ubuntu

Fedora

- 由Fedora專案社群開發、紅帽公司贊助
- 軟體包管理主要由yum實用程式提供，在22版後則由dnf取代。
- Fedora同樣提供圖形介面（例如pirut，pup和puplet）
- apt-rpm是yum的替代品，對於Debian類發行版的用戶來說可能更熟悉。

Debian

- Debian計劃最初由伊恩·默多克於1993年發起，Debian 0.01版在1993年9月15日發布
- Debian帶來了數萬個軟體包。為了方便用戶使用，這些軟體包都已經被編譯包裝為一種方便的格式，開發人員把它叫做deb包。
- Debian系統中，軟體包管理可由多種工具協同運作進行，範圍從最底層的dpkg命令直到圖形介面的Synaptic工具。
- 推薦在Debian系統中管理軟體包的標準工具是apt工具集。

Linux發行版列表

- https://zh.wikipedia.org/wiki/Linux%E5%8F%91%E8%A1%8C%E7%89%88%E5%88%97%E8%A1%A8
- Linux Mint是一種基於Ubuntu/Debian的Linux作業系統
- Linux Mint可以幾乎與Ubuntu軟體倉庫完全相容，採用apt管理和.deb軟體包。

Distribution 分類

	RPM 軟體管理	DPKG 軟體管理	其他未分類
商業公司	RHEL (Red Hat 公司) SuSE (Micro Focus)	Ubuntu (Canonical Ltd.)	
社群單位	Fedora CentOS OpenSuSE	Debian B2D	Gentoo

distribution 代表	軟體管理機制	使用指令	線上升級機制(指令)
Red Hat/Fedora	RPM	rpm, rpmbuild	YUM (yum)
Debian/Ubuntu	DPKG	dpkg	APT (apt-get)

Install on Mint

- You can head over to the Linux Mint Community website to search and browse software.
- Other options:
  1. If you prefer to use the Terminal, run the command sudo apt-get install packageName to install the specified package directly.
  2. To install .deb packages, run sudo dpkg -i filename.deb.
  3. To install .rpm packages, run sudo rpm -i filename.rpm.

Install steam on Mint

https://www.omgubuntu.co.uk/2016/06/install-steam-on-ubuntu-16-04-lts

1. sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys B05498B7
2. sudo sh -c 'echo "deb http://repo.steampowered.com/steam/ precise steam" >>
3. /etc/apt/sources.list.d/steam.list'
4. sudo apt-get update
5. sudo apt-get install steam
6. Afterwards start it with

   steam

Install Steam on CentOs

http://linuxsysconfig.com/how-to-install-steam-on-centos-7/

1. yum install http://dl.fedoraproject.org/pub/epel/beta/7/x86_64/epel-release-7-0.2.noarch.rpm
2. yum install http://pkgs.repoforge.org/rpmforge-release/rpmforge-release-0.5.3-1.el7.rf.x86_64.rpm
3. cat /etc/yum.repos.d/steam_fedora19.repo

   [steam_fedora19]

   name=Steam RPM packages (and dependencies) for Fedora

   baseurl=http://negativo17.org/repos/steam/fedora-19/x86_64/

   enabled=0

   skip_if_unavailable=1

   gpgcheck=0
4. yum install http://download1.rpmfusion.org/free/fedora/releases/19/Everything/i386/os/libtxc_dxtn-1.0.0-3.fc19.i686.rpm
5. yum --enablerepo=steam_fedora19 install steam

Install Steam on Fedora

- https://unix.stackexchange.com/questions/523196/how-can-i-install-steam-on-fedora-28
- sudo dnf install -y fedora-workstation-repositories
- sudo dnf install -y steam --enablerepo=rpmfusion-nonfree-steam

Yum（Yellow dog Updater, Modified）

- 由Duke University團隊修改Yellow Dog Linux的Yellow Dog Updater開發而成，是一個基於RPM包管理的字元前端軟體包管理器。
- 能夠從指定的伺服器自動下載RPM包並且安裝，可以處理依賴性關係，並且一次安裝所有依賴的軟體包，無須繁瑣地一次次下載、安裝。
- 被Yellow Dog Linux本身，以及Fedora、Red Hat Enterprise Linux採用。

DNF

- 全稱Dandified Yum，是RPM發行版的軟體包管理器Yellowdog Updater, Modified（yum）的下一代版本。
- DNF最早出現在Fedora 18中，並在Fedora 22、RHEL8中替代yum。

APT

- 進階打包工具（英語：Advanced Packaging Tools，縮寫為APT）是Debian及其衍生的Linux軟體包管理器。
- APT可以自動下載，組態，安裝二進位或者原始碼格式的軟體包，因此簡化了Unix系統上管理軟體的過程。
- APT最早被設計成dpkg的前端，用來處理deb格式的軟體包。現在經過APT-RPM組織修改，APT已經可以安裝在支援RPM的系統管理RPM套件。
- APT由以下的幾個主要的命令構成：
  - apt-get
  - apt-cache
  - apt-file

APT 範例

- 搜尋 apt-cache search <package> 這樣系統會列出與<package>名稱相匹配的套件。
- 安裝 apt-get install <package> 這樣系統會自動下載<package>以及所有的依存套件，同時進行套件的安裝。
- 移除 apt-get remove [--purge] <package> 這樣系統會自動移除<package>以及任何依此存套件的其它套件。 --purge指明套件應被完全清除。
- 升級 apt-get update 這樣系統會自動由對映點更新套件列表，如果想安裝最新套件，必須先運行一次。每次修改了 /etc/apt/sources.list後，也必須執行一次。
  - apt-get upgrade [-u] 這樣系統會自動將所有已經安裝在系統內的套件升級為最新版本。如果一個套件改變了依存關係，而需要安裝一個新的套件時，它將不會被升級，而是標識成hold。如果某個套件被設定hold標號，就不會被升級。
  - apt-get dist-upgrade [-u] 和apt-get upgrade類似，dist-upgrade會安裝和移除套件來滿足依存關係，因此具有一定的危險性。

pip

- pip是一個以Python電腦程式語言寫成的軟體包管理系統，他可以安裝和管理軟體包，另外不少的軟體包也可以在「Python軟體包索引」
- pip的其中一個主要特點就是其方便使用的命令列介面，這讓使用者可以透過以下的一句文字命令來輕易地安裝Python軟體包：

  pip install some-package-name
- 此外，使用者也可以輕易地透過以下的命令來移除軟體包：

  pip uninstall some-package-name
- pip也擁有一個透過「需求」檔案來管理軟體包和其相應版本數目的完整列表之功能，這容許一個完整軟體包組合可以在另一個環境（如另一部電腦）或虛擬化環境中進行有效率的重新創造。這個功能可以透過一個已正確進行格式化的文字檔案和以下的命令來完成：

  pip install -r requirements.txt

Linux 系統上的可執行檔

- 看有沒有可執行的那個權限 (具有 x permission)
- Linux 系統上真正認識的可執行檔其實是二進位檔案 (binary program)，例如 /usr/bin/passwd, /bin/touch 這些個檔案。
- shell scripts 只是利用 shell (例如 bash) 這支程式的功能進行一些判斷式，而最終執行的除了 bash 提供的功能外，是呼叫一些已經編譯好的二進位程式來執行

- 怎麼知道一個檔案是否為 binary 呢 - file

  # 先以系統的檔案測試看看：

  [root@study ~]# file /bin/bash

  /bin/bash: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked

  (uses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=0x7e60e35005254...stripped

  # 如果是系統提供的 /etc/init.d/network

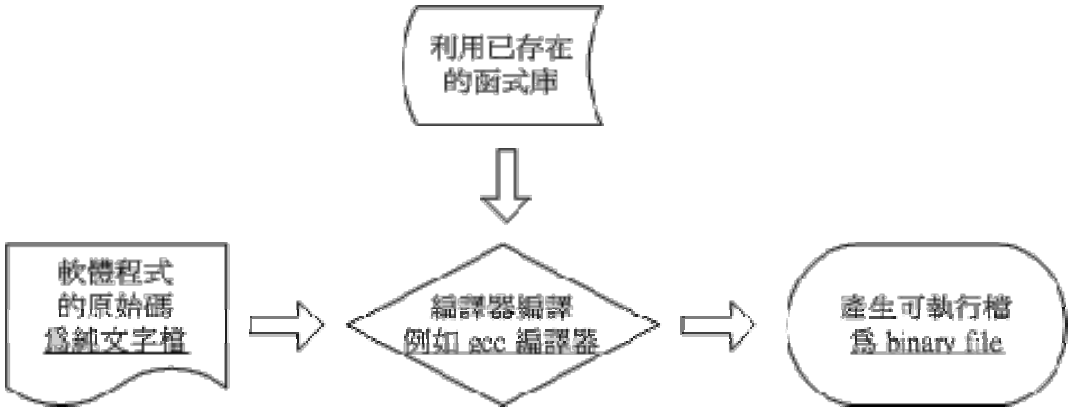
  [root@study ~]# file /etc/init.d/network

  /etc/init.d/network: Bourne-Again shell script, ASCII text executable

- 如果是 binary 而且是可以執行的時候，他就會顯示執行檔類別 (ELF 64-bit LSB executable)，同時會說明是否使用動態函式庫 (shared libs)，而如果是一般的 script ，那他就會顯示出 text executables 之類的字樣

如何做出binary program

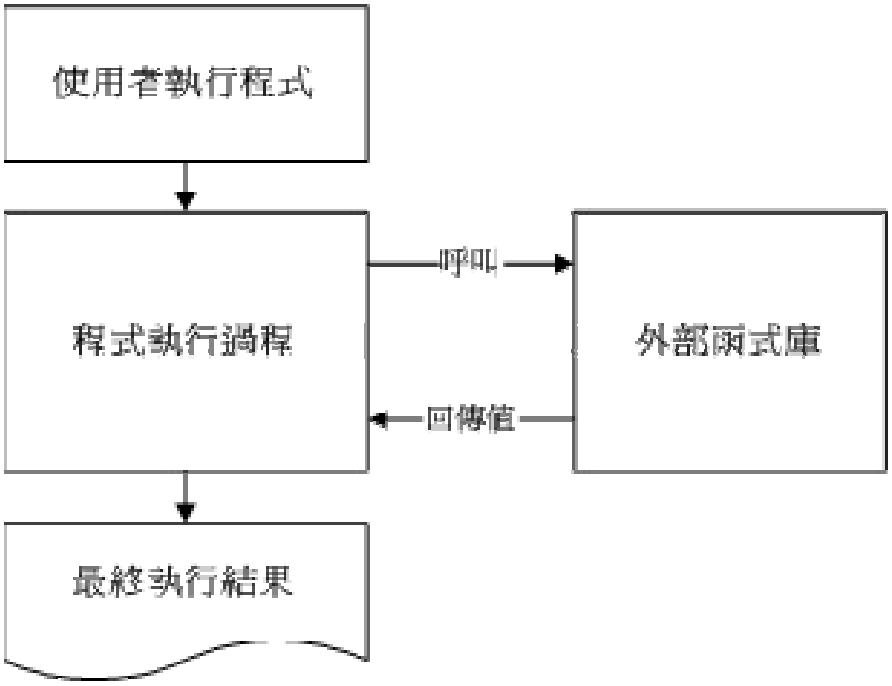
- 使用 vim 來進行程式的撰寫，寫完的程式就是所謂的原始程式碼
- 再來就是要將這個檔案『編譯』成為作業系統看的懂得 binary program
- 而要編譯自然就需要『編譯器』來動作， 經過編譯器的編譯與連結之後，就會產生一支可以執行的 binary program 。



目標檔 (Object file)

- 在編譯的過程當中還會產生所謂的目標檔 (Object file)，這些檔案是以 *.o 的副檔名樣式存在的
- C 語言的原始碼檔案通常以 *.c 作為副檔名。
- 有的時候，會在程式當中『引用、呼叫』 其他的外部副程式，或者是利用其他軟體提供的『函數功能』，這個時候，就必須要在編譯的過程當中， 將該函式庫給他加進去，如此一來，編譯器就可以將所有的程式碼與函式庫作一個連結 (Link) 以產生正確的執行檔。

函式庫



- 分為動態與靜態函式庫
  - 靜態: 副檔名：(副檔名為 .a) 這類的函式庫通常副檔名為 libxxx.a 的類型；在編譯的時候會直接整合到執行程式當中
  - 動態: 副檔名：(副檔名為 .so) 這類函式庫通常副檔名為 libxxx.so 的類型；動態函式庫在編譯的時候，在程式裡面只有一個『指向 (Pointer)』的位置
- Linux 的核心提供很多的核心相關函式庫與外部參數， 這些核心功能在設計硬體的驅動程式的時候是相當有用的資訊，這些核心相關資訊大多放置在 /usr/include, /usr/lib, /usr/lib64 裡面
- 判斷某個可執行的 binary 檔案含有什麼動態函式庫可用ldd

例如我想要知道 /usr/bin/passwd 這個程式含有的動態函式庫有哪些

[root@study ~]# ldd [-vdr] [filename]　#選項與參數： -v ：列出所有內容資訊； -d ：重新將資料有遺失的 link 點秀出來　-r ：將 ELF 有關的錯誤內容秀出來

ldd範例一

範例一：找出 /usr/bin/passwd 這個檔案的函式庫資料

[root@study ~]# ldd /usr/bin/passwd

....(前面省略)....

libpam.so.0 => /lib64/libpam.so.0 (0x00007f5e683dd000) <==PAM 模組

libpam_misc.so.0 => /lib64/libpam_misc.so.0 (0x00007f5e681d8000)

libaudit.so.1 => /lib64/libaudit.so.1 (0x00007f5e67fb1000) <==SELinux

libselinux.so.1 => /lib64/libselinux.so.1 (0x00007f5e67d8c000) <==SELinux

我們前言的部分不是一直提到 passwd 有使用到 pam 的模組

利用 ldd 察看一下這個檔案，看到 libpam.so 這就是 pam 提供的函式庫

ldd範例二

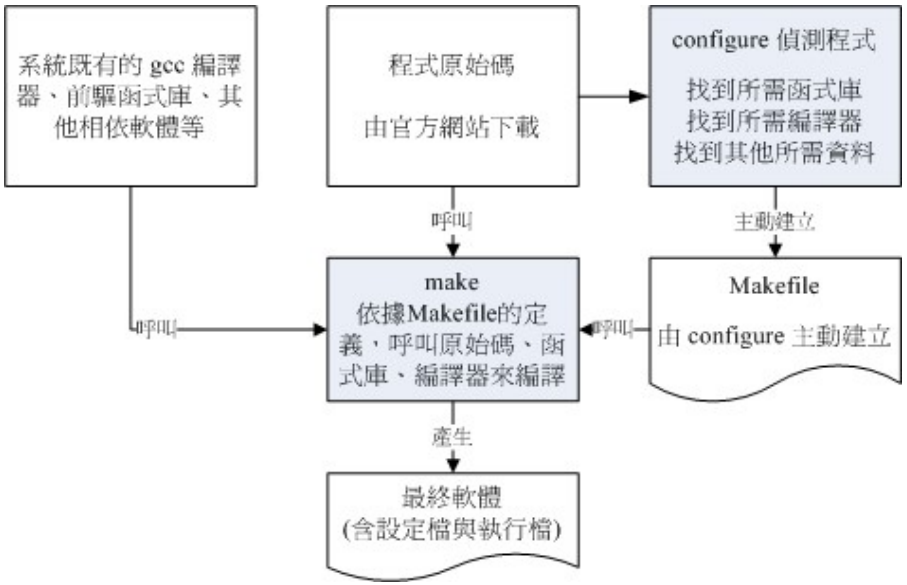
範例二：找出 /lib64/libc.so.6 這個函式的相關其他函式庫

```
[root@study ~]# ldd -v /lib64/libc.so.6
        /lib64/ld-linux-x86-64.so.2 (0x00007f7acc68f000)
        linux-vdso.so.1 =>  (0x00007fffa975b000)

        Version information:  <==使用 -v 選項，增加顯示其他版本資訊
        /lib64/libc.so.6:
                ld-linux-x86-64.so.2 (GLIBC_2.3) => /lib64/ld-linux-x86-64.so.2
                ld-linux-x86-64.so.2 (GLIBC_PRIVATE) => /lib64/ld-linux-x86-64.so.2
```

make

- 當執行 make 時，make 會在當時的目錄下搜尋 Makefile (or makefile) 這個文字檔，而 Makefile 裡面則記錄了原始碼如何編譯的詳細資訊



偵測程式

- 軟體開發商都會寫一支偵測程式來偵測使用者的作業環境， 以及該作業環境是否有軟體開發商所需要的其他功能
- 該偵測程式偵測完畢後，就會主動的建立這個 Makefile 的規則檔案
- 這支偵測程式的檔名為 configure 或者是 config 。
- 偵測程式會偵測的資料大約有底下這些：
  - 是否有適合的編譯器可以編譯本軟體的程式碼；
  - 是否已經存在本軟體所需要的函式庫，或其他需要的相依軟體；
  - 作業系統平台是否適合本軟體，包括 Linux 的核心版本；
  - 核心的表頭定義檔 (header include) 是否存在 (驅動程式必須要的偵測)。

Tarball

- Tarball 檔案，就是將軟體的所有原始碼檔案先以 tar 打包，然後再以壓縮技術來壓縮，通常最常見的就是以 gzip 來壓縮了。因為利用了 tar 與 gzip 的功能，所以 tarball 檔案一般的副檔名就會寫成 *.tar.gz 或者是簡寫為 *.tgz
  - tar xvzf file.tar.gz -C /path/to/somedirectory
- 近來由於 bzip2 與 xz 的壓縮率較佳，所以 Tarball 漸漸的以 bzip2 及 xz 的壓縮技術來取代 gzip 囉！因此檔名也會變成 *.tar.bz2, *.tar.xz 之類。
  - tar xvjf file.tar.bz2

Tarball內容

- 所以說， Tarball 是一個軟體包， 妳將他解壓縮之後，裡面的檔案通常就會有：
  - 原始程式碼檔案；
  - 偵測程式檔案 (可能是 configure 或 config 等檔名)；
  - 本軟體的簡易說明與安裝說明 (INSTALL 或 README)。

軟體升級兩種方式

- 直接以原始碼透過編譯來安裝與升級 – tarball
- 直接以編譯好的 binary program 來安裝與升級 - 這個預先編譯好程式的機制存在於很多 distribution 喔，包括
  - Red Hat 系統 (含 Fedora/CentOS 系列) 發展的 RPM 軟體管理機制與 yum 線上更新模式；
  - Debian 使用的 dpkg 軟體管理機制與 APT 線上更新模式等等。

Tarball 如何安裝

- 一個軟體的 Tarball 是如何安裝的
  - 將 Tarball 由廠商的網頁下載下來；
  - 將 Tarball 解開，產生很多的原始碼檔案；
  - 開始以 gcc 進行原始碼的編譯 (會產生目標檔 object files)；
  - 然後以 gcc 進行函式庫、主、副程式的連結，以形成主要的 binary file；
  - 將上述的 binary file 以及相關的設定檔安裝至自己的主機上面。

安裝gcc和make

- CentOs: yum groupinstall "Development Tools“
- Mint/Ubuntu:
  - $ sudo apt-get update
  - $ sudo apt-get upgrade
  - $ sudo apt-get install build-essential
  - $ gcc -v
  - $ make -v

C: helloWorld

- [root@study ~]# vim hello.c <==用 C 語言寫的程式副檔名建議用 .c

```
#include <stdio.h>
int main(void)
{
        printf("Hello World\n");
}
```

編譯與測試執行

[root@study ~]# gcc hello.c

[root@study ~]# ll hello.c a.out <== ll 即 ls -l

-rwxr-xr-x. 1 root root 8503 Sep 4 11:33 a.out   <==此時會產生這個檔名

-rw-r--r--. 1 root root   71 Sep 4 11:32 hello.c

[root@study ~]# ./a.out

Hello World

- 在預設的狀態下，如果我們直接以 gcc 編譯原始碼，並且沒有加上任何參數，則執行檔的檔名會被自動設定為 a.out
- 直接執行 ./a.out 這個執行檔
- hello.c 就是原始碼，而 gcc 就是編譯器， a.out 就是編譯成功的可執行 binary program

客製執行檔的檔名

[root@study ~]# gcc -c hello.c

[root@study ~]# ll hello*

-rw-r--r--. 1 root root 71 Sep 4 11:32 hello.c

-rw-r--r--. 1 root root 1496 Sep 4 11:34 hello.o <==就是被產生的目標檔

[root@study ~]# gcc -o hello hello.o

[root@study ~]# ll hello*

-rwxr-xr-x. 1 root root 8503 Sep 4 11:35 hello <==這就是可執行檔！ -o 的結果

-rw-r--r--. 1 root root 71 Sep 4 11:32 hello.c

-rw-r--r--. 1 root root 1496 Sep 4 11:34 hello.o

[root@study ~]# ./hello

Hello World

- 這個步驟主要是利用 hello.o 這個目標檔製作出一個名為 hello 的執行檔，詳細的 gcc 語法我們會在後續章節中繼續介紹！透過這個動作後，我們可以得到 hello 及 hello.o 兩個檔案， 真正可以執行的是 hello 這個 binary program

主、副程式連結

1. 編輯主程式：

```
[root@study ~]# vim thanks.c
#include <stdio.h>
int main(void)
{
        printf("Hello World\n");
        thanks_2();
}
```

上面的 thanks_2(); 那一行就是呼叫副程式啦！

```
[root@study ~]# vim thanks_2.c
#include <stdio.h>
void thanks_2(void)
{
        printf("Thank you!\n");
}
```

進行程式的編譯與連結 (Link)

2. 開始將原始碼編譯成為可執行的 binary file ：

```
[root@study ~]# gcc -c thanks.c thanks_2.c
[root@study ~]# ll thanks*
-rw-r--r--. 1 root root   75 Sep  4 11:43 thanks_2.c
-rw-r--r--. 1 root root 1496 Sep  4 11:43 thanks_2.o
-rw-r--r--. 1 root root   91 Sep  4 11:42 thanks.c
-rw-r--r--. 1 root root 1560 Sep  4 11:43 thanks.o
```

```
[root@study ~]# gcc -o thanks thanks.o thanks_2.o
[root@study ~]# ll thanks*
-rwxr-xr-x. 1 root root 8572 Sep  4 11:44 thanks
```

3. 執行一下這個檔案：

```
[root@study ~]# ./thanks
Hello World
Thank you!
```

為什麼要製作出目標檔

- 由於我們的原始碼檔案有時並非僅只有一個檔案，所以我們無法直接進行編譯。 這個時候就需要先產生目標檔，然後再以連結製作成為 binary 可執行檔。
- 如果有一天，你更新了 thanks_2.c 這個檔案的內容，你只要重新編譯 thanks_2.c 來產生新的 thanks_2.o ，然後再以連結製作出新的 binary 可執行檔即可！而不必重新編譯其他沒有更動過的原始碼檔案。 這對於軟體開發者來說，是一個很重要的功能
- 更多內容: http://linux.vbird.org/linux_basic/0520source_code_and_tarball.php

make 工具

```
# 1. 先編輯 makefile 這個規則檔，內容只要作出 main 這個執行檔
[root@study ~]# vim makefile
main: main.o haha.o sin_value.o cos_value.o
        gcc -o main main.o haha.o sin_value.o cos_value.o –lm          # 注意：第二行的 gcc 之前是 <tab> 按鍵產生的空格喔！
# 2. 嘗試使用 makefile 制訂的規則進行編譯的行為：
[root@study ~]# rm -f main *.o   <==先將之前的目標檔去除
[root@study ~]# make
cc    -c -o main.o main.c
cc    -c -o haha.o haha.c
cc    -c -o sin_value.o sin_value.c
cc    -c -o cos_value.o cos_value.c
gcc -o main main.o haha.o sin_value.o cos_value.o –lm  # 此時 make 會去讀取 makefile 的內容，並根據內容直接去給他編譯相關的檔案囉！
# 3. 在不刪除任何檔案的情況下，重新執行一次編譯的動作：
[root@study ~]# make
make: `main' is up to date.    # 只會進行更新 (update) 的動作。
```

Tarball安裝的基本步驟

1. 取得原始檔：將 tarball 檔案在 /usr/local/src 目錄下解壓縮；
2. 取得步驟流程：進入新建立的目錄底下，去查閱 INSTALL 與 README 等相關檔案內容 (很重要的步驟！)；
3. 相依屬性軟體安裝：根據 INSTALL/README 的內容察看並安裝好一些相依的軟體 (非必要)；
4. 建立 makefile：以自動偵測程式 (configure 或 config) 偵測作業環境，並建立 Makefile 這個檔案；
5. 編譯：以 make 這個程式並使用該目錄下的 Makefile 做為他的參數設定檔，來進行 make (編譯或其他) 的動作；
6. 安裝：以 make 這個程式，並以 Makefile 這個參數設定檔，依據 install 這個標的 (target) 的指定來安裝到正確的路徑！

Tarball 指令下達方式

./configure

這個步驟就是在建立 Makefile 這個檔案, 通常程式開發者會寫一支 scripts 來檢查你的 Linux 系統、相關的軟體屬性等等，這個步驟的相關資訊應該要參考一下該目錄下的 README 或 INSTALL 相關的檔案

make clean

make 會讀取 Makefile 中關於 clean 的工作。他可以去除目標檔案！因為誰也不確定原始碼裡面到底有沒有包含上次編譯過的目標檔案 (*.o) 存在，所以當然還是清除一下比較妥當的。 至少等一下新編譯出來的執行檔我們可以確定是使用自己的機器所編譯完成的！

make

make 會依據 Makefile 當中的預設工作進行編譯的行為, 主要是進行 gcc 來將原始碼編譯成為可以被執行的 object files ，但是這些 object files 通常還需要一些函式庫之類的 link 後，才能產生一個完整的執行檔！使用 make 就是要將原始碼編譯成為可以被執行的可執行檔，而這個可執行檔會放置在目前所在的目錄之下， 尚未被安裝到預定安裝的目錄中

make install

通常這就是最後的安裝步驟了，make 會依據 Makefile 這個檔案裡面關於 install 的項目，將上一個步驟所編譯完成的資料給他安裝到預定的目錄中，就完成安裝了

Tarball 的管理

1. 最好將 tarball 的原始資料解壓縮到 /usr/local/src 當中；
2. 安裝時，最好安裝到 /usr/local 這個預設路徑下；
3. 考慮未來的反安裝步驟，最好可以將每個軟體單獨的安裝在 /usr/local 底下；
4. 為安裝到單獨目錄的軟體之 man page 加入 man path 搜尋：
5. 如果你安裝的軟體放置到 /usr/local/software/ ，那麼 man page 搜尋的設定中，可能就得要在 /etc/man_db.conf 內的 40~50 行左右處，寫入如下的一行：

   MANPATH_MAP /usr/local/software/bin /usr/local/software/man

   這樣才可以使用 man 來查詢該軟體的線上文件囉！

更多rpm相關

- 參考教科書: Your UNIX/LINUX The Ultimate Guide
- http://linux.vbird.org/linux_basic/0520rpm_and_srpm.php