在 Linux 下建立軟體套件

作者:*Mendel Leo Cooper* <mailto:thegrendel@theriver.com> *http://personal.riverusers.com/~thegrendel/*
譯者:*J.S.Lin* <mailto:r6921068@ms.cc.ntu.edu.tw>
v1.52, 27 December 1997

這是份廣泛指引文件,可用來建立"一般性" UNIX 軟體發行套件在 Linux 下.

目錄

1 簡介

很多軟體套件提供給各式 UNIX, 包括 Linux, 是將原始碼檔案壓縮成檔案(archives)而發行的. 相同的套件可以建立以便在不同對象的機器上執行, 並且這可節省些軟體成品而免於必須製造各種發行版本. 軟體套件的單一發行版本結果能執行在各式實體(incarnations), 像是在 Intel 機器, DEC Alpha, RISC 工作站, 甚至是電算主機. 但不幸的是, 這將"建立"軟體的工作推給了最終使用者(end user), 實際上是"系統

管理者", 即坐在鍵盤前的同志...你. 雖然這樣, 認眞來說這過程並不是像它看起來那麼可怕或不可思議, 就像這指引所解說一樣.

2 從取得開始

你有下載或其它方式取得一個軟體套件. 最常見情況是它被打包起來 (*tarred*) 而且被壓縮 (*gzipped*), 成爲 `.tar.gz` 或 `.tgz` 形式. 首先將它複製到一工作目錄. 然後用 *untar* 和 *gunzip* 處理它. 處理它最適合的命令是 **tar xzvf *filename***, 這裏的 *filename* 當然是軟體檔名. 這種解開程序常常會安裝適當檔案在它所開啓的子目錄中. 注意的是如果套件名稱有 *.Z* 結尾, 那將要使用 **uncompress PACKAGENAME**, 然後再用 **tar xvf PACKAGENAME** 而不要用之前的步驟.

有時打包起來的檔案必須 *untarred* 且安裝自使用者的家目錄(home directory), 或者也許是在某個其它目錄, 像是套件設定資料所指定的. 假若你設法要 *untar* 它而得到的是錯誤訊息, 也許就是那原因. 讀一下套件文件, 特別是 `README` 和/或 `Install` 檔案, 如果存在, 編輯設定檔案和/或 `Makefiles` 如所需要的, 即與安裝指示符合. 注意的是通常 不 去更改 `Imake` 檔案, 因爲這可能有未預期的結果. 某些軟體套件允許自動安裝程序,它是藉由執行 **make install** 來放置二元碼在適當系統位置.

偶而, 你可能需要使用 `patch`檔案,該檔案有列出原本的與新的原始檔案的差異, 來升級或加入臭蟲(bug)的修正至未打包(unarchived)的原始檔案. doc檔案與/或 `README` 檔案會告知你這算不算該情況. 對於使用 Larry Wall 的強力 修補(`patch`) 工具之正常的語法(syntax)是 **patch < patchfile**.

你現在可以開始進行 建立 過程階段了.

3 使用 Make

`Makefile` 是建立過程的重點. 在它的最簡單的形式, Makefile 就是個 script ,它是用來編譯或建立"二元碼",套件的可執行部分. Makefile也可以提供軟體套件升級的工具,而不需要重新編譯每一個在套件中的原始檔案, 但其有不同的情況(或是不同的約定).

某些特點是, Makefile 開啓 `cc` 或 `gcc`. 這實際上是個前處理器, C (or C++) 編譯器, 和 連結器, 按照那個順序. 這過程轉換原始碼(source)爲可眞正去執行的二元碼(binaries).

下指令 *make* 通常只要鍵入 **make**. 這通常建立討論中的套件所有需要的可執行檔. 然而, *make* 也做其他工作,像是安裝檔案至合適的目錄 (**make install**) 而且刪除舊的 object 檔案 (**make clean**). 執行 **make -n** 允許預覽建立過程, 就是列出所有 make 所喚起的(triggered)所有命令, 但沒有眞的去執行它們.

只有在最簡單的軟體使用一般性的 Makefile. 較複雜的安裝需要根據函式庫(libraries), include 檔案以及你個別機器資源所在目錄來量身定製. 這特別是當需要 `X11` libraries 來安裝建立時. *Imake* 和 *xmkmf* 可完成這樣的工作.

引用自 man page,Imakefile 是個"模板(template)"的 Makefile. imake 工具參考 Imakefile 根據你的系統造出合適的 Makefile. 然而,幾乎在所有的情況下,你要執行 xmkmf, 它是要下 imake 指令用的 shell script, 也就是個前端介面(front end). 察閱一下內附在軟體檔案間的 README 或 INSTALL 檔案以取得特別的指示. 至於更詳細的步驟分析,請閱讀 imake 和 xmkmf 的 man pages.

要知道的是 *xmkmf* 和 *make* 可能需要以 root 身分來下指令, 特別是當 **make install** 來搬移二元碼到 `/usr/bin` 或 `/usr/local/bin` 目錄. 以一般使用者的身分而不具 root 權力 來使用 make 將可能會導致 `write access denied` 的錯誤訊息, 因爲你缺少寫入(write)至系統目錄的權限. 也檢查一下, 所造出來的二元碼對你與任何其他適用的使用者有適合的執行(execute)權限.

下指令 **xmkmf** 來使用 *Imake* 檔案, 建立一個新的而且適合你系統的 Makefile. 正常地下指令 **xmkmf** 都會附加 **-a** 引數, 因而自動地做 *make Makefiles, make includes,* 和 *make depend* 的動作. 這會設定變數(variables)和給定函式庫(library) 位置給編譯器和連結器. 有時會沒有 *Imake* 檔案, 引而代之的是有 *INSTALL* 和 *configure* 的 script, 也會到這樣的目的. 注意的是如果你要執行 *configure*, 那要下指令 **./configure** 以確保在目前目錄下正確的 script *configure* 是被呼叫. 在大部分的情況, 在發行套件中的 *README* 檔案會說明安裝步驟.

通常一個好的方法是從外觀檢視一下 `Makefile` 是 `xmkmf` 或 其中一個安裝的 scripts 所造出來的. Makefile 正常下會爲你的系統而修正, 但你偶而可能需要"旋扭(tweak)"或手動地修正錯誤.

你的一般安裝步驟將會是:

- 讀一下 *README* 檔案和其它合用的文件.
- 執行 **xmkmf -a**, 還是 *INSTALL* 或 *configure* script.
- 檢查一下 `Makefile`.
- 如果需要, 執行一下 **make clean**, **make Makefiles**, **make includes**, and **make depend**.
- 執行 **make**.
- 檢查一下權限.
- 如果需要, 執行一下 **make install**.

4 克服困難

如果用 **xmkmf** 和/或 **make** 很成功沒有錯誤, 你可以著手於 5 (next section). 然而, 在眞實生活, 很少事情在第一次就正確的工作. 這就是當你的機智豐富時,要勇於嘗試.

4.1 連結錯誤

- 如果 **make** 失敗而出現有 `Link error: -lX11: No such file or directory`, 甚至是在使用 **xmkmf** 後. 這可能意味著 *Imake* 檔案沒有適當地裝好. 檢查 *Makefile* 的第一部分有沒有像這樣的幾行:

  ```
  LIB=          -L/usr/X11/lib
  INCLUDE=      -I/usr/X11/include/X11
  LIBS=         -lX11 -lc -lm
  ```

  `-L` 和 `-I` 切換器(switches)會通告編譯器和連結器分別去哪裏尋找 *library* 和 *include* 檔案. 在這例子, *X11 libraries* 應該是在 `/usr/X11/lib` 目錄, 而 *X11 include* 檔案 應該是在 `/usr/X11/include/X11`目錄. 如果這在你的機器上是不對的, 必須改變 *Makefile* 而且再試試 **make**.

- 在非常少的情況, 以 *root* 身分跑一下 **ldconfig** 也許是個答案:

  \# /**etc**/**ldconfig -n** /**lib** 將會更新共享式函式庫符號連結 (shared library symbolic links). 這在正常的情形下不是需要的.

- 目前另一件事是如果 **xmkmf** 失敗, 則試試下列 script:

```
make -DUseInstalled -I/usr/X386/lib/X11/config
```

- 有時原始碼需要使用舊版的 X11R5 libraries 來建立. 如果你有 R5 libs 在 /usr/X11R6/lib (當你首先在安裝 Linux 時, 你是有選擇是否安裝它們的), 然後你只需要確定,你有連結到軟體所需要建立的地方. R5 libs 是有 `libX11.so.3.1.0`, `libXaw.so.3.1.0`, 和 `libXt.so.3.1.0`. 一般你需要連結至, 像是 **libX11.so.3 -> libX11.so.3.1.0**. 大概軟體也需要這個形式連結 **libX11.so -> libX11.so.3.1.0**. 當然, 要將這"斷掉(missing)"的連結建好, 要以 *root* 身分使用這指令 **ln -s libX11.so.3.1.0 libX11.so**.

- 某些套件會要求你去安裝一個或以上的升級版本. 舉例而言, StarDivision GmbH 的 *StarOffice* 組件(suite)是惡名昭彰(notorious)的要求 `libc` version 5.4.4 或更新. 以 *root*身分, 你需要複製一個或以上的函式庫(libraries)至合適的目錄, 刪去舊的函式庫(libraries),然後重設符號連結.

  小心: 在這要格外注意, 因爲如果你弄糟, 你可能會付出你的系統不能運作的代價.

  你通常可以找到升級的函式庫(libraries)在 *Sunsite* `<ftp://sunsite.unc.edu>`.

4.2 其它問題

- 在安裝好的 *Perl* 或 shell script 給你一個 `No such file or directory` 錯誤訊息. 在這情況,檢查檔案權限來確定檔案可執行 以及檢查檔案標頭是否含有使用 shell 或程式, 指明 script 所在地方. 舉例而言, script 開始如:

  ```
  #!/usr/local/bin/perl
  ```

  如果 *Perl* 實際裝在你的 `/usr/bin` 目錄, 而不是 `/usr/local/bin` 目錄, 那麼 script 無法執行. 有兩個方法來修改. script 檔案標頭改成 `#!/usr/bin/perl`, 或用符號連結至正確的目錄, **ln -s /usr/bin/perl /usr/local/bin/perl**.

- 某些 X11 軟體要求有 `Motif` 函式庫(libraries)才能建立. 而標準的 Linux 發行套件並沒有安裝 Motif libraries, 而且目前 Motif 還要額外 $100-$200 的花費 (雖然免費軟體 *Lesstif* `<http://www.lesstif.org/>` 在某些場合也能夠用). 如果你需要 Motif 來建立某一套件, 但缺少 Motif libraries, 那麼也許可以取得 靜態連結的二元碼*(statically linked binaries)*. 靜態連結在二元碼本身納入函式庫行程(library routines). 這造成二元碼檔案大了許多, 但是碼可在缺少該函式庫的系統上執行.

- 執行 *configure* script 會創造奇怪的 `Makefile`, 那像是與你所要設法建立的套件毫不相干. 這表示跑著錯誤的 *configure*, 其找到你的 path 上某處其他的 configure. 所以要下 *configure* 指令 應該要下成 **./configure** 即目前目錄下的.

- 某些程式會要求有 *setuid root*, 爲了要以有 *root* 權力 來執行. 要達成這目錄的指令是 以*root* 下指令 **chmod u+s filename**. (注意的是這程式已經是 *root* 所擁有了). 這在設定檔案權限中的 *setuid* bit 很有用. 這樣用在當程式在存取系統硬體,像是 modem 或 CD ROM drive, 或當 SVGA libs 被使在於主控台模式(console mode), 像是在一個特別惡名昭彰的(notorious)模擬器(emulation)套件. 如果程式正被 root 執行, 但給一般使用者一個 *access denied* 的錯誤訊息, 想像就是這樣造成的.

  警告: 一個有 *setuid* 成 root 的程式可能提出對你的系統有安全上的風險. 這個程式可以 root 權力來跑而所以有某些破害的潛在性. 在設定 *setuid* bit 前, 確定你知道該程式在做什麼, 必要時看一下原始碼.

4.3 旋扭與微調

你可能希望檢查 `Makefile` 來確定有選用對你系統最佳編譯選項. 舉例來說, 設定 *-O2* 旗標來選最高層級的最佳化而 *-fomit-frame-pointer* 旗標來造最小二元碼 (雖然 debugging 會沒有打開). 除非你知道你在做什麼, 而且在任何情況下, 除非艱苦建立的工作已完成, 否則不要玩這些.

4.4 去哪取得進一步的協助

在我的經驗, 大概25%的應用程式建立"完全沒有困難". 另外大約50%能"被說服"的建立, 要付出努力的範圍從無聊到極為困難. 那仍然表示有某些套件無論如何都無法建立. 即使那樣, 那些 Intel ELF 和/或 `a.out` 二元碼, 也可能在 *Sunsite* `<ftp://sunsite.unc.edu>`, *TSX-11 archive* `<ftp://tsx-11.mit.edu>` 或其它地方找到. 也許, 軟體的創造者可以提供編譯好的二元碼給你的特別的機器使用.

注意的是如果你取得預先編譯好的二元碼, 你會需要檢查是否與你的系統相容:

- 這二元碼必須在你的 硬體上跑 (i.e., Intel x86).
- 這二元碼必須與你的核心(kernel)相容 (i.e., a.out 或 ELF).
- 你的函式庫(libraries)必須是最新的.

如果全都失敗, 你可以在合適的 新聞群組尋求幫忙, 像是在*comp.os.linux.x* 或 *comp.os.linux.development*. 萬一你只是不夠幸運的話, 嘿, 試試也是有趣的.

5 最後步驟

讀一下軟體套件的文件來決定是否有某些環境變數需要設 (在 *.bashrc* 或 *.cshrc*) 以及 是否 *.Xdefaults* 和 *.Xresources* 檔案需要調整.

可能有個應用程式內定的檔案, 通常叫做 Xfoo.ad 在原本的 Xfoo 發行套件. 如果這樣, 編輯 Xfoo.ad 檔案適合你的機器, 然後重新命名 (**mv**) 成 Xfoo 而且 以 *root* 身分 安裝它到 `/usr/lib/X11/app-defaults` 目錄, 沒這樣做可能會造成軟體行為很奇怪, 甚至拒絕執行.

大多數的軟體套件附有一個或以上的格式化 man pages. 以 *root* 身分, 複製 Xfoo.man 檔案到合適的 `/usr/man` 目錄 (`man1` - `man9`), 而根據那再重新命名. 舉例而言, 如果 Xfoo.man 結果是在 /usr/man/man4, 那應該命名為 Xfoo.4 (mv Xfoo.man Xfoo.4). 根據一般約定使用者的命令擺在 `man1`, 遊戲是在 `man6`, 而管理的套件是在 `man8` (看一下 *man* 文件 以取得更詳細資料).當然,在你系統你可以不照這個約定, 只要你喜歡.

某些套件不會安裝二元碼(binaries)在合適的系統目錄, 換句話說, 他們沒有 *install* 選項在 `Makefile` 中. 如果是這情況, 你可以用 root 身分複製二元碼(binaries)到 `usr/local/bin` 目錄來手動安裝二元碼(binaries).

注意的是某些或全部上述步驟,在大部分情形下,應該會由 **make install** 來自動操作. 如果是這樣, `README` 或 `INSTALL` 文件檔會提到這個.

6 第一個例子: Xscrabble

Matt Chapman的 **Xscrabble** 似乎像是個頗有趣的程式, 因爲我曾是個貪婪的 Scrabble™(拼字遊戲)玩家.我下載下,解壓, 而且建立它以 README 檔案中的下列步驟:

```
xmkmf
make Makefiles
make includes
make
```

當然它不能正常運作...

```
gcc -o xscrab -O2 -O -L/usr/X11R6/lib
init.o xinit.o misc.o moves.o cmove.o main.o xutils.o mess.o popup.o
widgets.o display.o user.o CircPerc.o
-lXaw -lXmu -lXExExt -lXext -lX11 -lXt -lSM -lICE -lXExExt -lXext -lX11
-lXpm -L../Xc -lXc

BarGraf.o(.text+0xe7): undefined reference to `XtAddConverter'
BarGraf.o(.text+0x29a): undefined reference to `XSetClipMask'
BarGraf.o(.text+0x2ff): undefined reference to `XSetClipRectangles'
BarGraf.o(.text+0x375): undefined reference to `XDrawString'
BarGraf.o(.text+0x3e7): undefined reference to `XDrawLine'
etc.
etc.
etc...
```

我在 *comp.os.linux.x* 的新聞群組詢問過, 而且有些人好心的指出似乎 Xt, Xaw, Xmu, 和 X11 libs 沒有讓連結器(linker)找得到. 嗯...

有兩個主要的 Makefiles, 而且在 `src` 目錄下的那個讓我感興趣. 在 Makefile 一行有定義 LOCAL_LIBS 成: LOCAL_LIBS = $(XAWLIB) $(XMULIB) $(XTOOLLIB) $(XLIB) 這所指的 libs 並沒有被連結器找到.

找找下個指到 LOCAL_LIBS 的地方, 我看到該在 Makerfile 的495行:

```
     $(CCLINK) -o $@ $(LDOPTIONS) $(OBJS) $(LOCAL_LIBS) $(LDLIBS)
$(EXTRA_LOAD_FLAGS)
```

而現在 LDLIBS 是什麼呢?

```
     LDLIBS = $(LDPOSTLIB) $(THREADS_LIBS) $(SYS_LIBRARIES)
$(EXTRA_LIBRARIES)
```

SYS_LIBRARIES 是:

```
SYS_LIBRARIES = -lXpm -L../Xc -lXc
```

是個! 就是這樣遺失 libraries.

大概連結器需要在 LOCAL_LIBS 之前看到 LDLIBS... 所以, 第一件事要嘗試去修改 Makefile 就是改寫在495行的 $(LOCAL_LIBS) 和 $(LDLIBS) , 所以它現在變成:

```
        $(CCLINK) -o $@ $(LDOPTIONS) $(OBJS) $(LDLIBS) $(LOCAL_LIBS)
    $(EXTRA_LOAD_FLAGS)                          ^^^^^^^^^^^^^^^^^^^^^^^^
```

在上面的改後,我試著再次執行 **make** , 瞧, 它這次可正常跑了. 當然, Xscrabble 仍然需要一些微調(fine tuning]與玩弄(twiddling), 像是重新命名字典以及標註某些輔助的敘述在原始碼檔案之一, 但正因爲那樣,它已給我好幾個小時的消遣.

你可以寄 e-mail *Matt Chapman* `<mailto:matt@belgarath.demon.co.uk>`, 而且從他的 *home page* `<http://www.belgarath.demon.co.uk/programs/index.html>` 下載 **Xscrabble**.

Scrabble 是 Milton Bradley Co., Inc. 的註冊商標.

---

7 第二個例子: Xloadimage

這個例子提出較簡單的問題. **xloadimage** 程式加入到我的圖形工作組似乎是很有用. 我從一本由 Mui 和 Quercia 所著的好書 11 (X User Tools) 所附的 CD 上的原始碼目錄(source directory), 直接複製 *xloadi41.gz* 檔案下來. 如預期的, 用 **tar xzvf** 解開所有檔案. 然而, **make** 出現了令人討厭的錯誤而且中斷.

```
gcc -c -O -fstrength-reduce -finline-functions -fforce-mem
-fforce-addr -DSYSV  -I/usr/X11R6/include
-DSYSPATHFILE=\"/usr/lib/X11/Xloadimage\" mcidas.c

In file included from /usr/include/stdlib.h:32,
                 from image.h:23,
                 from xloadimage.h:15,
                 from mcidas.c:7:
/usr/lib/gcc-lib/i486-linux/2.6.3/include/stddef.h:215:
conflicting types for ‘wchar_t’
/usr/X11R6/include/X11/Xlib.h:74: previous declaration of
‘wchar_t’
make[1]: *** [mcidas.o] Error 1
make[1]: Leaving directory
‘/home/thegrendel/tst/xloadimage.4.1’
make: *** [default] Error 2
```

這錯誤訊息含有最根本的線索.

看一下檔案 *image.h* 的23行.

```
        #include <stdlib.h>
```

啊哈! 在 *xloadimage*, `wchar_t` 原始碼的某處已經重複定義了在標準 include 檔案 `stdlib.h` 已定義的東西. 首先試試將在 `image.h` 的23行標註(commenting out)起來, 也許 `stdlib.h include` 是不大需要的.

此時, *build* 過程沒有任何重大錯誤. **xloadimage** 程式現在功能都正常.

8 第三個例子: Fortune

最後例子需要某些 C 程式設計知識. 大部分的 Linux 軟體是用 C 寫的, 而且至少學點 C 明顯的對任何想軟體安裝的人會有助益.

惡名昭彰的(notorious) *fortune* 程式在每次 Linux 開機起來時秀出幽默的諺語 "fortune cookie". 不幸地(有雙關意思的), 設法在 Red Hat 發行套件 2.0.30 的核心下建立,出現了 一堆嚴重的錯誤.

```
~/fortune# make all


gcc -O2 -Wall -fomit-frame-pointer -pipe   -c fortune.c -o
fortune.o
fortune.c: In function ‘add_dir’:
fortune.c:551: structure has no member named ‘d_namlen’
fortune.c:553: structure has no member named ‘d_namlen’
make[1]: *** [fortune.o] Error 1
make[1]: Leaving directory ‘/home/thegrendel/for/fortune/fortune’
make: *** [fortune-bin] Error 2
```

看一下 `fortune.c`, 有關聯的幾行在這.

```
   if (dirent->d_namlen == 0)
            continue;
        name = copy(dirent->d_name, dirent->d_namlen);
```

我們需要找出 `dirent` 的 structure, 但它沒有宣告(declared)在 *fortune.c* 檔案中, 想用 **grep dirent** 來秀出是否在其它原始碼的檔案中, 但也沒有. 然而, 在 *fortune.c* 檔的最上方有下列這行.

```
#include <dirent.h>
```

這似乎是系統函式庫的 include 檔案, 所以要找 *dirent.h* 的合理位置是在 */usr/include*. 事實上, *dirent.h* 的確有在 */usr/include* 中, 但該檔沒有包含 `dirent` 的 structure. 然而, 參考另一個 *dirent.h* 檔.

```
#include <linux/dirent.h>
```

最後, 去 */usr/include/linux/dirent.h*, 我們可找到我們所需要宣告的 structure.

```
struct dirent {
        long            d_ino;
        __kernel_off_t  d_off;
        unsigned short  d_reclen;
        char            d_name[256]; /* We must not include
limits.h! */
};
```

足夠地確定, 這個 structure 宣告沒有包含 *d_namelen*, 但有一對與其相當的選擇. 其中最可能的是 *d_reclen*, 因爲 這個 structure member 表示某樣東西的 length 而且它是 short integer. 其他大略, *d_ino*, 可能是 inode number, 判斷它的 name 和 type. 事實上, 我們大概是處理 "directory entry" structure, 而元素表示檔案屬性, 它的名稱, inode, 和 length (以 blocks 作單位). 這似乎對我們的猜想很合理.

我們編輯檔案 *fortune.c*, 而且改變在551行和553行的 `d_namelen` 變成 `d_reclen`. 再試試 `make all`. **Success.** 這次建立沒有錯誤. 我們現在能夠從 *fortune* 獲得 "cheap thrills"

9 哪裏可找到原始碼檔案

現在你很渴望的使用你最新獲得的知識來加入一些工具和其它好東西到你的系統, 你可以在線上找到它們, *Linux Applications and Utilities Page* `<http://www.redhat.com/linux-info/linux-app-list/linapps.html>`, 或者在很合理價位的 CD ROM 的檔案, 藉由 *Red Hat* `<http://www.redhat.com/>`, *InfoMagic* `<mailto:orders@infomagic.com>`, 以及其它的.

一個眾多原始碼的寶庫是在 *comp sources UNIX archive* `<ftp://ftp.vix.com/pub/usenet/comp.sources.unix/>`.

很多 UNIX 原始碼發表在 *alt.sources* 新聞群組. 如果你要找特別的原始碼包裝的, 你可以貼在相關的 *alt.sources.wanted* 新聞群組. 另外一個不錯的地方是查看 *comp.os.linux.announce* 新聞群組. 要取得在 *Unix sources* `<mailto:unix-sources@pa.dec.com>` 的通信論壇, 請寄個 *subscribe* 訊息到那.

至於 *alt.sources* 新聞群組的檔案是在下面 ftp 站:

- *ftp.sterling.com/usenet/alt.sources/* `<ftp://ftp.sterling.com/usenet/alt.sources/>`
- *wuarchive.wustl.edu/usenet/alt.sources/articles* `<ftp://wuarchive.wustl.edu/usenet/alt.sources/articles>`
- *src.doc.ic.ac.uk/usenet/alt.sources/articles* `<ftp://src.doc.ic.ac.uk/usenet/alt.sources/articles>`

10 結語

總結, 百折不撓會使什麼都變成不一樣 (而且高難度挫折門檻明顯會有幫助). 使出全力,從失敗中獲得學習更是重要. 在每個過失的步伐, 每個失敗造就了能夠掌握 建立軟體藝術 的知識個體.

11 參考與進一步閱讀資料

BORLAND C++ TOOLS AND UTILITIES GUIDE, Borland International, 1992, pp. 9-42.
[Borland C++, ver. 3.1. 的發行手冊之一. 給了很好介紹在語法和概念上, 使用 Borland 在 DOS 下殘餘的實作.]

DuBois, Paul: SOFTWARE PORTABILITY WITH IMAKE, O'Reilly and Associates, 1996, ISBN 1-56592-226-3.
[這據說是完整的 imake 參考資料, 雖然我在寫本文時,還未取得.]

Frisch, Aeleen: ESSENTIAL SYSTEM ADMINISTRATION, O'Reilly and Associates, 1995, ISBN 1-56592-127-5.
[這是其它卓越的系統管理手冊已經有概略層面的談到建立軟體.]

Lehey, Greg: PORTING UNIX SOFTWARE, O'Reilly and Associates, 1995, ISBN 1-56592-126-7.

Mui, Linda 和 Valerie Quercia: X USER TOOLS, O'Reilly and Associates, 1994, ISBN 1-56592-019-8, pp. 734-760.

Oram, Andrew 和 Steve Talbott: MANAGING PROJECTS WITH MAKE, O'Reilly and Associates, 1991, ISBN 0-937175-90-0.

Peek, Jerry 和 Tim O'Reilly 與 Mike Loukides: UNIX POWER TOOLS, O'Reilly and Associates / Random House, 1997, ISBN 1-56592-260-3.
[很棒的概念來源, 而且有大量使用工具你可能從原始碼來建立, 使用在本文中討論的方法.]

Stallman, Richard M. 和 Roland McGrath: GNU MAKE, Free Software Foundation, 1995, ISBN 1-882114-78-7.
[應該是需要閱讀的.]

Welsh, Matt 和 Lar Kaufman: RUNNING LINUX, O'Reilly and Associates, 1996, ISBN 1-56592-151-8.
[仍然是全部 Linux 參考資料最好的, 雖然在某些地方缺少比較深入討論.]

當然還有 *make*, *imake*, *xmkmf*, *gcc*, *ldconfig*, *gzip*, *tar*, 和 *patch* 的 man pages.