Debian New Maintainers’ Guide

COLLABORATORS

	TITLE : Debian New Maintainers’ Guide	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Josip Rodin, Osamu Aoki, và Vietnamese Localization Team	March 4, 2019	
dịch bởi		March 4, 2019	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Bắt đầu Đúng cách

Tài liệu này mô tả việc xây dựng một gói Debian cho người dùng Debian bình thường và các nhà phát triển tiềm năng. Nó sử dụng ngôn ngữ khá phi kỹ thuật, và được mô tả với các ví dụ. Có một câu nói Latin cổ đại: *Longum iter est per praecepta, breve et efficax per exempla* (Đó là một chặng đường dài bởi các quy tắc, nhưng ngắn và hiệu quả với các ví dụ).

Việc viết lại tài liệu này với các nội dung cập nhật và nhiều các ví dụ thực tế hơn đã có ở Guide for Debian Maintainers (https://www.debian.org/doc/devel-manuals#debmake-doc) . Vui lòng sử dụng tài liệu mới này như là tài liệu hướng dẫn chính.

Tài liệu này đã được cập nhật cho bản phát hành Debian `jessie`. [1]

Một trong những điều làm cho Debian trở thành một hệ thống gói phần mềm hàng đầu. Mặc dù có một lượng lớn phần mềm đã có trong định dạng Debian, đôi khi bạn cần phải cài đặt phần mềm mà nó không có. Bạn tự hỏi làm thế nàođể bạn có thể tạo gói cho riêng bạn; Và có lẽ bạn nghĩ đó là một nhiệm vụ rất khó khăn. Vâng, nếu bạn là người mới làm quen trên Linux, thật khó, nhưng nếu bạn là một tân binh, bạn sẽ không đọc tài liệu này bây giờ :-) Bạn cần biết một chút về lập trình Unix nhưng bạn chắc chắn không cần phải là một cao thủ. [2]

Tuy nhiên một điều chắc chắn là: để tạo và duy trì đúng các gói Debian phải mất nhiều giờ. Không mắc sai lầm, vì hệ thống của chúng tôi làm việc bảo trì cần phải có cả kỹ thuật có thẩm quyền và cả sự siêng năng.

Nếu bạn cần trợ giúp về đóng gói, vui lòng đọc Phần 1.4.

Các phiên bản mới hơn của tài liệu này luôn sẵn sàng trực tuyến tại http://www.debian.org/doc/maint-guide/ và trong gói `maint-guide`. Bản dịch có thể có sẵn trong các gói như `maint-guide-es`. Xin lưu ý rằng tài liệu này có thể đã lỗi thời.

Vì đây là một hướng dẫn, tôi chọn để giải thích từng bước chi tiết cho một số chủ đề quan trọng. Một số người trong số họ có thể không có liên quan đến bạn. Làm ơn hãy kiên nhẫn. Tôi cũng đã cố ý bỏ qua một số trường hợp góc và chỉ cung cấp con trỏ để giữ tài liệu này đơn giản.

1.1 Động lực xã hội của Debian

Dưới đây là một số quan sát về động lực xã hội của Debian, được trình bày với hy vọng rằng nó sẽ chuẩn bị cho bạn tương tác với Debian:

- Tất cả chúng ta đều là tình nguyện viên.
  - Bạn không thể áp đặt lên những người khác phải làm gì.
  - Bạn nên có động cơ để làm việc một mình.

[1] Tài liệu giả định bạn đang sử dụng `jessie` hoặc hệ thống mới hơn. Nếu bạn cần phải làm theo các văn bản này trong một hệ thống cũ hơn (bao gồm cả một hệ thống Ubuntu cũ hơn vv), bạn phải cài đặt ít nhất các gói `dpkg` và `debhelper`.

[2] Bạn có thể tìm hiểu về cách xử lý cơ bản của một hệ thống Debian từ Debian Reference (http://www.debian.org/doc/manuals/debian-reference/) . Nó cũng chứa một số gợi ý để tìm hiểu về lập trình Unix.

- Hợp tác thân thiện là động lực.
  - Đóng góp của bạn không nên quá căng thẳng người khác.
  - Đóng góp của bạn chỉ có giá trị khi những người khác đánh giá cao nó.
- Debian không phải là trường học của bạn mà bạn nhận được sự chú ý tự động của giáo viên.
  - Bạn sẽ có thể tự học hỏi nhiều điều.
  - Sự chú ý của các tình nguyện viên khác là một nguồn lực khan hiếm.
- Debian đang không ngừng cải tiến.
  - Bạn sẽ phải tạo ra các gói chất lượng cao.
  - Bạn nên thích nghi với chính mình để thay đổi.

Có nhiều người tương tác với Debian với các vai trò khác nhau:

- **upstream author**: người đã tạo ra chương trình gốc.
- **upstream maintainer**: người hiện đang duy trì bảo trì chương trình gốc.
- **maintainer**: người tạo ra gói Debian cho chương trình.
- **sponsor**: một người giúp đỡ người bảo trì tải các gói lên kho lưu trữ Debian chính thức (sau khi kiểm tra nội dung của chúng).
- **mentor**: người giúp những người mới làm quen với bảo trì gói vv
- **Debian Developer** (DD): thành viên của dự án Debian với quyền tải đầy đủ lên kho lưu trữ Debian chính thức.
- **Debian Maintainer** (DM): người có quyền tải lên giới hạn về kho lưu trữ gói Debian chính thức.

Xin lưu ý rằng bạn không thể trở thành một **Debian Developer** (DD) chính thức qua 1 đêm, vì nó đòi hỏi nhiều kỹ năng hơn. Xin đừng nản lòng vì điều này. Nếu nó là hữu ích cho người khác, bạn vẫn có thể tải gói của bạn hoặc như là một **maintainer** thông qua một **sponsor** hoặc như một **Debian Maintainer**.

Xin lưu ý rằng bạn không cần tạo bất kỳ gói mới nào để trở thành Nhà phát triển Debian chính thức. Đóng góp cho các gói hiện có cũng có thể cung cấp một con đường để trở thành Nhà phát triển Debian chính thức. Có nhiều gói đang chờ người bảo dưỡng tốt (xem Phần 2.2).

Vì chúng ta chỉ tập trung vào các khía cạnh kỹ thuật của đóng gói trong tài liệu này, vui lòng tham khảo những điều dưới đây để tìm hiểu cách các chức năng của Debian và cách bạn có thể tham gia:

- Debian: 17 năm của Phần mềm Tự do,"do-ocracy", và dân chủ (http://upsilon.cc/~zack/talks/2011/20110321-taipei.pdf) (Các trang giới thiệu)
- Làm thế nào bạn có thể giúp Debian? (http://www.debian.org/intro/help) (chính thức)
- Câu hỏi Thường Gặp về Debian GNU / Linux, Chương 13 -"Góp phần Dự án Debian" (http://www.debian.org/doc/FAQ/ch-contributing) (bán chính thức)
- Debian Wiki, HelpDebian (http://wiki.debian.org/HelpDebian) (bổ sung)
- Trang web thành viên của Debian (https://nm.debian.org/) (chính thức)
- Câu hỏi thường gặp của trợ giảng của Debian (http://wiki.debian.org/DebianMentorsFaq) (bổ sung)

1.2 Các chương trình cần thiết cho sự phát triển

Trước khi bạn bắt đầu bất cứ điều gì, bạn nên chắc chắn rằng bạn đã cài đặt đúng một số gói bổ sung cần thiết cho sự phát triển. Lưu ý rằng danh sách không chứa bất kỳ gói nào được đánh dấu `essential` hoặc `required` - chúng tôi mong rằng bạn đã cài đặt xong.

Các gói sau đi kèm với cài đặt Debian tiêu chuẩn, do đó có lẽ bạn đã có chúng rồi (cùng với bất kỳ gói bổ sung nào mà chúng phụ thuộc). Tuy nhiên, bạn nên kiểm tra chúng bằng gói `aptitude show` *pakage* hoặc với `dpkg -s` *package*.

Gói phần quan trọng nhất để cài đặt trên hệ thống phát triển của bạn là gói `build-essential`. Một khi bạn cố gắng cài đặt nó, nó sẽ *pull in* gói khác cần thiết để có một môi trường xây dựng cơ bản.

Đối với một số loại bao bì, đó là tất cả những gì bạn cần; Tuy nhiên, có một bộ gói khác mà mặc dù không cần thiết cho tất cả các gói được xây dựng rất hữu ích để cài đặt hoặc có thể được yêu cầu bởi gói của bạn:

- `autoconf`, `automake`, và `autotools-dev` - nhiều chương trình mới hơn sử dụng các tập lệnh cấu hình và `Makefile` các tệp tin được xử lý trước với sự trợ giúp của các chương trình như thế này (xem `info autoconf`, `info automake`). `autotools-dev` giữ các phiên bản cập nhật của các tệp tin tự động nhất định và có tài liệu về cách tốt nhất để sử dụng các tệp đó.
- `debhelper` và `dh-make` - `dh-make` là cần thiết để tạo ra bộ xương của gói ví dụ của chúng tôi, Và nó sẽ sử dụng một số công cụ `debhelper` để tạo các gói. Chúng không cần thiết cho mục đích này, nhưng được *đánh giá cao* được đề xuất cho người bảo trì mới. Nó làm cho toàn bộ quá trình rất dễ dàng để bắt đầu, và để kiểm soát sau đó. (Xem dh_make(8), debhelper(1).) [3]

  Công cụ mới `debmake` có thể sử dụng thay thế cho tiêu chuẩn `dh-make`. Nó làm nhiều hơn và đi kèm thêm tài liệu HTML với các mẫu ví dụ đóng gói thêm trong `debmake-doc`.
- `devscripts` - gói này chứa một số kịch bản hữu ích có thể giúp ích cho người bảo trì, nhưng chúng cũng không cần thiết cho việc xây dựng các gói. Các gói đề nghị và đề xuất bởi gói này rất đáng để xem xét. (Xem `/usr/share/doc/devscripts/README.gz`.)
- `fakeroot` - công cụ tiện ích này cho phép bạn giả lập root khi cần thiết cho một số phần của quá trình xây dựng. (Xem fakeroot(1).)
- `file` - chương trình tiện ích này có thể xác định loại tập tin là gì. (Xem file(1).)
- `gfortran` - trình biên dịch GNU Fortran 95, sẽ là cần thiết nếu chương trình của bạn được viết bằng Fortran. (Xem gfortran(1).)
- `git` - gói này cung cấp một hệ thống quản lý phiên bản phổ biến được thiết kế để xử lý các dự án rất lớn với tốc độ và hiệu quả; Nó được sử dụng cho nhiều dự án mã nguồn mở lớn, nhất là nhân Linux. (Xem git(1), Tài liệu git (`/usr/share/doc/git-doc/index.html`).)
- `gnupg` - một công cụ cho phép bạn dùng chữ ký số để *ký* gói. Điều này đặc biệt quan trọng nếu bạn muốn phân phối các gói cho người khác, và chắc chắn bạn sẽ làm điều đó khi tác phẩm của bạn được đưa vào phân phối Debian. (Xem gpg(1).)
- `gpc` - trình biên dịch Pascal GNU, cần thiết nếu chương trình của bạn được viết bằng Pascal. Đáng lưu ý ở đây là `fp-compiler`, Free Pascal Compiler, cũng tốt trong nhiệm vụ này. (Xem gpc(1), ppc386(1).)
- `lintian` - đây là trình kiểm tra gói Debian, để cho bạn biết về bất kỳ lỗi phổ biến nào sau khi bạn xây dựng gói và giải thích các lỗi được tìm thấy. (Xem lintian(1), Hướng dẫn sử dụng Lintian (https://lintian.debian.org/manual/index.html) .)
- `patch` - tiện ích hữu ích này sẽ lấy một tệp có chứa sự thay đổi (được sản sinh bởi chương trình **diff**) và áp dụng nó vào tệp gốc, tạo ra một phiên bản vá. (Xem patch(1).)
- `patchutils` - gói này chứa một số tiện ích để làm việc với các bản vá như **lsdiff**, **interdiff** và lệnh **filterdiff** .
- `pbuilder` - gói này chứa các chương trình được sử dụng để tạo và duy trì môi trường **chroot**. Xây dựng gói Debian trong môi trường **chroot** này sẽ kiểm tra sự phụ thuộc thư viện phù hợp và tránh lỗi FTBFS (Fails To Build From Source). (Xem pbuilder (8) và pdebuild (1))

[3] Ngoài ra còn có một số gói chuyên dụng hơn nhưng tương tự như `dh-make-perl`, `dh-make-php`, v.v.

- `perl` - Perl là một trong những ngôn ngữ lập trình được sử dụng phổ biến nhất trên các hệ thống Unix ngày nay, thường được gọi là Unix's Swiss Army Chainsaw. (Xem perl(1).)
- `python` - Python là một trong những ngôn ngữ kịch bản thông dịch được sử dụng nhiều nhất trên hệ thống Debian, kết hợp sức mạnh vượt trội với cú pháp rất rõ ràng. (Xem python(1).)
- `quilt` - gói này giúp bạn quản lý một số lượng lớn các bản vá lỗi bằng cách theo dõi những thay đổi mà mỗi bản vá lỗi tạo ra. Các bản vá lỗi có thể được áp dụng, không áp dụng, làm mới và nhiều hơn nữa. (Xem quilt(1) và `/usr/share/doc/quilt/quilt.pdf.gz` .)
- `xutils-dev` - một vài chương trình, thường xây dựng cho X11, sử dụng chương trình này để tạo ra các tệp tin `Makefile` từ các cài đặt của các hàm macro. (Xem imake(1), xmkmf(1).)

Các mô tả ngắn được đưa ra ở trên chỉ phục vụ để giới thiệu cho bạn những gì mỗi gói thực hiện. Trước khi tiếp tục, hãy đọc tài liệu của mỗi chương trình có liên quan, bao gồm cả những chương trình được cài đặt thông qua sự phụ thuộc gói như **make**, ít nhất, cho việc sử dụng tiêu chuẩn. Có vẻ như bây giờ kiến thức khá nặng, nhưng sau đó bạn sẽ *rất* vui vì bạn đã đọc nó. Nếu bạn có câu hỏi cụ thể sau này, tôi sẽ đề nghị đọc lại các tài liệu đã đề cập ở trên.

1.3 Tài liệu cần thiết cho sự phát triển

Sau đây là tài liệu *rất quan trọng* mà bạn nên đọc cùng với tài liệu này:

- `debian-policy` - Debian Policy Manual (http://www.debian.org/doc/devel-manuals#policy) bao gồm các giải thích về cấu trúc và nội dung của kho lưu trữ Debian, một số vấn đề thiết kế hệ điều hành, Filesystem Hierarchy Standard (http://www.debian.org/doc/packaging-manuals/fhs/fhs-3.0.html) (FHS, nơi mà mỗi tập tin và thư mục phải là),vv .Đối với bạn, điều quan trọng nhất là nó mô tả các yêu cầu mà mỗi gói phải đáp ứng để được đưa vào trong phân phối. (Xem bản sao địa phương của `/usr/share/doc/debian-policy/policy.pdf.gz` and `/usr/share/doc/debian-policy/fhs/fhs-3.0.pdf.gz`.)
- `developer-reference` - Debian Developer's Refrence (http://www.debian.org/doc/devel-manuals#devref) mô tả tất cả các vấn đề không cụ thể về các chi tiết kỹ thuật của gói, như cấu trúc của kho lưu trữ, cách đổi tên, mồ côi, hoặc nhận các gói, làm thế nào để làm NMUs, làm thế nào để quản lý lỗi, cách thức đóng gói tốt nhất, khi nào và ở đâu để tải lên vv (Xem bản sao của địa phương của `/usr/share/doc/developers-reference/developers-reference.pdf`.)

Sau đây là tài liệu *quan trọng* mà bạn nên đọc cùng với tài liệu này:

- Autotools Tutorial (http://www.lrde.epita.fr/~adl/autotools.html) cung cấp một hướng dẫn rất tốt cho GNU Build System được biết như là GNU Autotools là thành phần quan trọng nhất cho Autoconf, Automake, Libtool, and gettext.
- `gnu-standards` - gói này chứa hai phần tài liệu từ dự án GNU: GNU Coding Standards (http://www.gnu.org/prep/standards/html_node/index.html) , và Information for Maintainers of GNU Software (http://www.gnu.org/prep/maintain/html_node/index.html) . Mặc dù Debian không yêu cầu những điều này được tuân theo, những điều này vẫn còn hữu ích như các hướng dẫn và thông thường. (Xem bản sao địa phương của `/usr/share/doc/gnu-standards/standards.pdf.gz` và `/usr/share/doc/gnu-standards/maintain.pdf.gz`.)

Nếu tài liệu này mâu thuẫn với bất kỳ tài liệu nào được đề cập ở trên, nếu chúng là đúng. Vui lòng gửi một báo cáo lỗi trên `maint-guide` gói bằng cách sử dụng **reportbug**.

Sau đây là một tài liệu hướng dẫn thay thế mà bạn có thể đọc cùng với tài liệu này:

- Debian Packaging Tutorial (http://www.debian.org/doc/packaging-manuals/packaging-tutorial/packaging-tutorial)

1.4 Nơi để yêu cầu trợ giúp

Trước khi bạn quyết định đặt câu hỏi của mình ở nơi công cộng, vui lòng đọc tài liệu tốt

- Các tệp trong gói `/usr/share/doc/`*`package`* cho tất cả các gói phù hợp
- Nội dung của lệnh **`man`** *`command`* cho tất cả các lệnh thích hợp
- Nội dung của lệnh **`info`** *`command`* cho tất cả các lệnh thích hợp
- các nội dung tại debian-mentors@lists.debian.org mailing list archive (http://lists.debian.org/debian-mentors/)
- các nội dung tại debian-devel@lists.debian.org mailing list archive (http://lists.debian.org/debian-devel/)

Bạn có thể sử dụng công cụ tìm kiếm web hiệu quả hơn bằng cách bao gồm các chuỗi tìm kiếm như `site:lists.debian.org` để hạn chế tên miền.

Làm một gói thử nghiệm nhỏ là một cách hay để tìm hiểu chi tiết về đóng gói. Kiểm tra các gói đã được duy trì tốt hiện tại là cách tốt nhất để tìm hiểu làm thế nào người khác đóng các gói

Nếu bạn vẫn có câu hỏi về đóng gói mà bạn không thể tìm thấy câu trả lời trong tài liệu sẵn có và tài nguyên web, bạn có thể nhờ họ tương tác:

- debian-mentors@lists.debian.org mailing list (http://lists.debian.org/debian-mentors/) . (Danh sách gửi thư này dành cho người mới bắt đầu.)
- debian-devel@lists.debian.org mailing list (http://lists.debian.org/debian-devel/) . (Danh sách gửi thư dành cho chuyên gia)
- IRC (http://www.debian.org/support#irc) ví dụ như `#debian-mentors`.
- Các đội tập trung vào một bộ gói cụ thể. (Danh sách đầy đủ tại https://wiki.debian.org/Teams (https://wiki.debian.org/Teams))
- Danh sách gửi thư cụ thể theo ngôn ngữ ví dụ như debian-devel-{french,italian,portuguese,spanish}@lists.debian.org hoặc debian-devel@debian.or.jp. (Danh sách đầy đủ tại https://lists.debian.org/devel.html (https://lists.debian.org/devel.html) và https://lists.debian.org/users.html (https://lists.debian.org/users.html))

Các nhà phát triển giàu kinh nghiệm của Debian sẽ sẵn sàng giúp bạn, nếu bạn hỏi đúng cách sau khi thực hiện những nỗ lực bắt buộc của bạn.

Khi bạn nhận được báo cáo lỗi (vâng, lỗi thực sự!), Bạn sẽ biết rằng đã đến lúc bạn phải khai thác vào Debian Bug Tracking System (http://www.debian.org/Bugs/) và đọc tài liệu ở đó, để được có khả năng giải quyết các báo cáo hiệu quả. Tôi rất khuyên bạn nên đọc Debian Developer's Reference, 5.8. "Handling bugs" (http://www.debian.org/doc/manuals/developers-reference/-pkgs.html#bug-handling) .

Ngay cả khi tất cả đều hoạt động tốt, đó là thời gian để bắt đầu cầu nguyện. Tại sao? Bởi vì chỉ trong vài giờ (hoặc ngày) người dùng từ khắp nơi trên thế giới sẽ bắt đầu sử dụng gói của bạn và nếu bạn mắc một số lỗi nghiêm trọng, bạn sẽ nhận được mailbombed bởi nhiều người dùng Debian tức giận ⋯... Đùa thôi :-)

Hãy thư giãn và sẵn sàng cho các báo cáo lỗi, vì còn rất nhiều việc phải làm trước khi gói của bạn hoàn toàn phù hợp với các chính sách của Debian và các hướng dẫn thực hành tốt nhất của nó (một lần nữa đọc *tài liệu* thật chi tiết). Chúc may mắn!

Chapter 2

Những bước đầu tiên

Hãy bắt đầu bằng cách tạo một gói riêng của bạn (hoặc, thậm chí tốt hơn, chấp nhận một cái hiện tại).

2.1 Luồng làm việc tạo ra gói Debian

Nếu bạn đang tạo ra một gói Debian từ một chương trình thượng nguồn, quy trình làm việc điển hình của việc xây dựng gói Debian liên quan đến việc tạo ra một số tệp được đặt tên cụ thể cho từng bước như sau:

- Lấy một bản sao của phần mềm thượng nguồn, thường ở dạng nén tar.
  - *package-version*`.tar.gz`
- Thêm các sửa đổi đóng gói cụ thể của Debian vào chương trình thượng nguồn trong thư mục `debian` và tạo một gói mã nguồn mới (non-native) (tức là bộ tập tin đầu vào sử dụng cho việc xây dựng gói Debian) với định dạng `3.0 (quilt)`.
  - *package_version*`.orig.tar.gz`
  - *package_version-revision*`.debian.tar.gz`[1]
  - *package_version-revision*`.dsc`
- Xây dựng các gói nhị phân Debian, là những gói có thể cài đặt bình thường trong định dạng `.deb` (hoặc định dạng `.udeb`, được sử dụng bởi Trình cài đặt Debian) từ gói nguồn Debian.
  - *package_version-revision_arch*`.deb`

Xin lưu ý rằng ký tự phân tách *package* và *version* đã thay đổi từ `-` (dấu trừ) trong tên tệp nén thành _ (gạch dưới) ở trong tên của gói Debian

Trong tên tệp ở trên, hãy thay thế *package* bằng **package name**, tên *version* bằng **upstream version**, *revision* thành **Debian revision**, và *arch* thành **package architecture**, như đã định nghĩa trong Debian Policy Manual. [2]

Mỗi bước của phác thảo này được giải thích với các ví dụ chi tiết trong các phần sau.

[1] Đối với các gói mã nguồn Debian kiểu cũ với định dạng `1.0`, *package_version-revision*`.diff.gz` được sử dụng để thay thế.

[2] Xem 5.6.1 "Source" (http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Source) , 5.6.7 "Package" (http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Package) , và 5.6.12 "Version" (http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Version) .**kiến trúc gói** theo Debian Policy Manual, 5.6.8 "Architecture" (http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Architecture) và được tự động ký bởi tiến trình xây dựng gói.

2.2 Chọn chương trình của bạn

Có thể bạn đã chọn gói bạn muốn tạo. Điều đầu tiên bạn cần làm là kiểm tra xem gói có trong kho lưu trữ đã được phân phối bằng cách sử dụng như sau:

- lệnh **aptitude**
- trang web Debian packages (http://www.debian.org/distrib/packages)
- trang web Debian Package Tracker (https://tracker.debian.org/)

Nếu một gói đã tồn tại, tốt, hãy cài đặt nó! :-) Nếu lúc cài đặt xuất hiện từ **orphaned** (điều đó có nghĩa người bảo trì gói đấy đã thiết lập tại Debian QA Group (http://qa.debian.org/)), bạn có thể chọn nó nếu nó vẫn có sẵn. Bạn cũng có thể giúp một gói nếu người bảo trị của gói đó đã tạo một "Request for Adoption" (**RFA**).[3]

Có một số tài nguyên trạng thái quyền sở hữu gói:

- Lệnh **wnpp-alert** từ gói `devscripts`
- Các gói công việc Cần thiết và Triển vọng (http://www.debian.org/devel/wnpp/)
- Debian Bug report logs: Bugs in pseudo-package `wnpp` in `unstable` (http://bugs.debian.org/wnpp)
- Debian Packages that Need Lovin' (http://wnpp.debian.net/)
- Browse `wnpp` bugs based on debtags (http://wnpp-by-tags.debian.net/)

Một lưu ý quan trọng là Debian đã có các gói cho hầu hết các loại chương trình, và số lượng các gói đã có trong kho lưu trữ Debian lớn hơn nhiều so với các cộng tác viên có quyền tải lên. Do đó, đóng góp cho các gói đã có sẵn trong kho lưu trữ luôn được đánh giá cao hơn (và nhiều khả năng nhận tài trợ) bởi các nhà phát triển khác [4]. Bạn có thể đóng góp bằng nhiều cách:

- Tiếp nhận gói mồ côi, nhưng vẫn đang được sử dụng
- tham gia vào packaging teams (http://wiki.debian.org/Teams)
- xử lý lỗi của các gói phổ biến
- chuẩn bị QA or NMU uploads (http://www.debian.org/doc/developers-reference/pkgs.html#nmu-qa-upload)

Nếu bạn có thể nhận bảo trì gói nào, hãy lấy nguồn (bằng các công cụ như `apt-get source` *`packagename`*) và kiểm tra chúng. Tài liệu này tiếc là không bao gồm thông tin toàn diện về việc nhận bảo trì gói. Rất may, bạn không cần phải mất nhiều thời gian khó khăn để tìm hiểu cách gói làm việc, khi đã có ai đó trước đây thực hiện các thiết lập ban đầu cho bạn. Hãy tiếp tục đọc, và; Rất nhiều lời khuyên dưới đây vẫn sẽ được áp dụng cho trường hợp của bạn.

Nếu là một gói mới, và bạn quyết định muốn xem nó trong Debian, thực hiện theo các bước sau:

- Trước tiên, bạn phải biết rằng chương trình hoạt động như thế nào, và đã thử nó một thời gian để xác nhận tính hữu dụng của nó.
- Bạn phải kiểm tra xem có ai khác đang làm việc trên gói trên trang Work-Needing và Prospective Packages (http://www.debian.org/devel/wnpp/) . Nếu không có ai khác đang làm việc trên nó, hãy gửi báo cáo lỗi ITP (Intent To Package) đến gói `wnpp` bằng cách sử dụng lệnh **reportbug**. Nếu ai đó đã làm việc đó rồi, hãy liên hệ với họ nếu bạn cảm thấy cần. Nếu không —hãy tìm một chương trình thú vị mà chưa ai đang duy trì.
- Phần mềm **phải có một giấy phép**.

[3] Xem phần Debian Developer's Reference 5.9.5. "Adopting a package" (http://www.debian.org/doc/manuals/developers-reference/pkgs.html#adopting) .
[4] Có thể nói rằng, sẽ luôn có những chương trình mới sẽ thật sự cần đóng gói mới

  - Đối với mục `main`, Chính sách Debian yêu cầu các gói phải **tuân thủ hoàn toàn Nguyên tắc về Phần mềm Tự do Debian** (DFSG (http://www.debian.org/social_contract#guidelines) và **không yêu cầu một gói nào bên ngoài mục `main`** để biên dịch hay thực thi. Đây là trường hợp bắt buộc.
  - Đối với mục `contrib`, nó phải tuân thủ DFSG nhưng nó có thể yêu cầu một gói bên ngoài mục `main` để biên dịch hoặc thực thi.
  - Còn mục `non-free`, nó có thể không tương thích với DFSG nhưng **cần được phân phối** .
  - Nếu bạn không chắc chắn về vị trí của gói đó, hãy đăng hỏi về giấy phép trên debian-legal@lists.debian.org (http://lists.debian.org/-debian-legal/) và xin tư vấn.

- Chương trình **không nên** tác động tới mối quan tâm bảo mật và bảo trì cho hệ thống Debian.
  - Chương trình nên được ghi chép đầy đủ và mã của nó cần phải có thể hiểu được (không làm rối)
  - Bạn nên liên hệ với tác giả của chương trình để kiểm tra nếu họ đồng ý với việc đóng gói nó trở nên thân thiện với Debian. Điều quan trọng là có thể tham khảo ý kiến của tác giả trong trường hợp có bất kỳ vấn đề với chương trình, do đó, không cố gắng với gói phần mềm không còn phát triển/bảo trì từ thượng nguồn
  - Chương trình chắc chắn **không nên** chạy setuid root, hoặc thậm chí cao hơn, nó không cần phải setuid hoặc setgid để làm bất cứ điều gì.
  - Chương trình không nên là một daemon, hoặc cho vào trong thư mục `*/sbin`, hoặc mở một cổng như là root.

Tất nhiên, cuối cùng chỉ là một biện pháp an toàn, và nhằm cứu bạn khỏi sự kích động của người dùng nếu bạn làm điều gì đó sai trái trong một số daemon setuid···Khi bạn có nhiều kinh nghiệm hơn trong trong gói, bạn sẽ có thể gói phần mềm kiểu như vậy.

Là một người bảo trì mới, bạn được khuyến khích để có được một số kinh nghiệm trong đóng gói với các gói đơn giản và không khuyến khích từ việc tạo ra các gói phức tạp.

- Gói đơn giản
  - gói nhị phân duy nhất, arch = all (bộ sưu tập dữ liệu như hình nền)
  - gói nhị phân duy nhất, arch = all (các tệp thực thi được viết bằng ngôn thông dịch như POSIX shell)
- Gói phức tạp trung bình
  - gói nhị phân duy nhất, arch = all (tệp tin thực thi nhị phân của ELF được biên dịch từ các ngôn ngữ như C và C ++)
  - nhiều gói nhị phân, arch = any + all (các gói cho các tập tin thực thi nhị phân ELF + tài liệu)
  - thượng nguồn ở định dạng khác với `tar.gz` hoặc `tar.bz2`
  - thượng nguồn chứa nội dung không thể giải quyết
- Gói phức tạp cao
  - gói chứa các kịch bản được sử dụng bởi các gói khác
  - gói thư viện ELF chung được sử dụng bởi các gói khác
  - nhiều gói nhị phân bao gồm gói thư viện ELF
  - gói nguồn có nhiều thượng nguồn
  - gói module hạt nhân
  - các gói bản vá cho hạt nhân
  - bất kỳ gói nào kịch bản người bảo trì không tầm thường

Đóng gói gói phức tạp cao không phải là quá khó, nhưng nó đòi hỏi một chút kiến thức. Bạn nên tìm kiếm hướng dẫn cụ thể cho từng tính năng phức tạp. Ví dụ, một số ngôn ngữ có tài liệu chính sách phụ của riêng mình:

- Perl policy (http://www.debian.org/doc/packaging-manuals/perl-policy/)
- Python policy (http://www.debian.org/doc/packaging-manuals/python-policy/)

- Java policy (http://www.debian.org/doc/packaging-manuals/java-policy/)

Có một câu trong tiếng Latinh nói rằng: *fabricando fit faber* (thực hành làm cho hoàn hảo). Đó là *tính cao nhất* được đề nghị để thực hành và thử nghiệm với tất cả các bước của gói Debian với các gói đơn giản trong khi đọc hướng dẫn này. Một tarball thượng nguồn đơn giản `hello-sh-1.0.tar.gz` được tạo như sau có thể cung cấp một điểm khởi đầu tốt: [5]

```
$ mkdir -p hello-sh/hello-sh-1.0; cd hello-sh/hello-sh-1.0
$ cat > hello <<EOF
#!/bin/sh
# (C) 2011 Foo Bar, GPL2+
echo "Hello!"
EOF
$ chmod 755 hello
$ cd ..
$ tar -cvzf hello-sh-1.0.tar.gz hello-sh-1.0
```

2.3 Lấy chương trình, và hãy thử nó

Vì vậy, điều đầu tiên cần làm là tìm và tải về mã nguồn gốc. Có lẽ bạn đã có tập tin nguồn mà bạn đã chọn ở trang chủ của tác giả. Mã nguồn của các chương trình Unix tự do thường có trong **tar**+**gzip** định dạng với phần mở rộng là `.tar.gz`, **tar**+**bzip2** định dạng với phần mở rộng là `.tar.bz2` , hoặc **tar**+**xz** định dạng với phần mở rộng là `.tar.xz` . Các bản nén này thường chứa một thư mục gọi là *`package-version`* với tất cả các nguồn bên trong.

Nếu phiên bản mới nhất của mã nguồn có sẵn thông qua Version Control System(VCS) như Git, Subversion hoặc CVS, bạn cần phải lấy nó với `git clone`, `svn co` hoặc `cvs co` và đóng gói lại nó vào **tar**+**gzip** định dạng cho mình bằng cách sử dụng tuỳ chọn `--exclude-vcs`

Nếu mã nguồn chương trình của bạn xuất hiện dưới dạng một số loại lưu trữ khác (ví dụ: tên tập tin kết thúc bằng `Z` hoặc `.zip` [6]), Bạn cũng nên giải nén nó bằng các công cụ thích hợp và đóng gói lại nó.

Nếu mã nguồn chương trình của bạn đi kèm với một số nội dung không tương thích DFSG, bạn cũng nên giải nén nó để loại bỏ các nội dung đó và đóng gói lại nó bằng một phiên bản chỉnh sửa của thượng nguồn có chứa `dfsg`.

Với một ví dụ: Tôi sẽ sử dụng chương trình được gọi là **gentoo** , trình quản lý tệp GTK+. [7]

Tạo một thư mục con trong thư mục home của bạn có tên là `debian` hoặc `deb` hoặc bất cứ thứ gì bạn thấy thích hợp (ví dụ như `~/gentoo` sẽ là tốt trong trường hợp này). Đặt tệp nén mã nguồn đã tải xuống vào trong đó và giải nén (với `tar xzf gentoo-0.9.12.tar.gz`). Đảm bảo không có thông báo cảnh báo lỗi, thậm chí *irrelevant*, bởi vì các công cụ giải nén của người khác có thể hoặc không thể bỏ qua những dị thường này, vì vậy họ có thể gặp sự cố khi giải nén chúng. Dòng lệnh trình báo của bạn có thể giống như sau:

```
$ mkdir ~/gentoo ; cd ~/gentoo
$ wget http://www.example.org/gentoo-0.9.12.tar.gz
$ tar xvzf gentoo-0.9.12.tar.gz
$ ls -F
gentoo-0.9.12/
gentoo-0.9.12.tar.gz
```

Bây giờ bạn có một thư mục con khác, gọi là `gentoo-0.9.12`. Đi vào thư mục đó và *cẩn trọng* đọc tài liệu được cung cấp. Thường có các tệp có `README*`, `INSTALL*`, `*.lsm` hoặc `*.html` . Bạn phải tìm hướng dẫn về cách biên dịch và cài đặt

[5] Đừng lo lắng về việc thiếu tệp `Makefile` . Bạn có thể cài đặt lệnh **hello** bằng cách sử dụng **debhelper** như trong Phần 5.11, hoặc bằng cách sửa đổi nguồn thượng nguồn để thêm tệp `Makefile` với mục install như trong Chương 3.

[6] Bạn có thể xác định định dạng lưu trữ bằng sử dụng lệnh **file** khi phần mở rộng tệp là không đủ thông tin.

[7] Chương trình này đã được đóng gói. phiên bản hiện tại (http://packages.qa.debian.org/g/gentoo.html) sử dụng Autotools làm cấu trúc xây dựng nên sẽ khác với các ví dụ sau dựa trên phiên bản 0.9.12.

chương trình (có lẽ họ giả sử rằng bạn sẽ muốn cài đặt chương trình vào thư mục `/usr/local/bin`), bạn sẽ không làm điều đó, nhưng còn nhiều thứ về sau trong Phần 3.3).

Bạn nên bắt đầu đóng gói với một thư mục mã nguồn hoàn toàn sạch sẽ (nguyên sơ), hoặc chỉ đơn giản với mã nguồn mới được giải nén

2.4 Xây dựng hệ thống đơn giản

Các chương trình đơn giản thường có một `Makefile` và có thể được biên dịch chỉ bằng cách chạy lệnh `make`. [8] Một số còn hỗ trợ `make check`, nhằm tự kiểm tra điều kiện biên dịch. Để cài đặt chương trình vào thư mục đích thường được thực hiện với `make install`.

Bây giờ hãy thử biên dịch và chạy chương trình của bạn, để đảm bảo nó hoạt động đúng và không làm hỏng cái gì khác trong khi cài đặt hoặc chạy.

Ngoài ra, bạn có thể chạy `make clean` (hoặc tốt hơn nữa là `make distclean`) để dọn dẹp thư mục build. Đôi khi thậm chí còn hỗ trợ kịch bản gỡ cài đặt, `make uninstall` được sử dụng để xóa tất cả các tập tin đã cài đặt.

2.5 Xây dựng hệ thống phổ biến

Rất nhiều chương trình phần mềm tự do được viết bằng ngôn ngữ C và C++. Nhiều trong số đó sử dụng Autotools hoặc CMake để làm cho chúng dễ dàng biên dịch đối với các nền tảng khác nhau. Các công cụ xây dựng này cần được sử dụng để tạo tệp `Makefile` từ tệp nguồn bắt buộc khác. Sau đó, các chương trình dễ dàng được xây dựng bằng cách sử dụng `make; make install` .

Autotools là hệ thống xây dựng GNU bao gồm Autoconf, Automake, Libtool, và gettext. Bạn có thể nhận ra các nguồn như vậy bằng các tệp `configure.ac`, `Makefile.am`, và `Makefile.in`.[9]

Bước đầu tiên của quy trình làm việc với Autotools thường là chạy với bản thượng nguồn `autoreconf -i -f` trong thư mục nguồn và các tệp được tạo ra cùng với mã nguồn.

```
configure.ac-----+-> autoreconf -+-> configure
Makefile.am -----+        |      +-> Makefile.in
src/Makefile.am -+        |      +-> src/Makefile.in
                          |      +-> config.h.in
                      automake
                      aclocal
                      aclocal.m4
                      autoheader
```

Việc chỉnh sửa các tệp `configure.ac` và `Makefile.am` yêu cầu vài kiên thức về **autoconf** và **automake**. Xem `info autoconf` và `info automake`.

Bước thứ hai của quy trình làm việc Autotools thường là chúng ta sẽ có được các tệp mới sinh ra và chỉ cần chạy `./configure && make` trong thư mục nguồn để biên dịch thành chương trình ***`binary`***.

```
Makefile.in -----+               +-> Makefile -----+-> make -> binary
src/Makefile.in -+-> ./configure -+-> src/Makefile -+
config.h.in -----+               +-> config.h -----+
                 |
  config.status -+
  config.guess --+
```

[8] Nhiều chương trình hiện đại đi kèm với một kịch bản `configure` mà khi thực hiện tạo tệp `Makefile` tùy chỉnh linh hoạt đối với hệ thống của bạn.

[9] Autotools là công cụ quá lớn để giải quyết trong hướng dẫn nhỏ này. Phần này chỉ nhằm cung cấp các từ khoá và tham khảo. Hãy đảm bảo đọc Autotools Tutorial (http://www.lrde.epita.fr/~adl/autotools.html) và bản sao cục bộ của `/usr/share/doc/autotools-dev/README.Debian.gz`, nếu bạn cần sử dụng nó.

Bạn có thể thay đổi nhiều thứ trong `Makefile`; ví dụ bạn có thể thay đổi thư mục mặc định để cài đặt tệp bằng cách sử dụng tùy chọn `./configure --prefix=/usr`.

Mặc dù không bắt buộc, việc cập nhật cấu hình `configure` và các tệp khác với `autoreconf -i -f` có thể cải thiện tính tương thích của mã nguồn. [10]

CMake là một hệ thống xây dựng thay thế. Bạn có thể nhận ra các nguồn như vậy bằng tệp `CMakeLists.txt`.

2.6 Tên và phiên bản gói

Nếu mã nguồn thượng nguồn là `gentoo-0.9.12.tar.gz`, bạn có thể dùng `gentoo` làm nguồn (source) **tên gói** và `0.9.12` là **phiên bản thượng nguồn**. Chúng được mô tả trong tệp `debian/changelog` chi tiết tại Phần 4.3.

Mặc dù phương pháp tiếp cận đơn giản này là chủ yếu, bạn có thể cần phải điều chỉnh **tên gói** và **phiên bản thượng nguồn** bằng cách đổi tên thượng nguồn theo Chính sách Debian và quy ước hiện hành.

Bạn phải chọn **tên gói** chỉ bao gồm các chữ thường (`a-z`), chữ số (`0-9`, dấu cộng (`+`) dấu trừ (`-`), và dấu chấm (`.`). Phải có ít nhất hai ký tự, phải bắt đầu bằng một ký tự chữ hoặc số, và không được giống với tên đã tồn tại. Bạn nên giữ chiều dài của nó trong vòng 30 ký tự. [11]

Nếu thượng nguồn sử dụng một số thuật ngữ chung chung như `test-suite` cho tên của nó, bạn nên đổi tên nó để xác định rõ ràng nội dung của nó và cũng tránh làm rối không gian tên. [12]

Bạn nên chọn **phiên bản thượng nguồn** chỉ bao gồm alphanumerics (`0-9A-Za-z`), dấu cộng (`+`), dấu ngã (`~`), và dấu chấm (`.`). Nó phải bắt đầu bằng một chữ số (`0-9`). [13]Tốt nhất nên giữ chiều dài của nó trong vòng 8 ký tự nếu có thể. [14]

Nếu thượng nguồn không sử dụng cấu trúc phiên bản bình thường như `2.30.32` mà lại sử dụng một số kiểu như `11Apr29`, một chuỗi tên mã ngẫu nhiên, hoặc một giá trị băm VCS như một phần của phiên bản, hãy chắc chắn loại bỏ chúng khỏi **phiên bản thượng nguồn**. Thông tin này có thể được ghi lại trong tệp `debian/changelog`. Nếu bạn cần tạo mới ra một chuỗi phiên bản, nên sử dụng định dạng `YYYYMMDD` như là `20110429` cho phiên bản thượng nguồn. Điều này đảm bảo rằng **dpkg** hiểu các phiên bản sau này chính xác như cho việc nâng cấp phiên bản. Nếu bạn cần đảm bảo chuyển đổi suôn sẻ sang cấu trúc phiên bản bình thường như `0.1` trong tương lai, hãy sử dụng định dạng `0~YYMMDD` như `0~110429` cho phiên bản thượng nguồn.

Các tên phiên bản [15] có thể được so sánh bằng cách sử dụng dpkg(1) như sau:

```
$ dpkg --compare-versions ver1 op ver2
```

Quy tắc so sánh phiên bản có thể được tóm tắt như sau:

- Chuỗi được so sánh từ đầu đến cuối chuỗi.
- Các chữ cái lớn hơn chữ số.
- Số được so sánh như số nguyên.
- Các chữ cái được so sánh theo thứ tự mã ASCII.
- Có các quy tắc đặc biệt cho các dấu chấm (`.`), cộng (`+`), ngã (`~`), như sau:
  `0.0` < `0.5` < `0.10` < `0.99` < `1` < `1.0~rc1` < `1.0` < `1.0+b1` < `1.0+nmu1` < `1.1` < `2.0`

Một trường hợp có thể xảy ra khi các bản phát hành thượng nguồn có phiên bản là `gentoo-0.9.12-ReleaseCandidate-99.tar.gz` nó là phiên bản trước của `gentoo-0.9.12.tar.gz`. Bạn cần đảm bảo rằng việc nâng cấp vẫn hoạt động bằng cách đổi tên nguồn thượng nguồn thành `gentoo-0.9.12~rc99.tar.gz`.

[10] Bạn có thể tự động hoá việc này bằng cách sử dụng gói `dh-autoreconf`. Xem Phần 4.4.3.

[11] Độ dài của trường tên gói mặc định trong **aptitude** là 30. Và có hơn 90% gói có độ dài tên dưới 24 ký tự.

[12] Nếu bạn theo dõi Debian Developer's Reference 5.1. "New packages" (http://www.debian.org/doc/developers-reference/pkgs.html#newpackage) , quá trình ITP thường sẽ bắt lỗi loại vấn đề này.

[13] Quy tắc chặt chẽ hơn này sẽ giúp bạn tránh nhầm lẫn tên tập tin.

[14] Độ dài của trường phiên bản mặc định trong **aptitude** là 10. Phiên bản Debian với dấu gạch nối trước thường tiêu tốn 2. Đối với hơn 80% gói, phiên bản thượng nguồn ít hơn 8 ký tự và phiên bản Debian có ít hơn 2 ký tự. Đối với hơn 90% gói, phiên bản thượng nguồn ít hơn 10 ký tự và phiên bản Debian ít hơn 3 ký tự.

[15] Các chuỗi tên phiên bản có thể là **upstream version** (*`version`*), **Debian revision** (*`revision`*), hoặc **version** (*`version-revision`*). Xem Phần 8.1 để biết cách mà **Debian revision** hoạt động

2.7 Thiết lập dh_make

Thiết lập các biến môi trường shell `$DEBEMAIL` và `$DEBFULLNAME` để các công cụ bảo trì Debian nhận biết địa chỉ email và tên của bạn cho việc sử dụng cho các gói .[16]

```
$ cat >>~/.bashrc <<EOF
DEBEMAIL="your.email.address@example.org"
DEBFULLNAME="Firstname Lastname"
export DEBEMAIL DEBFULLNAME
EOF
$ . ~/.bashrc
```

2.8 Khởi tạo gói Debian mới

Các gói Debian thường là các gói non-native Debian được tạo từ các chương trình thượng nguồn. Nếu bạn muốn tạo gói Debian mới từ nguồn của một nguồn thượng nguồn `gentoo-0.9.12.tar.gz`, bạn có thể tạo một gói Debian đầu tiên bằng cách sử dụng lệnh **dh_make** như sau:

```
$ cd ~/gentoo
$ wget http://example.org/gentoo-0.9.12.tar.gz
$ tar -xvzf gentoo-0.9.12.tar.gz
$ cd gentoo-0.9.12
$ dh_make -f ../gentoo-0.9.12.tar.gz
```

Tất nhiên, hãy thay tên tệp bằng tên của bản lưu trữ gốc của bạn. [17] Xem dh_make(8) để biết chi tiết.

Bạn sẽ thấy một số câu hỏi về loại gói bạn muốn tạo. Gentoo là một gói nhị phân duy nhất —nó chỉ tạo ra một gói nhị phân, tức chỉ cần một tập tin `.deb`—vì vậy chúng ta sẽ chọn tùy chọn đầu tiên (với `s`), kiểm tra thông tin trên màn hình và xác nhận bằng cách nhấn *`ENTER`*. [18]

Việc thực hiện **dh_make** sẽ tạo một bản sao của tarball thượng nguồn `gentoo_0.9.12.orig.tar.gz` trong thư mục cha để phù hợp với việc tạo ra gói nguồn non-native Debian với tên `debian.tar.gz` sau này:

```
$ cd ~/gentoo ; ls -F
gentoo-0.9.12/
gentoo-0.9.12.tar.gz
gentoo_0.9.12.orig.tar.gz
```

Vui lòng lưu ý hai điểm chính của tên tệp tin này `gentoo_0.9.12.orig.tar.gz`:

- Tên gói và phiên bản phân cách nhau bởi kí tự _ (gạch dưới)
- Thành phần `.orig` được chèn trước `.tar.gz`

[16] Ví dụ dưới đây giả định bạn đang sử dụng Bash là shell đăng nhập mặc định của bạn. Nếu bạn sử dụng một số shell đăng nhập khác như Z shell, hãy sử dụng các file cấu hình tương ứng thay vì `~/.bashrc`.

[17] Nếu thượng nguồn đã cung cấp thư mục `debian` và cả nội dung của nó, hãy chạy lệnh **dh_make** với tùy chọn thêm `--addmissing`. Định dạng nguồn `3.0 (quilt)` là đủ mạnh để không làm hỏng các gói này. Bạn có thể cần phải cập nhật các nội dung được cung cấp bởi phiên bản thượng nguồn cho gói Debian của bạn.

[18] Có vài sự lựa chọn bao gồm: `s` cho Single binary package, `i` cho arch-Independent package, `m` cho Multiple binary packages, `l` cho Library package, `k` cho Kernel module package, `n` cho kernel patch package, và `b` cho `cdbs` package. Tài liệu này tập trung vào sử dụng câu lệnh **dh** (từ gói `debhelper`) để tạo một single binary package, nhưng cũng nói tới làm sao sử dụng nó cho arch-independent hay multiple binary packages. Gói `cdbs` cung cấp các nền tảng kịch bản khác nhau cho lệnh **dh** và cả nằm ngoài phạm vi tài liệu này.

Bạn cũng nên lưu ý rằng nhiều tệp mẫu đã được tạo ra trong thư mục `debian` Chúng được giải thích ở Chương 4 và Chương 5. Bạn cũng nên hiểu rằng việc đóng gói không thể là một quá trình tự động hoàn toàn. Bạn sẽ cần phải chỉnh sửa thượng nguồn cho Debian (xem Chương 3). Sau đó, bạn phải cần sử dụng phương pháp thích hợp cho công việc xây dựng gói Debian (Chương 6), kiểm thử chúng (Chương 7), vả đẩy chúng lên máy chủ (Chương 9). Tất cả các bước sẽ được giải thích.

Nếu bạn vô tình xoá một số tệp khuôn mẫu trong khi đang làm việc với chúng, bạn có thể khôi phục chúng bằng cách chạy **dh_make** với tùy chọn `--addmissing` trong thư mục mã nguồn đang làm việc

Cập nhật cho một gói hiện sẽ có thể phức tạp vì nó có thể sử dụng các kỹ thuật cũ. Khi học những điều cơ bản, vui lòng gắn bó với việc tạo ra một gói theo kỹ thuật mới; Giải thích thêm được đưa ra trong Chương 8.

Vui lòng lưu ý rằng mã nguồn không cần chứa bất kỳ hệ thống xây dựng nào đã thảo luận trong Phần 2.4 và Phần 2.5. Nó có thể chỉ là một tập hợp các dữ liệu đồ hoạ,... Việc cài đặt các tệp có thể được chỉ thực hiện bằng các tập tin cấu hình `debhelper` như là `debian/install` (xem Phần 5.11).

Chapter 3

Sửa đổi mã nguồn

Xin lưu ý rằng không có nhiều thời gian để đi vào *tất cả* các chi tiết để khắc phục tất cả thượng nguồn, nhưng đây sẽ là một số bước cơ bản và vài vấn đề mà mọi người thường xuyên gặp phải.

3.1 Cài đặt quilt

Chương trình **quilt** cung cấp phương pháp cơ bản để ghi lại các sửa đổi đối với mã nguồn thượng nguồn cho gói Debian. Rất hữu ích khi có một mặc định hơi tùy chỉnh, vì vậy hãy tạo một alias **dquilt** cho gói Debian bằng cách thêm các dòng sau vào `~/.bashrc`. Dòng thứ hai là tính năng bash-completion gợi ý của của lệnh **quilt** vào lệnh **dquilt**:

```
alias dquilt="quilt --quiltrc=${HOME}/.quiltrc-dpkg"
complete -F _quilt_completion -o filenames dquilt
```

Sau đó hãy tạo tệp `~/.quiltrc-dpkg` như sau:

```
d=. ; while [ ! -d $d/debian -a $(readlink -e $d) != / ]; do d=$d/..; done
if [ -d $d/debian ] && [ -z $QUILT_PATCHES ]; then
    # if in Debian packaging tree with unset $QUILT_PATCHES
    QUILT_PATCHES="debian/patches"
    QUILT_PATCH_OPTS="--reject-format=unified"
    QUILT_DIFF_ARGS="-p ab --no-timestamps --no-index --color=auto"
    QUILT_REFRESH_ARGS="-p ab --no-timestamps --no-index"
    QUILT_COLORS="diff_hdr=1;32:diff_add=1;34:diff_rem=1;31:diff_hunk=1;33:diff_ctx=35: ↩
        diff_cctx=33"
    if ! [ -d $d/debian/patches ]; then mkdir $d/debian/patches; fi
fi
```

Xem quilt(1) và `/usr/share/doc/quilt/quilt.pdf.gz` để biết cách sử dụng **quilt**.

3.2 Sửa lỗi ở thượng nguồn

Hãy giả sử bạn đã tìm thấy một lỗi trong thượng nguồn `Makefile` như sau, dòng `install: gentoo` nên được sửa lại thành `install: gentoo-target`.

```
install: gentoo
        install ./gentoo $(BIN)
        install icons/* $(ICONS)
        install gentoorc-example $(HOME)/.gentoorc
```

Hãy sửa và ghi lại nó bằng lệnh **dquilt** với bản vá mới là `fix-gentoo-target.patch`: [1]

```
$ mkdir debian/patches
$ dquilt new fix-gentoo-target.patch
$ dquilt add Makefile
```

Bạn thay đổi tệp tin `Makefile` như sau

```
install: gentoo-target
        install ./gentoo $(BIN)
        install icons/* $(ICONS)
        install gentoorc-example $(HOME)/.gentoorc
```

Lệnh **dquilt** sẽ tạo ra bản vá `debian/patches/fix-gentoo-target.patch` và thêm mô tả của nó như sau DEP-3: Patch Tagging Guidelines (http://dep.debian.net/deps/dep3/) :

```
$ dquilt refresh
$ dquilt header -e
... mô tả bản vá
```

3.3 Cài đặt các tệp tin đến đích của chúng

Hầu hết các phần mềm của bên thứ ba cài đặt chính nó vào hệ thống phân cấp thư mục `/usr/local`. Với Debian, người quản trị hệ thống nên dành riêng cho việc sử dụng cá nhân, vì vậy các gói không được sử dụng các thư mục như `/usr/local/bin` nhưng nên sử dụng thư mục hệ thống như `/usr/bin`, tuân theo Tiêu chuẩn Phân cấp Hệ thống Tập tin(Filesystem Hierarchy Standard) (http://www.debian.org/doc/packaging-manuals/fhs/fhs-3.0.html) (FHS).

Thông thường, make(1) được sử dụng để tự động xây dựng chương trình và thực hiện `make install` để cài đặt các chương trình trực tiếp tới đích mong muốn (sau phần `install` là target trong `Makefile`). Để Debian tạo ra các gói trước khi cài đặt, nó sẽ sửa đổi hệ thống xây dựng để cài đặt các chương trình vào cây thư mục giả định dưới một thư mục tạm thay vì một đích thực.

Hai khác biệt giữa cài đặt chương trình bình thường và hệ thống đóng gói Debian có thể được hiểu rõ bởi gói `debhelper`qua lệnh **dh_auto_configure** và **dh_auto_install** nếu đáp ứng các điều kiện sau:

- Tệp `Makefile` phải tuân theo các quy ước GNU và hỗ trợ biến `$(DESTDIR)` . [2]
- Mã nguồn phải tuân theo Filesystem Hierarchy Standard (FHS)

Các chương trình sử dụng GNU **autoconf** tuân theo các quy ước của GNU một cách tự động, do đó chúng rất dễ đóng gói. Trên cơ sở điều này cùng các phương pháp chẩn đoán khác, người ta ước tính rằng `debhelper` sẽ hoạt động với khoảng 90% gói mà không làm thay đổi gì đối hệ thống xây dựng của thượng nguồn. Vì vậy, việc đóng gói không phải là phức tạp như bạn thấy

[1] Thư mục `debian/patches` sẽ tồn tại nếu bạn đã chạy **dh_make** như đã mô tả trước. Thao tác ví dụ này cũng tạo ra nhưng chỉ trong trường hợp bạn đang cập nhật một gói hiện có.

[2] Xem GNU Coding Standards: 7.2.4 DESTDIR: Support for Staged Installs (http://www.gnu.org/prep/standards/html_node/DESTDIR.html#DESTDIR) .

Nếu bạn cần thực hiện thay đổi tệp `Makefile`, bạn nên cẩn thận hỗ trợ biến `$(DESTDIR)`. Mặc dù nó bị bỏ bởi mặc định, biến `$(DESTDIR)` được thêm vào cho mỗi đường dẫn tập tin để cài đặt chương trình. Kịch bản đóng gói sẽ đặt `$(DESTDIR)` là thư mục tạm thời.

Đối với một gói nguồn tạo ra một gói nhị phân duy nhất, thư mục tạm thời được sử dụng bởi lệnh **dh_auto_install** sẽ được cài đặt vào `debian/`*`package`*. [3] Tất cả mọi thứ có trong thư mục tạm thời sẽ được cài đặt trên hệ thống của người dùng khi họ cài đặt gói của bạn; Khác biệt duy nhất là **dpkg** sẽ cài đặt các tập tin vào đường dẫn liên quan đến thư mục gốc thay vì thư mục làm việc của bạn.

Lưu ý rằng ngay cả khi chương trình của bạn cài đặt trong `debian/`*`package`*, nó vẫn cần phải hoạt động chính xác khi cài đặt từ tệp `.deb` vào thư mục gốc. Vì vậy bạn không được tạo hệ thống xây dựng cài đặt vào như đường dẫn tuyệt đối như `/home/me/deb/`*`package-version`*`/usr/share/`*`package`* ở các tệp tin trong gói.

Đây là một phần tệp tin `Makefile` của `gentoo`[4]:

```
# Where to put executable commands on 'make install'?
BIN     = /usr/local/bin
# Where to put icons on 'make install'?
ICONS   = /usr/local/share/gentoo
```

Chúng ta thấy rằng các tệp được cài đặt vào `/usr/local`. Như đã giải thích ở trên, cây thư mục đó được dành riêng cho việc sử dụng cục bộ trên Debian, do đó cần thay đổi những đường dẫn đó như sau:

```
# Where to put executable commands on 'make install'?
BIN     = $(DESTDIR)/usr/bin
# Where to put icons on 'make install'?
ICONS   = $(DESTDIR)/usr/share/gentoo
```

Vị trí chính xác cần được sử dụng cho các tệp nhị phân, icons, tài liệu, v.v. được xác định trong Filesystem Hierarchy Standard (FHS). Bạn nên đọc qua nó và các phần có liên quan đến gói của bạn.

Vậy, chúng ta nên cài đặt chương trình vào `/usr/bin` thay vì `/usr/local/bin`, trang hướng dẫn vào `/usr/share/man/man1` thay vì `/usr/local/man/man1`, và vân vân. Chú ý là không có trang hướng dẫn trong `Makefile` của `gentoo`, nhưng vì Debian Policy đòi hỏi mọi chương trình phải có, chúng ta sẽ làm sau và sẽ cài đặt nó trong `/usr/share/man/man1`.

Một số chương trình không sử dụng biến trong `Makefile` để xác định các đường dẫn như vậy. Điều này có nghĩa là bạn có thể phải chỉnh sửa một số mã nguồn C để khắc phục chúng giúp sử dụng đúng vị trí. Nhưng tìm kiếm ở đâu, và chính xác những gì? Bạn có thể tìm ra điều này bằng cách thực hiện:

```
$ grep -nr --include='*.[c|h]' -e 'usr/local/lib' .
```

lệnh **grep** này sẽ tìm đệ quy trong cây mã nguồn và cho bạn tên tập tin và vị trí dòng chứa cụm từ `usr/local/lib`.

Sửa đổi các tệp và thay thế `usr/local/lib` bằng `usr/lib`. Điều này có thể thực hiện tự động như sau:

```
$ sed -i -e 's#usr/local/lib#usr/lib#g' \
        $(find . -type f -name '*.[c|h]')
```

Nếu bạn muốn xác nhận của mỗi lần thay thế thay, điều này có thể được thực hiện như sau:

[3] Với một gói mã nguồn tạo ra nhiều gói nhị phân, lệnh **dh_auto_install** sử dụng thư mục `debian/tmp` như là thư mục tạm thời trong khi lệnh **dh_install** sẽ tạo ra các kịch bản `debian/`*`package-1`*`.install` và `debian/`*`package-2`*`.install` sẽ chia nội dung của `debian/tmp` vào thư mục tạm thời là `debian/`*`package-1`* và `debian/`*`package-2`*, để tạo gói nhị phân *`package-1`*`_*.deb` và *`package-2`*`_*.deb`.

[4] Đây chỉ là một ví dụ để xem `Makefile` trông như thế nào. Nếu `Makefile` được tạo bởi lệnh **./configure**, cách chính xác để sửa lỗi trên `Makefile` là thực thi **./configure** từ lệnh **dh_auto_configure** với các tùy chọn mặc định bao gồm `--prefix =/usr` .

```
$ vim '+argdo %s#usr/local/lib#usr/lib#gce|update' +q \
        $(find . -type f -name '*.[c|h]')
```

Tiếp theo, bạn nên tìm mục `install` (tìm kiếm dòng bắt đầu bằng `install:` thường sẽ có) và đổi tên tất cả các tham chiếu tới các thư mục khác. Được xác định ở trên cùng của `Makefile`.

Bản gốc ban đầu, mục `install` của `gentoo` như sau:

```
install: gentoo-target
        install ./gentoo $(BIN)
        install icons/* $(ICONS)
        install gentoorc-example $(HOME)/.gentoorc
```

Hãy sửa lỗi thượng nguồn này và ghi lại nó bằng lệnh **dquilt** dưới dạng `debian/patches/install.patch`.

```
$ dquilt new install.patch
$ dquilt add Makefile
```

Trong trình soạn thảo của bạn, thay đổi này cho gói Debian như sau:

```
install: gentoo-target
        install -d $(BIN) $(ICONS) $(DESTDIR)/etc
        install ./gentoo $(BIN)
        install -m644 icons/* $(ICONS)
        install -m644 gentoorc-example $(DESTDIR)/etc/gentoorc
```

Bạn sẽ nhận thấy rằng hiện có một lệnh `install -d` trước các lệnh khác trong quy tắc. Tệp `Makefile` ban đầu không có nó bởi vì thường thư mục `/usr/local/bin` và các thư mục khác đã tồn tại trên hệ thống mà bạn đang chạy `make install`. Tuy nhiên, khi chúng ta sẽ cài đặt vào một cây thư mục cá nhân mới, chúng ta sẽ phải tạo ra một trong những thư mục đó.

Chúng ta cũng có thể thêm vào những thứ khác ở cuối quy tắc, như việc cài đặt các tài liệu bổ sung mà các tác giả thượng nguồn đôi khi bỏ qua:

```
        install -d $(DESTDIR)/usr/share/doc/gentoo/html
        cp -a docs/* $(DESTDIR)/usr/share/doc/gentoo/html
```

Kiểm tra cẩn thận, và nếu mọi thứ đều ổn, hãy yêu cầu **dquilt** tạo ra bản vá vào `debian/patches/install.patch` và thêm mô tả cho nó:

```
$ dquilt refresh
$ dquilt header -e
... mô tả bản vá
```

Bây giờ bạn có một loạt các bản vá.

1. Bản vá thượng nguồn: `debian/patches/fix-gentoo-target.patch`
2. Bản vá việc đóng gói riêng cho Debian: `debian/patches/install.patch`

Bất cứ khi nào bạn thực hiện các thay đổi không phải đặc trưng cho gói Debian như `debian/patches/fix-gentoo-target.patch`, hãy chắc chắn gửi chúng đến người bảo trì thượng nguồn để có thể được áp dụng vào trong phiên bản kế tiếp của chương trình, sẽ là hữu ích cho mọi người khác. Đồng thời nhớ tránh tạo các bản sửa lỗi cụ thể cứng nhắc cho Debian hoặc Linux —hoặc thậm chí là Unix! Hãy làm cho chúng di động. Điều này sẽ làm cho các bản sửa lỗi của bạn dễ dàng hơn để áp dụng.

Nhớ rằng bạn không phải gửi toàn bộ tệp `debian/*` tới thượng nguồn.

3.4 Thư viện khác

Có một vấn đề phổ biến khác: các thư viện thường khác nhau từ nền tảng này đến nền tảng khác. Ví dụ, một tệp `Makefile` có thể chứa một tham chiếu đến thư viện không tồn tại trên hệ thống Debian. Trong trường hợp đó, chúng ta cần thay đổi nó thành một thư viện tồn tại trong Debian nhưng phục vụ cùng một mục đích.

Hãy giả sử một dòng trong `Makefile` của chương trình của bạn (hoặc `Makefile.in`) như sau.

```
LIBS = -lfoo -lbar
```

Nếu chương trình không biên dịch được khi thư viện `foo` không tồn tại và nó tương đương với thư viện `foo2` được cung cấp trên hệ thống Debian , bạn có thể sửa vấn đề này với `debian/patches/foo2.patch` bằng cách thay đổi từ `foo` sang `foo2`:[5]

```
$ dquilt new foo2.patch
$ dquilt add Makefile
$ sed -i -e ’s/-lfoo/-lfoo2/g’ Makefile
$ dquilt refresh
$ dquilt header -e
... mô tả bản vá
```

[5] Nếu có sự thay đổi API từ thư viện `foo` sang thư viện `foo2`, yêu cầu thay đổi mã nguồn để khớp với API mới.

Chapter 4

Các tệp yêu cầu trong thư mục `debian`

Có một thư mục con mới trong thư mục nguồn của chương trình được gọi là `debian`. Có một số tệp tin trong thư mục này mà chúng ta nên sửa đổi để các tuỳ chỉnh hành vi của gói. Các tệp quan trọng trong số chúng là `control`, `changelog`, `copyright`, và `rules`, chúng là những tệp yêu cầu đối với tất cả các gói [1]

4.1 `control`

Tập tin này chứa các giá trị khác nhau mà **dpkg**, **dselect**, **apt-get**, **apt-cache**, **aptitude**, và các công cụ quản lý gói khác sẽ sử dụng để quản lý gói. Nó được xác định bởi Debian Policy Manual, 5 "Control files and their fields" (http://www.debian.org/doc/-debian-policy/ch-controlfields.html)

Đây là tệp `control` mà **dh_make** tạo ra cho chúng ta:

```
 1 Source: gentoo
 2 Section: unknown
 3 Priority: optional
 4 Maintainer: Josip Rodin <joy-mg@debian.org>
 5 Build-Depends: debhelper (>=10)
 6 Standards-Version: 4.0.0
 7 Homepage: <insert the upstream URL, if relevant>
 8
 9 Package: gentoo
10 Architecture: any
11 Depends: ${shlibs:Depends}, ${misc:Depends}
12 Description: <insert up to 60 chars description>
13  <insert long description, indented with spaces>
```

(Tôi đã thêm các số dòng.)

Dòng 1–7 là thông tin điều khiển cho gói nguồn. Dòng 9–13 là thông tin điều khiển cho gói nhị phân.

Dòng 1 là tên của gói nguồn

Dòng 2 là các mục loại mục chương trình

Như bạn có thể nhận thấy, kho phần mềm Debian được chia làm nhiều khu vực: `main` (phần mềm tự do), `non-free` (thật sự không phải phần mềm tự do) and `contrib` (phần mềm tự do phụ thuộc vào phần mềm không tự do). Mỗi loại được chi thánh các phần danh mục. Vậy chúng ta có `admin` cho các chương trình quản trị, `devel` cho các công cụ phát triển, `doc` cho tài liệu,

[1] Trong chương này, các tệp tin trong thư mục `debian` được đề cập đến mà không có `debian/` để đơn giản hóa bất cứ khi nhắc tới.

`libs` cho thư viện, `mail` cho chương trình liên quan thư điện tử, `net` cho ứng dụng mạng và chạy nền, `x11` cho các chương trình X11 khác, và nhiều nữa. [2]

Chúng ta hãy thay đổi nó thành x11. (Tiền tố `main/` với ngụ ý là chúng ta có thể bỏ qua nó).

Dòng 3 miêu tả mức độ quan trọng của việc ngươi dùng cài đặt gói này. [3]

- Ưu tiên `optional` thường sẽ làm việc với gói mới mà không xung đột với các gói chứa ưu tiên `required`, `important`, hay `standard` .

Trường Section và Priority được sử dụng bởi các chương trình front-ends như **aptitude** khi họ sắp xếp gói và chọn mặc định. Khi bạn tải gói lên Debian, giá trị của hai trường này có thể bị ghi đè bởi các nhà bảo trì kho lưu trữ, trong trường hợp đó bạn sẽ được thông báo bằng email.

Vì đây là gói ưu tiên thông thường và không xung đột với bất kỳ điều gì khác, chúng tôi sẽ thay đổi mức độ ưu tiên thành `optional`.

Dòng 4 là tên và địa chỉ email của người bảo trì. Hãy đảm bảo rằng trường này có tiêu đề `To` hợp lệ cho email, bởi vì sau khi tải lên, hệ thống theo dõi lỗi sẽ sử dụng chính trường này để gửi email lỗi cho bạn. Tránh sử dụng dấu phẩy, ký hiệu, hoặc dấu ngoặc đơn.

Dòng 5 là trưởng `Build-Depends` bao gồm danh sách của các gói yêu cầu để xây dựng gói của bạn. Bạn cũng có thể có thêm trường `Build-Depends-Indep` bổ sung. [4]. Một số gói như `gcc` và `make` là yêu cầu ngầm định bởi gói `build-essential` Nếu bạn cần có các công cụ khác để xây dựng gói của mình, bạn nên thêm chúng vào các trường này. Nhiều mục được phân cách bằng dấu phẩy; Hãy đọc tiếp để hiểu rõ hơn về các phụ thuộc gói nhị phân.

- Đối với tất cả các gói đóng gói với lệnh **dh** trong tệp tin `debian/rules`, bạn phải có `debhelper(>=9)` trong trường `Build-Depends` để thỏa mãn yêu cầu Chính sách Debian đối với mục `clean`.
- Gói nguồn có gói nhị phân với `Architecture: any` được xây dựng lại bởi autobuilder. Vì thủ tục autobuilder này chỉ cài đặt các gói được liệt kê trong trường `Build-Depends` trước khi chạy `debian/rules build` (see Phần 6.2), Trường `Build-Depends` rất cần thiết để liệt kê tất cả các gói cần thiết còn `Build-Depends-Indep` hiếm khi được sử dụng.
- Đối với các gói nguồn với các gói nhị phân có kiến trúc `Architecture: all`, trường `Build-Depends-Indep` có thể liệt kê tất cả các gói cần thiết trừ khi chúng đã được liệt kê trong trường `Build-Depends` để thỏa mãn yêu cầu của Chính sách Debian đối với mục `clean`.

Nếu bạn không chắc chắn nên sử dụng trường nào, hãy sử dụng trường `Build-Depends` để an toàn. [5]

Để tìm ra những gói nào cần để xây dựng chương trình của bạn:

```
$ dpkg-depcheck -d ./configure
```

Để tự tìm chính xác các phụ thuộc khi xây dựng ***/usr/bin/foo***, thực hiện

```
$ objdump -p /usr/bin/foo | grep NEEDED
```

và cho mỗi thư viện được liệt kê (ví dụ: **libfoo.so.6**), thực hiện

[2] Xem Debian Policy Manual, 2.4 "Sections" (http://www.debian.org/doc/debian-policy/ch-archive.html#s-subsections) và List of sections in `sid` (http://packages.debian.org/unstable/) .

[3] Xem Debian Policy Manual, 2.5 "Priorities" (http://www.debian.org/doc/debian-policy/ch-archive.html#s-priorities) .

[4] Xem Debian Policy Manual, 7.7 "Relationships between source and binary packages - Build-Depends, Build-Depends-Indep, Build-Conflicts, Build-Conflicts-Indep" (http://www.debian.org/doc/debian-policy/ch-relationships.html#s-sourcebinarydeps) .

[5] Tình huống này hơi lạ là một tính năng được ghi lại trong Debian Policy Manual, Footnotes 55 (http://www.debian.org/doc/debian-policy/-footnotes.html#f55) . Đây không phải là do sử dụng lệnh **dh** trong tệp tin `debian/rules` nhưng do cách **dpkg-buildpackage** làm việc. Tình huống tương tự cũng áp dụng cho hệ thống xây dựng tự động cho Ubuntu (https://bugs.launchpad.net/launchpad-buildd/+bug/238141) .

```
$ dpkg -S libfoo.so.6
```

Sau đó chỉ cần lấy phiên bản `-dev` của mỗi gói như một mục nhập `Build-Depends`. Nếu bạn sử dụng **ldd** cho mục đích này, nó sẽ báo cáo phụ thuộc lib gián tiếp, kết quả là

`gentoo` yêu cầu `xlibs-dev`, `libgtk1.2-dev` and `libglib1.2-dev` để xây dựng, vậy nên chúng ta sẽ thêm chúng vào vị trí tiếp theo `debhelper`.

Dòng 6 là phiên bản của Debian Policy Manual (http://www.debian.org/doc/devel-manuals#policy) mà phần mềm này tuân theo, một trong tài liệu bạn đọc trong khi xây dựng gói của bạn.

Trên dòng 7, bạn có thể đặt URL của trang chủ thượng nguồn của phần mềm

Dòng 9 là tên của gói nhị phân. Nó thường giống với tên của gói nguồn, nhưng nó không nhất thiết phải giống nhau.

Dòng 10 mô tả các kiến trúc mà gói nhị phân có thể được biên dịch. Giá trị này thường là một trong các tùy chọn sau, tùy thuộc vào loại gói nhị phân: [6]

- `Architecture: any`
  - Gói nhị phân được tạo ra là một kiến trúc phụ thuộc thường là các chương trình xây dựng bằng ngôn ngữ biên dịch.
- `Architecture: all`
  - Gói nhị phân được tạo ra là một kiến trúc độc lập thường bao gồm các văn bản, hình ảnh hoặc các tập lệnh theo ngôn ngữ kịch bản.

Chúng ta để lại dòng 10 vì nó được viết bằng C. dpkg-gencontrol(1) sẽ tự điền vào trường Architecture một cách thích hợp cho bất kỳ máy nào gói nguồn này được biên soạn.

Nếu gói của bạn là kiến trúc độc lập (ví dụ: mã shell hoặc kịch bản Perl, hoặc tài liệu), hãy thay đổi điều này thành `all` và đọc Phần 4.4 về `binary-indep` thay vì `binary-arch` để xây dựng gói.

Dòng 11 cho thấy một trong những tính năng mạnh mẽ nhất của hệ thống đóng gói Debian. Gói có thể liên quan đến nhau theo nhiều cách khác nhau. Ngoài `Depends`, các trường mối quan hệ khác là `Recommends`, `Suggests`, `Pre-Depends`, `Breaks`, `Conflicts`, `Provides` và `Replaces`.

Các công cụ quản lý gói thường có chung hành vi khi đọc các mối quan hệ này; Nếu không, nó sẽ được giải thích. (See dpkg(8), dselect(8), apt(8), aptitude(1), etc.)

Đây là mô tả đơn giản về các mối quan hệ của gói: [7]

- `Depends`

  Gói của bạn sẽ không được cài đặt trừ khi các gói phụ thuộc được cài đặt. Sử dụng trường này nếu chương trình của bạn hoàn toàn không chạy (hoặc sẽ gây ra sự cố nghiêm trọng) trừ khi một gói cụ thể có mặt.

- `Recommends`

  Sử dụng trường này cho cho các gói không bắt buộc nhưng thường được sử dụng kèm với chương trình của bạn. Khi người dùng cài đặt chương trình của bạn, các công cụ quản lý gói có thể sẽ nhắc cài đặt gói đề xuất. **aptitude** và **apt-get** cài đặt các gói được đề xuất cùng với gói của bạn theo mặc định (nhưng người dùng có thể vô hiệu hóa hành vi này). **dpkg** sẽ bỏ qua trường này.

- `Suggests`

  Sử dụng trường này cho các gói sẽ hoạt động tốt với chương trình của bạn nhưng không cần thiết. Khi người dùng cài đặt chương trình của bạn, họ sẽ không được nhắc nhở để cài đặt gói đề xuất. **aptitude** có thể được cấu hình để cài đặt các gói đề xuất cùng với gói của bạn nhưng đây không phải là mặc định. **dpkg** và **apt-get** sẽ bỏ qua trường này.

[6] Xem Debian Policy Manual, 5.6.8 "Architecture" (http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Architecture) để biết chi tiết.

[7] Xem Debian Policy Manual, 7 "Declaring relationships between packages" (http://www.debian.org/doc/debian-policy/ch-relationships.html) .

- `Pre-Depends`

  Trường này mạnh hơn `Depends`. Gói sẽ không được cài đặt trừ khi các gói mà nó được phụ thuộc trước được cài đặt và *được cấu hình chính xác*. Sử dụng điều này một cách *rất* tỉ mỉ và chỉ sau khi thảo luận về nó trên danh sách gửi thư debian-devel@lists.debian.org (http://lists.debian.org/debian-devel/) . Đọc: đừng sử dụng nó.:-)

- `Conflicts`

  Gói sẽ không được cài đặt cho đến khi tất cả các gói xung đột đã được gỡ bỏ. Sử dụng này nếu chương trình của bạn hoàn toàn không chạy hoặc sẽ gây ra những vấn đề nghiêm trọng nếu một gói cụ thể có mặt.

- `Breaks`

  Khi cài đặt gói nó sẽ phá vỡ tất cả các gói được liệt kê. Thông thường trường `Breaks` chỉ định áp dụng các phiên bản sớm hơn một giá trị nhất định. Nói chung là sử dụng các công cụ quản lý gói cấp cao để nâng cấp các gói được liệt kê.

- `Provides`

  Đối với một số loại gói có nhiều tên ảo khác đã được xác định. Bạn có thể có một danh sách đầy đủ trong tệp virtual-package-names-list.txt.gz (http://www.debian.org/doc/packaging-manuals/virtual-package-names-list.txt) . Sử dụng điều này nếu chương trình của bạn cung cấp một chức năng của một gói ảo hiện có.

- `Replaces`

  Sử dụng điều này khi chương trình của bạn thay thế các tệp từ một gói khác hoặc thay thế hoàn toàn một gói khác (được sử dụng kết hợp với `Conflicts`). Các tệp từ các gói có tên sẽ được ghi đè bằng các tệp từ gói của bạn.

Tất cả các trường này có cú pháp thống nhất. Họ là một danh sách các tên gói được phân cách bằng dấu phẩy. Các tên gói này cũng có thể là danh sách các tên gói thay thế, được phân cách bằng ký hiệu thanh dọc `|` (biểu tượng ống).

Các trường có thể hạn chế khả năng áp dụng của chúng đối với các phiên bản cụ thể của từng gói có tên. Hạn chế của từng gói riêng lẻ được liệt kê trong dấu ngoặc đơn sau tên của nó và phải có một mối quan hệ từ danh sách dưới đây và theo sau là một giá trị số phiên bản. Các mối quan hệ được cho phép là: `<<`, `<=`, `=`, `>=`, và `>>` tương ứng với bé hơn, bé hơn hoặc bằng, chính xác bằng, lớn hơn hoặc bằng, và lớn hơn. Ví dụ,

```
Depends: foo (>= 1.2), libbar1 (= 1.3.4)
Conflicts: baz
Recommends: libbaz4 (>> 4.0.7)
Suggests: quux
Replaces: quux (<< 5), quux-foo (<= 7.6)
```

Tính năng cuối cùng bạn cần biết là về `${shlibs:Depends}`, `${perl:Depends}`, `${misc:Depends}`, vân vân.

dh_shlibdeps(1) tính toán sự phụ thuộc vào các thư viện chia sẻ cho các gói nhị phân. Nó sinh ra một danh sách của tệp thực thi ELF và thư viện chia sẻ mà nó đã tìm cho mỗi gói nhị phân. Danh sách này được sử dụng để thay thế `${shlibs:Depends}`.

dh_perl(1) tính toán sự phụ thuộc Perl. Nó sinh ra một danh sách của các phụ thuộc trong `perl` hoặc `perlapi` cho mỗi gói nhị phân. Danh sách này được sử dụng thay thế cho `${perl:Depends}`.

Một vài lệnh `debhelper` có thể làm cho gói được sinh ra phụ thuộc vào vài gói mở rộng thêm. Tất cả các lệnh như vậy sinh ra một danh sách của các gói cho mỗi gói nhị phân. Danh sách này được sử dụng để thay thế cho `${misc:Depends}`.

dh_gencontrol(1) tạo ra `DEBIAN/control` cho mỗi gói nhị phân trong khi thay thế `${shlibs:Depends}`, `${perl:Depends}`, `${misc:Depends}`, vân vân.

Có tất cả những điều đó, chúng ta có thể để trường `Depends` chính xác như bây giờ và chèn một dòng sau là `Suggests: file`, bởi vì `gentoo` có thể sử dụng một số tính năng được cung cấp bởi gói `file` .

Dòng 9 là URL Trang chủ. Giả sử rằng đây là http://www.obsession.se/gentoo/.

Dòng 12 là mô tả ngắn. Thiết bị đầu cuối thông thường có 80 cột nên không được dài hơn 60 ký tự. Tôi sẽ thay đổi nó thành `fully GUI-configurable, two-pane X file manager`

Dòng 13 là nơi diễn tả dài. Đây sẽ là một đoạn cho biết thêm chi tiết về gói. Cột 1 của mỗi dòng nên để trống. Không có dòng trắng, nhưng bạn có thể đặt một `.` (dấu chấm) trong một cột để mô phỏng. Ngoài ra, không có nhiều dòng trống sau mô tả dài. [8]

Chúng ta có thể chèn các trường `Vcs-*` để ghi lại thông tin của VCS trong các dòng 6 và 7. [9] Hãy giả sử rằng gói `gentoo` có VCS của nó nằm trong Debian Alioth Git Service ở `git://git.debian.org/git/collab-maint/gentoo.git`.

Cuối cùng, đây là tệp `control` đã được cập nhật

```
 1 Source: gentoo
 2 Section: x11
 3 Priority: optional
 4 Maintainer: Josip Rodin <joy-mg@debian.org>
 5 Build-Depends: debhelper (>=10), xlibs-dev, libgtk1.2-dev, libglib1.2-dev
 6 Standards-Version: 4.0.0
 7 Vcs-Git: https://anonscm.debian.org/git/collab-maint/gentoo.git
 8 Vcs-browser: https://anonscm.debian.org/git/collab-maint/gentoo.git
 9 Homepage: http://www.obsession.se/gentoo/
10
11 Package: gentoo
12 Architecture: any
13 Depends: ${shlibs:Depends}, ${misc:Depends}
14 Suggests: file
15 Description: fully GUI-configurable, two-pane X file manager
16  gentoo is a two-pane file manager for the X Window System. gentoo lets the
17  user do (almost) all of the configuration and customizing from within the
18  program itself. If you still prefer to hand-edit configuration files,
19  they're fairly easy to work with since they are written in an XML format.
20  .
21  gentoo features a fairly complex and powerful file identification system,
22  coupled to an object-oriented style system, which together give you a lot
23  of control over how files of different types are displayed and acted upon.
24  Additionally, over a hundred pixmap images are available for use in file
25  type descriptions.
26  .
29  gentoo was written from scratch in ANSI C, and it utilizes the GTK+ toolkit
30  for its interface.
```

(Tôi đã thêm các số dòng.)

4.2 `copyright`

Tập tin này chứa thông tin về bản quyền và giấy phép của các nguồn thượng nguồn Debian Policy Manual, 12.5 "Copyright information" (http://www.debian.org/doc/debian-policy/ch-docs.html#s-copyrightfile) kiểm soát nội dung của nó và DEP-5: Machine-parseable `debian/copyright` (http://dep.debian.net/deps/dep5/) cung cấp các chỉ dẫn cho định dạng của nó

dh_make có thể cho bạn một tệp mẫu `copyright` . Hãy sử dụng tuỳ chọn `--copyright gpl2` để lấy tệp mẫu cho gói `gentoo` phát hành dưới giấy phép GPL-2.

Bạn phải điền thông tin còn thiếu để hoàn thành tệp này, chẳng hạn như nơi tạo ra gói ,thông báo bản quyền thực tế, và giấy phép. Đối với một số giấy phép phần mềm tự do thông thường (GNU GPL-1, GNU GPL-2, GNU GPL-3, LGPL-2, LGPL-2.1, LGPL-3, GNU FDL-1.2, GNU FDL-1.3, Apache-2.0, hoặc Artistic license), bạn chỉ cần tham khảo các tệp thích hợp trong thư mục `/usr/share/common-licenses/` tồn tại trên bất kỳ hệ thống Debian. Nếu không, bạn phải bao gồm giấy phép đầy đủ.

[8] Những mô tả này bằng tiếng Anh. Các bản dịch về các mô tả này được cung cấp bởi The Debian Description Translation Project - DDTP (http://www.debian.org/intl/l10n/ddtp) .

[9] Xem Debian Developer's Reference, 6.2.5. "Version Control System location" (http://www.debian.org/doc/manuals/developers-reference/best-pkging-practices.html#bpp-vcs) .

Tóm lại, tệp `copyright` của gói `gentoo` trông sẽ như sau:

```
 1 Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
 2 Upstream-Name: gentoo
 3 Upstream-Contact: Emil Brink <emil@obsession.se>
 4 Source: http://sourceforge.net/projects/gentoo/files/
 5
 6 Files: *
 7 Copyright: 1998-2010 Emil Brink <emil@obsession.se>
 8 License: GPL-2+
 9
10 Files: icons/*
11 Copyright: 1998 Johan Hanson <johan@tiq.com>
12 License: GPL-2+
13
14 Files: debian/*
15 Copyright: 1998-2010 Josip Rodin <joy-mg@debian.org>
16 License: GPL-2+
17
18 License: GPL-2+
19  This program is free software; you can redistribute it and/or modify
20  it under the terms of the GNU General Public License as published by
21  the Free Software Foundation; either version 2 of the License, or
22  (at your option) any later version.
23  .
24  This program is distributed in the hope that it will be useful,
25  but WITHOUT ANY WARRANTY; without even the implied warranty of
26  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
27  GNU General Public License for more details.
28  .
29  You should have received a copy of the GNU General Public License along
30  with this program; if not, write to the Free Software Foundation, Inc.,
31  51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
32  .
33  On Debian systems, the full text of the GNU General Public
34  License version 2 can be found in the file
35  '/usr/share/common-licenses/GPL-2'.
```

(Tôi đã thêm các số dòng.)

Hãy làm theo HOWTO cung cấp bởi ftpmasters và gửi tới debian-devel-announce: http://lists.debian.org/debian-devel-announce/-2006/03/msg00023.html.

4.3 `changelog`

Đây là tệp yêu cầu, có định dạng đặc biệt được mô tả trong Debian Policy Manual, 4.4 "debian/changelog" (http://www.debian.org/-doc/debian-policy/ch-source.html#s-dpkgchangelog) . Định dạng này được sử dụng bởi **dpkg** và các chương trình khác để đọc số phiên bản, số sửa đổi, bản phân phối và tính khẩn cấp của gói

Đối với bạn, điều này cũng rất quan trọng vì bạn nên ghi lại tất cả những thay đổi bạn đã làm. Nó sẽ giúp mọi người tải gói của bạn và xem liệu có vấn đề gì với gói mà họ cần biết hay không. Nó sẽ được lưu lại như `/usr/share/doc/gentoo/changelog.Debian.gz` trong gói nhị phân.

dh_make tạo ra tệp mặc định, và nó sẽ trông như sau:

```
1  gentoo (0.9.12-1) unstable; urgency=medium
2
```

```
3   * Initial release (Closes: #nnnn)  <nnnn is the bug number of your ITP>
4
5  -- Josip Rodin <joy-mg@debian.org>  Mon, 22 Mar 2010 00:37:31 +0100
6
```

(Tôi đã thêm các số dòng.)

Dòng 1 là tên gói, phiên bản, phân phối và mức độ khẩn cấp. Tên phải khớp với tên gói nguồn; Phân phối nên là `unstable`, và nên cài mức độ khẩn cấp lên medidum (trung bình) trừ khi có bất kỳ lý do cụ thể nào cho giá trị khác.

Dòng 3-5 là một mục thông tin, nơi bạn ghi lại những thay đổi được thực hiện trong bản sửa đổi gói này (không phải là những thay đổi ở thượng nguồn —có một tệp tin đặc biệt cho mục đích đó được tạo ra bởi các tác giả thượng nguồn, mà sau này bạn sẽ cài đặt dưới dạng `/usr/share/doc/gentoo/changelog.gz`). Hãy giả sử báo lỗi ITP (Intent To Package) của bạn có số là `12345`. Các dòng mới chèn trên dòng trên cùng nhất của mục nội dung này và bắt đầu với * (dấu sao). Bạn có thể thực hiện điều này với dch(1), bạn có thể chỉnh sửa bằng tay với một trình soạn thảo văn bản theo quy ước định dạng được sử dụng bởi dch(1).

Để ngăn chặn một gói bị vô tình tải lên trước khi hoàn thành gói, bạn nên thay đổi giá trị phân phối thành chưa phát hành `UNRELEASED`.

Bạn sẽ kết thúc với một cái gì đó như thế này:

```
1  gentoo (0.9.12-1) UNRELEASED; urgency=low
2
3   * Initial Release. Closes: #12345
4   * This is my first Debian package.
5   * Adjusted the Makefile to fix $(DESTDIR) problems.
6
7  -- Josip Rodin <joy-mg@debian.org>  Mon, 22 Mar 2010 00:37:31 +0100
8
```

(Tôi đã thêm các số dòng.)

Khi bạn đã hài lòng với tất cả các thay đổi và ghi lại chúng trong `changelog`, bạn nên thay đổi giá trị phân phối từ `UNRELEASED` sang `unstable` (hoặc ngay cả `experimental`). [10]

Bạn có thể đọc về cập nhật tệp `changelog` tại Chương 8.

4.4 `rules`

Bây giờ chúng ta cần phải xem xét các quy tắc chính xác mà dpkg-buildpackage(1) sẽ thực sự sử dụng để tạo gói. Tệp này thật sự là tệp `Makefile`, nhưng nó khác với (các) tệp Makefile trong nguồn của thượng nguồn. Và không giống các tệp khác trong thư mục `debian`, đây là tệp thực thi

4.4.1 Các target của tệp `rules`

Mọi tệp `rules`, giống bất kỳ `Makefile` khác, bao gồm một vài quy tắc, định nghĩa nên target và cách nó thực hiện như thế nào. [11] Một quy tắc mới bắt đầu với tuyên bố tên target của nó trong cột đầu tiên. Các dòng theo sau bắt đầu với phím TAB (ASCII 9) để chỉ định công thức thực hiện target đó. Các dòng trống và các dòng bắt đầu với # (thăng) được xem là nhận xét và bị bỏ qua. [12]

Một quy tắc mà bạn muốn thực hiện được gọi bởi tên target của nó như là một đối số dòng lệnh. Ví dụ, `debian/rules` *`build`* và `fakeroot make -f debian/rules` *`binary`* thực hiện các quy tắc cho các target là *`build`* và *`binary`*.

Đây là một giải thích đơn giản của các target

[10] Nếu bạn sử dụng lệnh `dch -r` để tạo thay đổi cuối này, vui lòng đảm bảo bạn đã lưu tệp `changelog` một cách rõ ràng bởi trình soạn thảo

[11] Bạn có thể bắt đầu học cách viết `Makefile` từ Debian Reference, 12.2. "Make" (http://www.debian.org/doc/manuals/debian-reference/ch12#_make) . Tài liệu khả dụng như http://www.gnu.org/software/make/manual/html_node/index.html hoặc như gói `make-doc` trong `non-free`.

[12] Debian Policy Manual, 4.9 "Main building script: debian/rules" (http://www.debian.org/doc/debian-policy/ch-source.html#s-debianrules) giải thích chi tiết.

- `clean`: làm sạch tất cả các tệp đã biên dịch, sinh ra và vô ích trong build-tree(Yêu cầu)
- `build`: xây dựng từ mã nguồn thành các chương trình và sinh các tài liệu trong build-tree. (Bắt buộc)
- `build-arch`: xây dựng mã nguồn thành các chương trình phụ thuộc vào kiến trúc trong build-tree. (Bắt buộc)
- `build-indep`: xây dựng mã nguồn thành các tài liệu không phụ thuộc và kiến trúc trong build-tree. (Bắt buộc)
- `install`: cài đặt các tệp tin vào hệ thống giả cho mỗi gói nhị phân dưới thư mục `debian`. Nếu định nghĩa target `binary*` sẽ có hiệu quả phụ thuộc vào target này. (Tuỳ chọn)
- `binary`: tạo tất cả gói nhị phân (sẽ hiểu quả khi là sự kết hợp của `binary-arch` và `binary-indep` . (Bắt buộc)[13]
- `binary-arch`: tạo các gói nhị phân phụ thuộc (`Architecture: any`) trong thụ mục cha. (Bắt buộc)[14]
- `binary-indep`: để tạo ra các gói nhị phân không phụ thuộc (arch-independent)(`Architecture: all`) trong thu mục cha. (Bắt buộc)[15]
- `get-orig-source`: để lấy phiên bản gần nhất của gói nguồn gốc từ một bản lưu trữ thượng nguồn. (Tuỳ chọn)

Bạn bây giờ có thể bị choáng ngợp, nhưng mọi thứ đơn giản hơn nhiều khi kiểm tra tệp `rules` mà **dh_make** sử dụng làm mặc định.

4.4.2 Tệp `rules` mặc định

Phiên bản **dh_make** mới nhất tạo một tệp `rules` đơn giản nhưng mạnh mẽ bằng sử dụng lệnh **dh**:

```
 1 #!/usr/bin/make -f
 2 # See debhelper(7) (uncomment to enable)
 3 # output every command that modifies files on the build system.
 4 #DH_VERBOSE = 1
 5
 6 # see FEATURE AREAS in dpkg-buildflags(1)
 7 #export DEB_BUILD_MAINT_OPTIONS = hardening=+all
 8
 9 # see ENVIRONMENT in dpkg-buildflags(1)
10 # package maintainers to append CFLAGS
11 #export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
12 # package maintainers to append LDFLAGS
13 #export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed
14
15
16 %:
17         dh $@
```

(Tôi đã thêm các số dòng và thêm một số chú thích. Trong tệp tin `rules` thực tế, các ký tự khoảng cách là một mã TAB.)

Có thể bạn đã quen thuộc với các dòng như dòng 1 từ kịch bản shell hay Perl. Nó nói với hệ điều hành rằng tập tin này sẽ được xử lý với `/usr/bin/make`.

Dòng 4 có thể bỏ ghi chú để đặt biến `DH_VERBOSE` thành 1, sao cho các lệnh **dh** hiển thị các lệnh **dh_*** được thực hiện. Bạn cũng có thể thêm một dòng `export DH_OPTIONS=-v` ở đây, để mỗi **dh_*** hiển thị thêm nhiều kết quả đầu ra mà lệnh được thực hiện như thế nào. Điều này giúp bạn hiểu chính xác điều gì đang xảy ra đằng sau tệp tin đơn giản `rules` này và để gỡ lỗi các vấn đề gặp phải. **dh** là công cụ mới được thiết kế để trở thành một phần cốt lõi của công cụ `debhelper`, và không che giấu bất cứ điều gì từ bạn.

[13] Target này được sử dụng bởi `dpkg-buildpackage` như trong Phần 6.1.
[14] Target này được sử dụng bởi `dpkg-buildpackage -B` như trọng Phần 6.2.
[15] Target này được sử dụng bởi `dpkg-buildpackage -A`.

Các dòng 16 và 17 là nơi mà tất cả công việc được thực hiện với một quy tắc ngầm định sử dụng quy tắc mẫu. Dấu hiệu phần trăm có nghĩa là"bất kỳ mục tiêu" nào, sau đó gọi một chương trình duy nhất, **dh**, với tên mục tiêu. [16] Lệnh **dh** là một gói tập lệnh bao gồm các trình tự thích hợp của các chương trình **dh_*** tùy thuộc vào đối số của nó. [17]

- `debian/rules clean` chạy `dh clean`, chúng bao gồm chạy các lệnh sau:

```
dh_testdir
dh_auto_clean
dh_clean
```

- `debian/rules build` chạy `dh build`; chúng bao gồm chạy các lệnh sau:

```
dh_testdir
dh_auto_configure
dh_auto_build
dh_auto_test
```

- `fakeroot debian/rules binary` chạy `fakeroot dh binary`; chúng bao gồm chạy các lệnh sau[18]:

```
dh_testroot
dh_prep
dh_installdirs
dh_auto_install
dh_install
dh_installdocs
dh_installchangelogs
dh_installexamples
dh_installman
dh_installcatalogs
dh_installcron
dh_installdebconf
dh_installemacsen
dh_installifupdown
dh_installinfo
dh_installinit
dh_installmenu
dh_installmime
dh_installmodules
dh_installlogcheck
dh_installlogrotate
dh_installpam
dh_installppp
dh_installudev
dh_installwm
dh_installxfonts
dh_bugfiles
dh_lintian
dh_gconf
```

[16] Thao tác này sử dụng tính năng mới của `debhelper` v7+. Các khái niệm thiết kế của nó được giải thích tại Not Your Grandpa's Debhelper (http://joey.kitenet.net/talks/debhelper/debhelper-slides.pdf) được trình bày ở DebConf9 bởi tác giả `debhelper`. Từ phiên bản `lenny`, **dh_make** đã tạo ra một tệp `rules` phức tạp hơn nhiều với các quy tắc và nhiều kịch bản **dh_*** cho mỗi quy tắc, hầu hết trong số chúng hiện không cần thiết (và hiểu thị tuổi của gói). Lệnh **dh** mới đơn giản và giải phóng chúng ta khỏi các công việc thường xuyên làm "thủ công". Bạn vẫn có đủ sức mạnh để tùy biến quá trình với các target là `override_dh_*`. Xem Phần 4.4.3. Nó chỉ dựa trên gói `debhelper` và không làm xáo trộn tiến trình xây dựng gói như cách mà gói `cdbs` có xu hướng.

[17] Bạn có thể xác minh trình tự thực tế của các chương trình **dh_*** bằng gọi một *`target`* mà không thật sự chạy chúng bởi gọi `dh --no-act` *`target`* hoặc `debian/rules -- '--no-act` *`target`*`'`.

[18] Ví dụ sau đây giả định tệp `debian/compat` của bạn có một giá trị bằng hoặc lớn hơn 9 để tránh gọi bất kỳ câu lệnh hỗ trợ từ python tự động.

```
dh_icons
dh_perl
dh_usrlocal
dh_link
dh_compress
dh_fixperms
dh_strip
dh_makeshlibs
dh_shlibdeps
dh_installdeb
dh_gencontrol
dh_md5sums
dh_builddeb
```

- `fakeroot debian/rules binary-arch` chạy `fakeroot dh binary-arch`; chúng bao gồm chạy các lệnh tương tự như `fakeroot dh binary` nhưng với tuỳ chọn `-a` cho mỗi câu lệnh.
- `fakeroot debian/rules binary-indep` chạy `fakeroot dh binary-indep`; chúng bao gồm hầu hết các lệnh tương tự như `fakeroot dh binary` nhưng trừ các lệnh **dh_strip**, **dh_makeshlibs**, và **dh_shlibdeps** và với tuỳ chọn -i cho mỗi lệnh còn lại.

Các chức năng của lệnh **dh_*** phần lớn chính là tên của chúng [19] Có một vài điểm nổi bật có thể giải thích một cách đơn giản ở đây là giả sử một môi trường xây dựng điển hình dựa trên một `Makefile`: [20]

- **dh_auto_clean** thường thực hiện như sau nếu một tệp `Makefile` tồn tại với target là `distclean`. [21]

```
make distclean
```

- **dh_auto_configure** thường thực hiện như sau nếu `./configure` tồn tại (các đối số được viết tắt cho dễ đọc).

```
/configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var ...
```

- **dh_auto_build** thường thực hiện target đầu tiền của tệp `Makefile` nếu nó tồn tại.

```
make
```

- **dh_auto_test** thường thực hiện như sau nếu `Makefile` tồn tại target `test`. [22]

```
make test
```

- **dh_auto_install**thường thực hiện như sau nếu `Makefile` tồn tại với target `install` (dòng căn chỉnh cho dễ đọc)

```
make install \
  DESTDIR=/path/to/package_version-revision/debian/package
```

[19] Để có thông tin đầy đủ về những gì các tập lệnh **dh_*** thực hiện chính xác, và các tùy chọn khác của chúng là gì, vui lòng đọc các trang hướng dẫn tương ứng và tài liệu `debhelper`.

[20] Các lệnh hỗ trợ các môi trường xây dựng khác như `setup.py` có thể được liệt kê bằng cách thực thi `dh_auto_build --list` trong thư mục mã nguồn gói.

[21] Nó thực sự tìm kiếm target có sẵn đầu tiên trong `Makefile` trong số `distclean`, `realclean`, hoặc `clean`, và thực hiện điều đó.

[22] Nó thực sự tìm kiếm target đầu tiên có sẵn trong tệp `Makefile` trong số `test` hoặc `check`, và thực hiện nó.

Tất cả các target yêu cầu lệnh **fakeroot** sẽ được chứa **dh_testroot**, nó sẽ xuất hiện lỗi nếu bạn không sử dụng lệnh này để giả vờ là root.

Phần quan trọng cần biết là tệp `rules` được tạo bởi **dh_make** và nó chỉ là một đề xuất. Nó sẽ làm việc cho hầu hết các gói nhưng đối với những thứ phức tạp hơn, đừng ngại tùy chỉnh nó để phù hợp với nhu cầu của bạn.

Mặc dù `install` không phải là target bắt buộc, nó hỗ trợ. `fakeroot dh install` hành động như `fakeroot dh binary` nhưng dừng lại sau **dh_fixperms**.

4.4.3 Tuỳ chỉnh tệp `rules`

Có nhiều cách để tuỳ chỉnh tệp `rules` đã được tạo ra bởi lệnh **dh**

Lệnh `dh $@` có thể được tuỳ chỉnh như sau: [23]

- Thêm lệnh hỗ trợ **dh_python2**. (Tuỳ chọn tốt nhất cho Python.) [24]
  - Bao gồm gói `python` trong trường `Build-Depends`.
  - Sử dụng `dh $@ --with python2`.
  - Việc này xử lý các mô-đun Python sử dụng framework `python`
- Thêm lệnh hỗ trợ **dh_pysupport**.
  - Bao gồm gói `python-support` trong trường `Build-Depends`.
  - Sử dụng `dh $@ --with pysupport`
  - Việc này xử lý các mô-đun Python sử dụng framework `python-support`
- Thêm lệnh hỗ trợ **dh_pycentral**. (không dùng nữa)
  - Bao gồm gói `python-central` trong trường `Build-Depends`.
  - Sử dụng `dh $@ --with python-central`
  - Nó cũng bỏ kích hoạt lệnh **dh_pysupport**
  - Việc này xử lý các mô-đun Python sử dụng framework `python-central`
- Thêm lệnh hỗ trợ **dh_installtex**
  - Bao gồm gói `tex-common` trong trường `Build-Depends`.
  - Sử dụng `dh $@ --with tex`
  - Thao tác này sẽ đăng ký các phông chữ Loại 1, các mẫu phân đoạn và các định dạng với TeX.
- Thêm 2 lệnh hỗ trợ **dh_quilt_patch** và **dh_quilt_unpatch**
  - Bao gồm gói `quilt` trong trường `Build-Depends`.
  - Sử dụng `dh $@ --with quilt`
  - Nó áp dụng và bỏ áp dụng các bản vá cho mã nguồn ở thượng nguồn từ các tệp tin trong thư mục `debian/patches` cho gói nguồn với định dạng `1.0`.
  - Điều này là không cần thiết nếu bạn sử dụng định dạng gói nguồn mới `3.0 (quilt)`.
- Thêm lệnh hỗ trợ **dh_dkms**
  - Bao gồm gói `dkms` trong trường `Build-Depends`.
  - Sử dụng `dh $@ --with dkms`
  - Điều này xử lý chính xác việc sử dụng DKMS của gói mô-đun hạt nhân.

[23] Nếu một gói cài đặt tệp với đường dẫn `/usr/share/perl5/Debian/Debhelper/Sequence/`*`custom_name`*`.pm`, bạn nên kích hoạt chức năng tuỳ chỉnh của nó bằng `dh $@ --with `*`custom-name`*.

[24] Việc sử dụng **dh_python2** được yêu thích hơn là sử dụng **dh_pysupport** và **dh_pycentral**. Đừng sử dụng **dh_python**

- Thêm lệnh hỗ trợ **dh_autotools-dev_updateconfig** và **dh_autotools-dev_restoreconfig**.
  - Bao gồm gói `autotools-dev` trong trường `Build-Depends`.
  - Sử dụng `dh $@ --with autotools-dev`
  - Việc này cập nhật và khôi phục tệp `config.sub` và `config.guess`.
- Thêm lệnh hỗ trợ **dh_autoreconf** và **dh_autoreconf_clean**.
  - Bao gồm gói `dh-autoreconf` package in `Build-Depends`.
  - Sử dụng `dh $@ --with autoreconf`.
  - Nó cập nhật các tệp GNU Build System và khôi phục chúng sau khi xây dựng.
- Thêm lệnh hỗ trợ **dh_girepository**.
  - Bao gồm gói `gobject-introspection` trong trường `Build-Depends`.
  - Use `dh $@ --with gir`.
  - Điều này tính toán các gói phụ thuộc cho các gói truyền dữ liệu đối tượng GObject và tạo ra tên các gói thay thế cho `${gir:Depends}`
- Thêm hỗ trợ cho tính năng gõ nhanh trong **bash**
  - Bao gồm gói `bash-completion` trong trường `Build-Depends`.
  - Sử dụng `dh $@ --with bash-completion`
  - Cài đặt **bash** completions sử dụng một tệp cấu hình tại `debian/`*`package`*`.bash-completion`.

Nhiều lệnh **dh_*** được gọi từ **dh** mới có thể tuỳ chỉnh các tệp cấu hình tương ứng trong thư mục `debian`. Xem Chương 5 và sách hướng dẫn của mỗi lệnh cho việc tùy chỉnh tính năng như vậy.

Bạn có thể cần phải chạy lệnh **dh_*** được gọi qua **dh** mới với các đối số được thêm vào, hoặc để chạy các lệnh bổ sung với chúng, hoặc bỏ qua chúng. Đối với các trường hợp như vậy, bạn tạo một target mới là `override_dh_`*`foo`* bằng quy tắc của nó trong tệp `rules` định nghĩa một target `override_dh_`*`foo`* cho lệnh **dh_*foo*** bạn muốn thay đổi. Về cơ bản có nghĩa là *chạy tôi thay vì mặc định*. [25]

Xin lưu ý rằng các lệnh **dh_auto_*** có xu hướng làm nhiều hơn những gì đã giải thích một cách đơn giản ở đây nhằm có thể quản lý toàn bộ trường hợp có thể xảy ra. Nó sẽ là một ý tưởng tồi để sử dụng các target `override_dh_*` để thay thế các lệnh tương đương đã đơn giản hoá (ngoại trừ target `override_dh_auto_clean` target) vì nó có thể bỏ qua những tính năng thông minh từ gói `debhelper` features.

Ví dụ: Nếu bạn muốn lưu dữ liệu cấu hình hệ thống vào thư mục `/etc/gentoo` thay vì thư mục cha `/etc` cho gói `gentoo` hãy sử dụng Autotools, bạn có thể viết đè đối số mặc định `--sysconfig=/etc` được sử dụng bởi lệnh **dh_auto_configure**để chay lệnh **./configure** như sau:

```
override_dh_auto_configure:
        dh_auto_configure -- --sysconfig=/etc/gentoo
```

Các đối số được truyền vào `--` sẽ được nối vào đối số mặc định của chương trình "tự động-thực hiện" để ghi đè chúng. Sử dụng lệnh **dh_auto_configure** sẽ tốt hơn so với việc trực tiếp gọi lệnh **./configure** ở đây vì nó sẽ chỉ ghi đè đối số `--sysconfig` mà giữ lại bất kỳ đối số hữu ích nào khác cho lệnh **./configure** .

Nếu tệp `Makefile` trong mã nguồn của gói `gentoo` yêu cầu bạn chỉ định target `build` là target để xây dựng chương trình đó [26], bạn có thể tạo một target `override_dh_auto_build` để kích hoạt điều này.

```
override_dh_auto_build:
        dh_auto_build -- build
```

[25] Trong phiên bản `lenny`, nếu bạn muốn thay đổi hành vi của một tập lệnh **dh_***, bạn sẽ phải tìm thấy dòng liên quan trong `rules` và tự điều chỉnh nó.
[26] **dh_auto_build** với không có đối số sẽ thực hiện target đầu tiên trong tệp `Makefile`.

Điều này đảm bảo `$(MAKE)` được chạy với tất cả các đối số mặc định được cung cấp bởi lệnh **dh_auto_build** cộng với đối số `build`.

Nếu tệp `Makefile` trong mã nguồn cho `gentoo` yêu cầu bạn chỉ định target `packageclean` để làm sạch môi trường xây dựng cho gói Debian package thay vì sử dụng `distclean` hoặc `clean`, bạn có thể tạo một target `override_dh_auto_clean` để kích hoạt chúng.

```
override_dh_auto_clean:
        $(MAKE) packageclean
```

Nếu tệp `Makefile` trong mã nguồn cho `gentoo` chứa một target `test` mà bạn không muốn chạy nó trong quá trình xây dựng gói Debian, bạn có thể sử dụng một target `override_dh_auto_test` trống để bỏ qua nó.

```
override_dh_auto_test:
```

Nếu `gentoo` có chứa changelog từ thượng nguồn khác thường như tệp `FIXES`, **dh_installchangelogs** mặc định sẽ không cài đặt tệp này. Lệnh **dh_installchangelogs** yêu cầu `FIXES` như là đối số của nó để cài đặt tệp này. [27]

```
override_dh_installchangelogs:
        dh_installchangelogs FIXES
```

Khi bạn sử dụng lệnh **dh** mới, hãy sử dụng các target rõ ràng được liệt kê trong Phần 4.4.1, khác với target `get-orig-source`, có thể khó khăn để hiểu được chính xác hiệu quả. Vui lòng hạn chế target rõ ràng đối với các target `override_dh_*` và các target độc lập, nếu có thể.

[27] Các tệp tin `debian/changelog` và `debian/NEWS` luôn luôn được cài đặt tự động. Lịch sử sửa đổi của thượng nguồn được tìm thấy được chuyển thành tệp chữ thường `changelog`, `changes`, `changelog.txt`, và `changes.txt`.

Chapter 5

Các tệp khác trong thư mục `debian`

Để kiểm soát hầu hết những gì `debhelper` thực hiện khi xây dựng gói, bạn thêm các tệp cấu hình tuỳ chọn trong thư mục `debian`. Chương này sẽ cung cấp một cái nhìn tổng quan về những tệp tin này và định dạng của nó. Vui lòng tham khảo thêm Debian Policy Manual (http://www.debian.org/doc/devel-manuals#policy) và Debian Developer's Reference (http://www.debian.org/doc/devel-manuals#devref) .

Lệnh **dh_make** sẽ tạo một số tệp cấu hình mẫu dưới thư mục `debian`. Hầu hết tên các tập tin có hậu tố là `.ex`. Một số chúng có các tên tệp bắt đầu bằng tên gói nhị phân như *`package`*. Hãy xem tất cả tại. [1]

Một số tệp cấu hình mẫu cho `debhelper` có thể không được tạo bởi lệnh **dh_make**. Trong các trường hợp này, bạn cần phải tự tạo chúng bằng trình soạn thảo.

Nếu bạn muốn hoặc cần kích hoạt bất kỳ tệp nào, hãy làm như sau:

- Đổi tên các tệp mẫu bằng cách xoá bỏ hậu tố `.ex` hoặc `.EX` nếu có;
- Đổi tên các tệp cấu hình để sử dụng tên gói thực sự của thay cho *`package`*;
- sửa đổi nội dung tệp mẫu theo ý bạn muốn;
- loại bỏ các tập tin mẫu mà bạn không cần;
- sửa đổi tệp `control` (xem Phần 4.1), nếu bạn cần;
- chỉnh sửa tệp `rules` (xem Phần 4.4), nếu bạn cần.

Bất kỳ tệp tin cấu hình `debhelper` nào mà không có tiền tố *`package`*, chẳng hạn như `install` , sẽ áp dụng cho gói nhị phân đầu tiên. Khi có nhiều gói nhị phân, các cấu hình của chúng được xác định bằng các tệp chứa tiền tố là tên gói nhị phân đấy như *`package-1`*`.install`, *`package-2`*`.install`, vân vân.

5.1 `README.Debian`

Bất kỳ chi tiết bổ sung hoặc sự khác biệt giữa gói gốc ban đầu và phiên bản Debian của bạn nên được ghi lại ở đây

dh_make đã tạo ra một tệp mặc định, giống như thế này:

```
gentoo for Debian
-----------------
<possible notes regarding this package - if none, delete this file>
 -- Josip Rodin <joy-mg@debian.org>, Wed, 11 Nov 1998 21:02:14 +0100
```

Nếu bạn không có gì để ghi chú, hãy xoá tệp này. Xem dh_installdocs(1).

[1] Trong chương này, các tệp tin trong thư mục `debian` được đề cập đến mà không có `debian/` để đơn giản hóa bất cứ khi nhắc tới.

5.2 compat

Tệp `compat` định nghĩa mức độ tương thích với `debhelper`. Hiện tại, bạn nên để nó là `debhelper` v10 như sau:

```
$ echo 10 > debian/compat
```

Bạn có thể sử dụng mức v9 trong trường hợp nhất định sự tương thích với các hệ thống cũ. Tuy nhiên, sử dụng bất kỳ mức nào dưới 9 là không khuyến kích và nên là bị chặn cho các gói mới.

5.3 conffiles

Một trong những điều gây phiền phức nhất về phần mềm là khi bạn đã dành rất nhiều thời gian và nỗ lực để cấu hình một chương trình, và khi nâng cấp phần mềm thì mọi thay đổi của bạn bị đè. Debian giải quyết vấn đề này bằng cách gọi các tập tin cấu hình trong gói là conffiles. [2] Khi bạn nâng cấp một gói, bạn sẽ được hỏi rằng bạn có muốn giữ các tệp cấu hình cũ (hiện tại) hay không.

dh_installdeb(1) *tự động* đánh dâu bất kỳ tệp nào trong thư mục `/etc` là conffiles, vậy nếu chương trình của bạn có conffiles ở đây, bạn không cần thiết phải thêm chúng vào trong tệp này. Hầu hết các loại gói, nơi lưu trữ conffiles duy nhất nên là `/etc`, và vì thế tệp này không cần thiết phải tồn tại

Nếu chương trình của bạn sử dụng các tệp cấu hình nhưng cũng viết lại chúng như là cơ sở dư liệu, tốt nhất bạn không nên định nghĩa chúng là conffiles vì **dpkg** sẽ nhắc nhở người dùng xác minh các thay đổi mọi lúc.

Nếu chương trình bạn đang đóng gói đòi hỏi mỗi người dùng chỉnh sửa các tập tin cấu hình trong thư mục `/etc`, có hai cách phổ biến để định nghĩ chúng không phải là conffiles, giữ **dpkg** yên tĩnh:

- Tạo một liêt kết tượng trưng trong thư mục `/etc` trỏ tới tệp ở thư mục `/var` được sinh ra bởi maintainer scripts.
- Tạo một tệp bở maintainer scripts dưới thư mục `/etc`.

Để biêt thêm thông tin về maintainer scripts, xem Phần 5.18.

5.4 *package*.cron.*

Nếu gói của bạn yêu cầu các tác vụ được lên lịch thường xuyên để hoạt động đúng, bạn có thể sử dụng các tệp này để làm điều đó. Bạn có thể thiết lập các tác vụ thường xuyên hoặc xảy ra hàng giờ, hàng ngày, hàng tuần, hàng tháng, hoặc bất cứ lúc nào bạn muốn. Các tên tập tin là:

- *package*`.cron.hourly` - Được cài đặt tại `/etc/cron.hourly/`*package*; chạy mỗi giờ một lần.
- *package*`.cron.daily` - Được cài đặt tại `/etc/cron.daily/`*package*; chạy mỗi ngày một lần.
- *package*`.cron.weekly` - Được cài đặt tại `/etc/cron.weekly/`*package*; chạy mỗi tuần một lần.
- *package*`.cron.monthly` - Được cài đặt tại `/etc/cron.monthly/`*package*: chạy mỗi tháng một lần.
- *package*`.cron.d` - Được cài đặt tại `/etc/cron.d/`*package*: cho bất kỳ thời gian khác.

Hầu hết các tệp này là kịch bản shell, ngoại trừ tệp trong *package*`.cron.d` tuân thủ theo định dạng của crontab(5).

Không cần chỉ định tệp `cron.*` cho cài đặt ghi chép nhật ký (log rotation); để làm điều này, xem dh_installlogrotate(1) and logrotate(8).

[2] Xem dpkg(1) và Debian Policy Manual, "D.2.5 Conffiles" (http://www.debian.org/doc/debian-policy/ap-pkg-controlfields.html#s-pkg-f-Conffiles) .

5.5 dirs

Tệp này chỉ định bất kỳ thư mục nào chúng ta cần nhưng không tạo theo quy trình cài đặt thông thường (`make install DESTDIR=...` được gọi bởi `dh_auto_install`). Điều này có nghĩa sẽ có vấn đề với `Makefile`.

Các tệp này được liệt kê trong tệp `install` không cần thư mục của chúng được tạo trước. Xem Phần 5.11.

Tốt nhất là thử cài đặt trước và chỉ sửa dụng điều này nếu bạn gặp rắc rối. Không có dấu gạch chéo trước tên thư mục được liệt kê trong `dirs`

5.6 *package*.doc-base

Nếu gói của bạn có tài liệu khác với trang hướng dẫn sử dụng và thông tin (man, info), bạn nên sử dụng tệp `doc-base` để đăng ký, để người dùng có thể tìm thấy nó bằng như: dhelp(1), dwww(1), or doccentral(1).

Thường bao gồm các tệp HTML, PS và PDF, được đặt tại `/usr/share/doc/`*`packagename`*`/`.

Đây là tệp doc-base của gói `gentoo` `gentoo.doc-base` trong giống như sau:

```
Document: gentoo
Title: Gentoo Manual
Author: Emil Brink
Abstract: This manual describes what Gentoo is, and how it can be used.
Section: File Management
Format: HTML
Index: /usr/share/doc/gentoo/html/index.html
Files: /usr/share/doc/gentoo/html/*.html
```

Thông tin về định dạng tệp tin này, xem install-docs(8) và the Debian `doc-base` manual ở bản copy `/usr/share/doc/doc-base/doc-base.html/index.html` trên máy của bạn được cung cấp bởi gói `doc-base` .

Để biết thêm chi tiết về cài đặt tài liệu bổ sung, vui lòng xem Phần 3.3

5.7 docs

Tệp này chỉ định tên tệp của các tệp tài liệu mà chúng ta có thể dh_installdocs(1) cài đặt vào thư mục tạm thời cho chúng.

Mặc định, nó sẽ bao gồm tất cả các tệp tồn tại ở trong thư mục top-level (không tính thư mục con) của mã nguồn bao gồm `BUGS`, `README*`, `TODO` vân vân.

Cho `gentoo`, vài tệp tin cũng được bao gồm:

```
BUGS
CONFIG-CHANGES
CREDITS
NEWS
README
README.gtkrc
TODO
```

5.8 emacsen-*

Nếu gói của bạn cung cấp các tệp Emacs có thể được cài tại thời điểm cài đặt gói, bạn có thể sử dụng những tệp này để thiết lập nó.

Chúng được cài đặt vào thư mục tạm thời bởi dh_installemacsen(1).

Nếu bạn không cần, hãy xoá chúng đi

5.9 *package*.examples

Lệnh dh_installexamples(1) cài các tệp và thư mục được liệt kê trong tệp này như là các tệp mẫu.

5.10 *package*.init và *package*.default

Nếu gói của bạn là một daemon cần được chạy tại thời điểm start-up của hệ thống, bạn bỏ qua đề xuất ban đầu của mình, phải không ?

Tệp *package*.init được cài đặt vào kịch bản /etc/init.d/*package* mà có thể starts và stops tiếng trình daemon. Một bộ khung mẫu được cung cấp bởi lệnh **dh_make** là init.d.ex. Bạn sẽ có thể phải đổi tên và chỉnh sửa nó, khá nhiều, trong khi đảm bảo cung cấp header phù hợp với Linux Standard Base (http://www.linuxfoundation.org/collaborate/workgroups/lsb) (LSB). Nó được cài đặt vào thư mục tạm thời bởi dh_installinit(1).

Tệp *package*.default sẽ được cài đặt vào /etc/default/*package*. Tệp này cài đặt mặc định được sử dụng bởi kịch bản init. Tệp *package*.default thường được sử dụng để vô hiệu hoá một daemon đang chạy, hoặc cài đặt vài tuỳ chọn mặc định hoặc độ trê. Nếu kịch bản init của bạn có các tính năng cấu hình nhất định, bạn có thể cài chúng vào tệp*package*.default, thay vì viết trực tiếp vào chính kịch bản init.

Nếu chương trình của bạn cung cấp một tệp cho kịch bản init, bạn cũng có thể sử dụng nó hoặc không. Nếu bạn không sử dụng kịch bản init của họ thì hãy tạo một tệp là *package*.init. Tuy nhiên nếu kịch bản init của thượng nguồn trông ổn và để đúng vị trí, bạn vẫn cần cài đặt liên kết tượng trưng rc*. Để làm điều này bạn cần phải ghi đè **dh_installinit** trong tệp rules với các dòng sau:

```
override_dh_installinit:
        dh_installinit --onlyscripts
```

Nếu bạn không cần nó, hãy xoá những tệp này.

5.11 install

Nếu những tệp tin này cần được cài đặt vào trong gói của bạn nhưng make install của bạn sẽ không làm điều đó. Vậy hãy đặt các tên tập tin và đích của nó vào tệp install này, chúng sẽ được cài đặt bởi dh_install(1).[3] Bạn nên kiểm tra có hay không cách khác cụ thể hơn để sử dụng trước. Ví dụ, các tài liệu nên đặt ở docs chứ không phải trong tệp này.

Mỗi dòng trong tệp install thể hiện cho mỗi tệp được cài đặt, mỗi dòng bắt đầu với tên của tệp nguồn (tương đối với thư mục build) sao đó một dấu cách (phím space) và tiếp theo là vị trí thư mục đích sẽ được cài đặt (tương đối với thư mục cài đặt). Một ví dụ như tệp nhị phân src/*bar* muốn cài đặt; tệp install trông giống như sau:

```
src/bar usr/bin
```

[3] Tệp này sẽ thay thế lệnh dh_movefiles(1) không được dùng nữa, được cấu hình bởi tệp files .

Nghĩa là khi gói này được cài đặt, sẽ có một lệnh thực thi ở `/usr/bin/`*`bar`*.

Ngoài ra, tệp `install` có thể chứa tên của tệp tin mà không có thư mục đích để cài đặt khi thư mục tương đối không thay đổi. Định dạng này thường được sử dụng cho một gói lớn phân chia đầu ra của gói của nó thành nhiều gói nhị phân bằng cách sử dụng *`package-1`*`.install`, *`package-2`*`.install`, vân vân.

Lệnh **dh_install** sẽ tìm lại trong thư mục `debian/tmp`, nếu nó không tìm thấy chúng trong thư mục hiện tại, (hoặc bạn đã nói với nó qua tuỳ chọn `--sourcedir`).

5.12 *package*`.info`

Nếu gói của bạn có các trang thông tin, bạn nên cài chúng bởi dh_installinfo(1) bằng cách liệt kê chúng trong tệp *`package`*`.info`.

5.13 *package*`.links`

Nếu bạn cần tạo các liên kết tượng trưng trong thư mục gói như người bảo trì gói, bạn nên cài chúng bởi dh_link(1) bằng cách liệt kê đường dẫn đầy đủ của nguồn và đích trong tệp *`package`*`.links`.

5.14 {*package*`.,source/}lintian-overrides`

Nếu chương trình `lintian` báo có một chuẩn đoán về lỗi cho trường hợp khi Debian policy chấp nhận ngoại lệ cho một số quy tắc, bạn có thể sử dụng tệp *`package`*`.lintian-overrides` hoặc `source/lintian-overrides` để không còn báo lỗi từ lintian. Vui lòng đọc Lintian User's Manual (`https://lintian.debian.org/manual/index.html`) và tránh lạm dụng điều này.

`package``.lintian-overrides` cho gói nhị phân có tên là *`package`* và được cài đặt vào `usr/share/lintian/overrides/`*`pa`* bởi lệnh **dh_lintian**.

`source/lintian-overrides` là cho gói nguồn. Nó không phải cho cài đặt.

5.15 `manpage.*`

Các chương trình của bạn nên có một trang hướng dẫn (manual page). Nếu không có, bạn có thể tạo mới chúng. Lệnh **dh_make** tạo vài mẫu tệp cho các trang hướng dẫn. Chúng cần được sao chép và chỉnh sửa cho mỗi lệnh thiếu trong trang hướng dẫn. Vui lòng đảm bảo xoá các mẫu không sử dụng.

5.15.1 `manpage.1.ex`

Các trang hướng dẫn thường viết bằng nroff(1). Mẫu `manpage.1.ex` cũng được viết bằng **nroff**. Xem trang hướng dẫn man(7) để biết mô tả ngắn gọn về cách chỉnh sửa tệp đó.

Cuối cùng, tên của tệp trang hướng dẫn nên là tên của chương trình mà tệp đó mô tả, vậy chúng ta sẽ đổi tên nó từ `manpage` thành `gentoo`. Tên tệp cũng bao gồm `.1` là đuôi cho loại một, điều đó có nghĩa là một trang hướng dẫn cho lệnh của người dùng. Hãy chắc chắn xác minh rằng mục này chính xác.Đây là một danh sách ngắn của các mục:

Mục	Mô tả	Ghi chú
1	Các lệnh người dùng	Lệnh hoặc kịch bản thực thi
2	Lời gọi Hệ thống	Chức năng được cung cấp từ nhân
3	Lời gọi Thư viện	Chức năng trong thư viện hệ thống
4	Các tệp đặc biệt	Thường xuyên là tệp tại `/dev`
5	Định dạng tệp	Ví dụ định dạng `/etc/passwd`

Mục	Mô tả	Ghi chú
6	Các trò chơi	Trò chơi hoặc chương trình giải trí
7	Gói Macro	Ví dụ như macros **man**
8	Quản trị hệ thống	Các chương trình chỉ thực thi bởi root
9	Chương trình của nhân	Các lời gọi không theo chuẩn

Vì vậy trang man của `gentoo` nên được gọi là `gentoo.1`. Nếu không có tệp `gentoo.1` trong thượng nguồn, bạn nên tạo nó bằng cách đổi tên template `manpage.1.ex` thành `gentoo.1` và chỉnh sửa nó bằng cách sử dụng thông tin từ các ví dụ và tài liệu thượng nguồn.

Bạn có thể sử dụng lệnh **help2man** để tạo một manpage mới, help2man sử dụng các dữ liệu từ tuỳ chọn `--help` và `--version` của chương trình để tạo ra manpage. [4]

5.15.2 `manpage.sgml.ex`

Mặt khác, nếu bạn thích viết SGML thay vì **nroff**, bạn có thể sử dụng tệp mẫu `manpage.sgml.ex`. Nếu bạn làm điều này, bạn phải:

- đổi tên tệp tin thành `gentoo.sgml`.
- cài đặt gói `docbook-to-man`
- thêm `docbook-to-man` vào dòng `Build-Depends` của tệp `control`
- thêm mục target `override_dh_auto_build` vào tệp `rules`:

```
override_dh_auto_build:
        docbook-to-man debian/gentoo.sgml > debian/gentoo.1
        dh_auto_build
```

5.15.3 `manpage.xml.ex`

Hoặc nếu bạn yêu thích XML hơn là SGML, bạn có thể sử dụng mẫu `manpage.xml.ex` . Nếu bạn làm điều này, bạn phải:

- đổi tên thành `gentoo.1.xml`
- cài đặt gói `docbook-xsl` và một XSLT processor như `xsltproc` (khuyên dùng)
- thêm tên các gói `docbook-xsl`, `docbook-xml`, và `xsltproc` vào dòng `Build-Depends` trong tệp `control`
- thêm mục target `override_dh_auto_build` vào tệp `rules`:

```
override_dh_auto_build:
        xsltproc --nonet \
         --param make.year.ranges 1 \
         --param make.single.year.ranges 1 \
         --param man.charmap.use.subset 0 \
         -o debian/ \
http://docbook.sourceforge.net/release/xsl/current/manpages/docbook.xsl\
        debian/gentoo.1.xml
        dh_auto_build
```

[4] Lưu ý rằng lệnh **help2man**sẽ yêu cầu nhiều tài liệu chi tiết hơn có sẵn trong hệ thống info. Nếu thiếu lệnh này trong trang **info**, bạn nên tạo nó bằng cách sử dụng lệnh **help2man**.

5.16 *package*.manpages

Nếu gói của bạn có các trang hướng dẫn, bạn nên cài đặt chúng bởi dh_installman(1) bằng cách liệt kê chúng trong tệp *package*.manpage

Để cài đặt docs/gentoo.1 như một tệp manpage cho gói gentoo, hãy tạo tệp gentoo.manpages như sau:

```
docs/gentoo.1
```

5.17 NEWS

Lệnh dh_installchangelogs(1) sẽ cài đặt điều này.

5.18 {pre,post}{inst,rm}

Các tệp postinst, preinst, postrm và prerm [5] được gọi là *maintainer scripts*. Chúng là các kịch bản được đặt trong vùng điều khiển của gói và sẽ được chạy bởi **dpkg** khi gói của bạn được cài đặt, nâng cấp hoặc xóa.

Là một chuyên gia bảo trì, bạn nên tránh bất kỳ chỉnh sửa thủ công các tập lệnh bảo trì vì chúng dễ xảy ra vấn đề. Để biết thêm thông tin, hãy tham khảo Debian Policy Manual, 6 "Package maintainer scripts and installation procedure" (http://www.debian.org/doc/debian-policy/ch-maintainerscripts.html) , và xem các tệp ví dụ được cung cấp bởi **dh_make**.

Nếu bạn không lắng nghe tôi và đã tạo ra tập lệnh bảo trì tuỳ chỉnh cho một gói, bạn nên chắc chắn kiểm tra chúng không chỉ với **install** và **upgrade** mà còn **remove** và **purge**.

Nâng cấp lên phiên bản mới nên silent (lặng lẽ, ổn định) và không phá vỡ (để các người dùng không cần chú ý nâng cấp ngoại trừ việc phát hiện ra rằng lỗi cũ đã được khắc phục và có thể có các tính năng mới).

Khi sự nâng cấp cần thiết việc xâm nhập (ví dụ: các tập tin cấu hình rải rác thông qua các thư mục chính của người dùng với cấu trúc hoàn toàn khác), và bạn có thể coi đây là phương án cuối cùng để chuyển gói sang trạng thái dự phòng an toàn (ví dụ như vô hiệu hóa dịch vụ) và cung cấp tài liệu thích hợp theo yêu cầu của chính sách (README.Debian và NEWS.Debian). Đừng làm phiền người dùng với các câu hỏi **debconf** được gọi từ các tập lệnh bảo trì để nâng cấp.

Gói ucf cung cấp *conffile-like* xử lý hạ tầng để ngăn người dùng thay đổi các tệp mà không được gắn nhãn *conffiles* chẳng hạn như những tài liệu được quản lý bởi các kịch bản bảo trì. Điều này sẽ giảm thiểu các vấn đề phát sinh.

Các kịch bản bảo trì nằm trong số các cải tiến Debian nhằm giải thích **tại sao mọi người chọn Debian**. Bạn phải thật cẩn thận để không biết chúng thành nguồn gây phiền toái.

5.19 *package*.symbols

Việc đóng gói một thư viện là không dễ dàng cho người bảo trì mới làm quen và nên tránh. Còn nếu gói của bạn có các thử viện, bạn nên có tệp debian/*package*.symbols. Xem Phần A.2.

5.20 TODO

Lệnh dh_installdocs(1) cài đặt tệp này.

[5] Mặc dù việc sử dụng biểu thức viết tắt **bash** {pre,post}{inst,rm} để chỉ ra các tệp này, bạn nên sử dụng cú pháp POSIX cho các maintainer scripts tương thích với dash như trình shell của hệ thống

5.21 watch

Định dạng tệp `watch` là tài liệu uscan(1). Tệp `watch` cấu hình chương trình **uscan** (có trong gói `devscripts`) để theo dõi trang web mà bạn lấy nguồn. Nó cũng được sử dụng bởi dịch vụ Debian Package Tracker (https://tracker.debian.org/) .

Đây là nội dung của nó:

```
# watch control file for uscan
version=3
http://sf.net/gentoo/gentoo-(.+)\.tar\.gz debian uupdate
```

Thông thường với một tệp `watch`, đường dẫn URL tại `http://sf.net/gentoo` được tải về và tìm kiếm cho các liên kết theo biểu mẫu `<a href=...>`. Tên (phần phía sau của đường dẫn `/`) của mỗi liên kết URL được so sánh với mẫu biểu thức chính quy Perl (xem perlre(1))`gentoo-(.+)\.tar\.gz`. Trong số các tệp phù hợp, tệp đượctải xuống có số phiên bản lớn nhất và chương trình **uupdate**được chạy để tạo một bản nguồn đã được cập nhật.

Mặc dù điều này là đúng cho vài trang web, dịch vụ tải của SourceForge tại http://sf.net là một ngoại lệ. Khi tệp `watch` chứa một đường dẫn URL khớp với biểu thức chính quy Perl `^http://sf\.net/`, chương trình **uscan** thay thế nó với `http://qa.debian.org/watch/sf.php/` và sau đó áp dụng quy tắc này. Dịch vụ chuyển hướng URL tại http://qa.debian.org/- được thiết kế để cung cấp dịch vụ ổn định đến tệp mong muốn với bất kỳ `watch` mẫu biểu thức `http://sf.net/`*`project/tar-name`* Điều này giải quyết các vấn đề liên quan đế thay đổi định kỳ các đường dẫn SourceForge.

Nếu thượng nguồn cung cấp chữ ký mật mã của tarball, bạn nên xác minh tính xác thực của nó bằng cách sử dụng tùy chọn `pgpsigurlmangle` như mô tả trong uscan(1).

5.22 source/format

Trong tệp `debian/source/format`, nên có một dòng chỉ định định dạng mong muốn cho gói nguồn (kiểm tra dpkg-source> (1) để có danh sách đầy đủ). Sau phiên bản `squeeze`, cần chỉ ra rằng:

- `3.0 (native)` cho các gói Debian native hoặc
- `3.0 (quilt)` cho mọi thứ khác.

Định dạng `3.0 (quilt)` mới sẽ ghi lại các sửa đổi trong một loạt bản vá lỗi **quilt** trong thư mục `debian/patches` . Những thay đổi này sau đó sẽ tự động được áp dụng trong quá trình giải nén gói nguồn. [6] Các sửa đổi Debian chỉ đơn giản được lưu trữ trong kho lưu trữ `debian.tar.gz` chứa tất cả các tệp tin trong thư mục `debian`. Định dạng mới này hỗ trợ đưa các tệp tin nhị phân như các biểu tượng PNG của người bảo trì gói mà không yêu cầu thủ thuật. [7]

Khi **dpkg-source** giải nén một gói nguồn trong định dạng nguồn `3.0 (quilt)`, nó sẽ tự động áp dụng tất cả các bản vá được liệt kê trong `debian patches/series`. Bạn có thể bỏ qua việc áp dụng các bản vá ở cuối quá trình giải nén với tùy chọn `--skip-patches`.

5.23 source/local-options

Khi bạn muốn quản lý các hoạt động đóng gói Debian bằng VCS, bạn thường tạo một nhánh (ví dụ. `upstream`) để theo dõi thượng nguồn và một nhánh khác (thường là `master` đối với Git) để theo dõi gói Debian. Đối với trường hợp thứ hai, bạn thường xuyên muốn có thượng nguồn chưa được vá lỗi với các tệp của bạn `debian/*` để dễ dàng hợp nhất với mã nguồn mới của thượng nguồn.

[6] Xem DebSrc3.0 (http://wiki.debian.org/Projects/DebSrc3.0) để biết tổng quan việc chuyển đổi sang các định dạng mã nguồn mới `3.0 (quilt)` and `3.0 (native)`.

[7] Trên thực tế, định dạng mới này cũng hỗ trợ nhiều tarballs thượng nguồn và các phương pháp nén khác. Những điều này nằm ngoài phạm vi của tài liệu này.

Sau khi bạn xây dựng một gói, nguồn thường đã được vá. Bạn cần phải gỡ vá bằng tay bằng cách chạy `dquilt pop -a` trước khi đẩy hết vào nhánh `master`. Bạn có thể tự động hóa việc này bằng cách thêm tệp tin tuỳ chọn `debian/source/local-options` để chứa các `unapply-patches`. Tệp này không được bao gồm trong gói nguồn đã tạo và chỉ thay đổi hành vi xây dựng cục bộ. Tệp này có thể chứa `abort-on-upstream-changes`, (xem thêm dpkg-source (1)).

```
unapply-patches
abort-on-upstream-changes
```

5.24 source/options

Các tệp được tự động sinh trong cây nguồn có thể gây khó chịu cho đóng gói vì chúng tạo ra các tệp vá lỗi không có ý nghĩa. Có các mô đun tùy chỉnh như **dh_autoreconf** nhằm giải quyết các vấn đề này như mô tả trong Phần 4.4.3.

Bạn có thể cung cấp một biểu thức chính quy Perl là đối số của tùy chọn `--extend-diff-ignore` khi chạy dpkg-source (1) để bỏ qua các thay đổi được thực hiện đối với các tệp được tạo tự động trong khi tạo gói nguồn.

Như một giải pháp chung để giải quyết vấn đề của các tệp tự động tạo mới, bạn có thể lưu trữ một đối số tùy chọn **dpkg-source** trong tệp tin `source/options` . Ví dụ sau sẽ bỏ qua việc tạo tệp vá cho `config.sub`, `config.guess` và `Makefile`.

```
extend-diff-ignore = "(^|/)(config\.sub|config\.guess|Makefile)$"
```

5.25 patches/*

Định dạng nguồn cũ `1.0` tạo một tệp `diff.gz` duy nhất có chứa tệp bảo trì gói trong `debian` và các tệp vá cho nguồn. Một gói như vậy hơi rườm rà để kiểm tra và hiểu cho mỗi thay đổi cây nguồn sau đó. Vì thế đây không phải định dạng tốt.

Định dạng mới `3.0 (quilt)` sẽ lưu các bản vá lỗi vào các tệp `debian/patches/*` bằng cách sử dụng công cụ **quilt**. Các bản vá này và các dữ liệu gói khác nằm trong thư mục `debian` được đóng gói dưới dạng tệp `debian.tar.gz`. Do lệnh **dpkg-source** có thể xử lý dữ liệu vá lỗi định dạng của **quilt** trong định dạng `3.0 (quilt)` mà không cần tới gói `quilt`, nó không cần `quilt` ở trong mục `Build-Depends`. [8]

Lệnh **quilt** được giải thích trong quilt(1). Nó ghi chép lại các thay đổi của mã nguồn và tạo thành các bản vá `-p1` trong thư mục `debian/patches` và cây mã nguồn không bị ảnh hưởng ngoại trừ thư mục `debian`. Trình tự áp dụng bản vá được ghi lại tại tệp `debian/patches/series`. Bạn có thể áp dụng (=push), un-apply (=pop), và làm mới một các dễ dàng. [9]

Đối với Chương 3, chúng tôi đã tạo ba bản vá lỗi trong `debian/patches`.

Vì các bản vá lỗi của Debian được đặt trong `debian/patches`, hãy đảm bảo thiết lập lệnh **dquilt** đúng như mô tả trong Phần 3.1.

Khi mọi người (kể cả bạn) cung cấp một bản vá lỗi mới *`foo`*`.patch` cho mã nguồn, áp dụng cho mã nguồn định dạng gói nguồn `3.0 (quilt)` khá đơn giản:

```
$ dpkg-source -x gentoo_0.9.12.dsc
$ cd gentoo-0.9.12
$ dquilt import ../foo.patch
$ dquilt push
$ dquilt refresh
$ dquilt header -e
... describe patch
```

[8] Một số phương pháp bảo trì bộ vá đã được đề xuất và đang được sử dụng cho các gói Debian. Hệ thống **quilt** là hệ thống bảo trì được ưa thích sử dụng. Các hệ thống khác bao gồm **dpatch**, **dbs**, và **cdbs**. Nhiều trong số này giữ các bản vá như các tệp tin `debian/patches/*`.

[9] Nếu bạn đang yêu cầu một nhà tài trợ để tải lên gói của bạn, cần làm rõ ràng và viết tài liệu của những thay đổi của bạn là rất quan trọng để tiến hành xem xét gói bởi nhà tài trợ.

Các bản vá được lưu trữ trong phiên bản mới `3.0 (quilt)` phải là *fuzz* tự do. Bạn có thể đảm bảo điều này với `dquilt pop -a; while dquilt push; do dquilt refresh; done`.

Chapter 6

Biên dịch gói

Bây giờ chúng ta xem như đã sẵn sàng để biên dịch gói.

6.1 (Tái) biên dịch toàn bộ

Để có thể (tái) biên dịch toàn bộ gói một cách đúng đắn, bạn cần phải đảm bảo là bạn đã cài đặt

- gói `build-essential`,
- các gói liệt kê ở trong mục `Build-Depends` (xem Phần 4.1), và
- các gói được liệt kê trong mục `Build-Depends-indep` (see Phần 4.1).

Sau đó bạn thực thi lệnh sau ở trong thư mục nguồn:

```
$ dpkg-buildpackage -us -uc
```

Lệnh này sẽ làm hết tất cả những thứ cần thiết để tạo gói nhị phân và nguồn hoàn chỉnh cho bạn. Nó sẽ

- xóa sạch cây thư mục nguồn (`debian/rules clean`)
- biên dịch gói nguồn (`dpkg-source -b`)
- biên dịch chương trình (`debian/rules build`)
- biên dịch gói nhị phân (`fakeroot debian/rules binary`)
- tạo tập tin `.dsc`
- tạo tập tin `.changes`, dùng lệnh **dpkg-genchanges**

Nếu kết quả biên dịch đạt mức thỏa mãn yêu cầu, kí tên các tập tin `.dsc` và `.changes` dùng mã GPG riêng tư của bạn bằng lệnh **debsign**. Bạn cần phải nhập mật mã, hai lần. [1]

Đối với một gói không đến từ Debian, ví dụ như `gentoo`, bạn sẽ thấy những tập tin sau trong thư mục cấp trên (`~/gentoo`) sau khi biên dịch các gói xong:

[1] Mã GPG này phải được ký tên bởi một nhà phát triển Debian để có thể được kết nối tới mạng lưới tín nhiệm và phải được đăng ký vào tập hợp mã tin cậy của Debian (http://keyring.debian.org) . Điều này cho phép các gói bạn tải lên được chấp nhận vào kho của Debian. Xem Tạo một mã GPG mới (http://keyring.debian.org/creating-key.html) và Debian Wiki về Ký tên lên mã (http://wiki.debian.org/Keysigning) .

- `gentoo_0.9.12.orig.tar.gz`

  Đây là tập tin nén chứa mã nguồn nguyên gốc từ các nhà phát triển, chỉ được đổi tên về tên như trên để tuân thủ theo quy định của Debian. Chú ý rằng tập tin trên được tạo trước tiên bởi lệnh `dh_make -f ../gentoo-0.9.12.tar.gz`.

- `gentoo_0.9.12-1.dsc`

  Đây là tóm tắt nội dung của mã nguồn. Tập tin này được tạo ra từ tập tin `control`, và nó được sử dụng khi giải nén mã nguồn bằng lệnh dpkg-source(1).

- `gentoo_0.9.12-1.debian.tar.gz`

  Tập tin nén này chứa các nội dung trong thư mục `debian`. Mỗi điều chỉnh bạn thêm vào mã nguồn ban đầu được lưu ở dạng một bản vá **quilt** trong `debian/patches`.

  Nếu bất cứ ai muốn tự tạo gói từ đầu, họ có thể làm thế bằng việc sử dung ba tập tin phía trên. Quy trình giải nén thì khá dễ dàng: chỉ cần sao chép ba tập tin đến một nơi khác và chạy lệnh `dpkg-source -x gentoo_0.9.12-1.dsc`. [2]

- `gentoo_0.9.12-1_i386.deb`

  Đây là gói nhị phân hoàn chỉnh của bạn. Bạn có thể dùng lệnh **dpkg** để cài đặt và tháo gỡ gói nhị phân này như những gói khác.

- `gentoo_0.9.12-1_i386.changes`

  Tập tin này mô tả tất cả những thay đổi trong phiên bản hiện thời của gói; nó được dùng bởi các chương trình bảo trì kho FTP của Debian để cài đặt các gói nhị phân và gói nguồn. Nó một phần được tạo ra từ tập tin `changelog` và tập tin `.dsc`.

  Trong lúc bạn làm việc với gói, biểu hiện của gói sẽ thay đổi và những tính năng mới sẽ được thêm vào. Những người tải gói của bạn về có thể nhìn vào tập tin này và nhìn sơ nhanh được có những gì đã thay đổi. Các chương trình bảo trì kho Debian sẽ đăng các nội dung của tập tin này lên bó thư debian-devel-changes@lists.debian.org (http://lists.debian.org/debian-devel-changes/) .

Các tập tin `gentoo_0.9.12-1.dsc` và `gentoo_0.9.12-1_i386.changes` phải được ký tên bằng lệnh **debsign** với mã GPG riêng tư của bạn trong thư mục `~/.gnupg/`, trước khi tải chúng lên kho FTP của Debian. Chữ ký GPG chứng minh những tập tin này là thật sự của bạn bằng mã GPG công cộng của bạn.

Lệnh **debsign** có thể được dùng để ký tên với mã nhận dạng của khóa GPG bí mật của bạn (tốt cho việc tài trợ các gói) được chỉ ra trong dòng sau trong tập tin `~/.devscripts`:

```
DEBSIGN_KEYID=Your_GPG_keyID
```

Những chuỗi số dài trong các tập tin `.dsc` và `.changes` là mã băm SHA1/SHA256 cho những tập tin đã được đề cập. Bất kỳ ai tải xuống các tập tin của bạn có thể sử dung và kiểm tra chúng bằng lệnh sha1sum(1) hoặc sha256sum(1) và nếu các số không khớp, họ sẽ biết ngay là tập tin đã bị lỗi hoặc đã bị chỉnh sửa.

6.2 Tự động biên dịch

Debian hỗ trợ nhiều cổng (http://www.debian.org/ports/) với mạng lưới biên dịch tự động (http://www.debian.org/devel/buildd/) chạy các dịch vụ **buildd** trên các máy tính chạy nhiều nền tảng khác nhau. Mặc dù bạn không cần phải tự làm điều này, bạn cần phải biết những gì sẽ xảy ra với các gói phần mềm của bạn. Hãy nhìn vào cách chúng biên dịch các gói phần mềm của bạn cho nhiều nền tảng. [3]

Đối với các gói phần mềm cho nền tảng `Architecture: any`, hệ thống tự động biên dịch thực hiện hành động tái biên dịch. Nó đảm bảo việc cài đặt của

- gói `build-essential`, và
- các gói được liệt kê trong trường `Build-Depends` (see Phần 4.1).

[2] Bạn có thể tránh việc áp dụng các bản vá **quilt** dưới định dạng nguồn `3.0 (quilt)` khi kết thúc việc giải nén với tham số `--skip-patches`. Một lựa chọn khác là bạn có thể chạy lệnh `dquilt pop -a` sau khi làm xong các thao tác thông thường.

[3] Hệ thống tự động biên dịch thực tế cần các kế hoạch phức tạp hơn nhiều kế hoạch được giải thích ở đây. Những chi tiết như thế là quá tầm của tài liệu này.

Sau đó nó chạy lệnh sau trong thư mục nguồn:

```
$ dpkg-buildpackage -B
```

Lệnh này sẽ làm tất cả mọi thứ để tạo ra các gói phần mềm nhị phân phụ thuộc vào nền tảng trên một nền tảng khác. Nó sẽ:

- xóa sạch cây thư mục nguồn (`debian/rules clean`)
- biên dịch chương trình (`debian/rules build`)
- biên dịch các gói phần mềm nhị phân phụ thuộc vào nền tảng (`fakeroot debian/rules binary-arch`)
- ký tên vào tập tin nguồn `.dsc`, dùng lệnh **gpg**
- tạo và ký tên tập tin sẽ được tải lên `.changes`, dùng lệnh **dpkg-genchanges** và **gpg**

Đây là lý do bạn thấy gói phần mềm của bạn cho các nền tảng khác.

Mặc dù các gói phần mềm liệt kê trong trường `Build-Depends-Indep` bị buộc phải được cài đặt cho quá trình đóng gói bình thường của chúng tôi (xem Phần 6.1), chúng không bị buộc phải được cài đặt bởi hệ thống tự động biên dịch bởi vì nó biên dịch chỉ những gói phần mềm phụ thuộc vào nền tảng. [4] Điểm khác biệt giữa thủ tục đóng gói bình thường và đóng gói tự động là cái gì chỉ ra bạn nên ghi lại những gói phần mềm bắt buộc trong trường `Build-Depends` hay `Build-Depends-Indep` của tập tin `debian/control` (xem Phần 4.1).

6.3 Lệnh debuild

Bạn có thể tự động hóa hành động biên dịch xung quanh việc thực thi lệnh **dpkg-buildpackage** và lệnh **debuild**. Xem debuild(1).

Lệnh **debuild** chạy lệnh **lintian** để thực hiện bài kiểm tra tĩnh sau khi biên dịch gói phần mềm Debian. Lệnh **lintian** có thể được thay đổi với các dòng sau trong tập tin `~/.devscripts`:

```
DEBUILD_DPKG_BUILDPACKAGE_OPTS="-us -uc -I -i"
DEBUILD_LINTIAN_OPTS="-i -I --show-overrides"
```

Dọn dẹp mã nguồn và tái biên dịch gói phần mềm từ tài khoản của bạn thì đơn giản như sau:

```
$ debuild
```

Bạn có thể dọn dẹp cây mã nguồn đơn giản như sau:

```
$ debuild -- clean
```

[4] Khác với khi dưới gói phần mềm `pbuilder`, môi trường **chroot** dưới gói `sbuild` vốn được sử dụng bởi hệ thống tự động biên dịch không buộc sử dụng một hệ thống nhỏ gọn và có thể có nhiều gói phần mềm thừa đã được cài đặt.

6.4 Gói phần mềm `pbuilder`

Để có một môi trường biên dịch (**chroot**) sạch sẽ để kiểm tra các điều kiện biên dịch, gói phần mềm `pbuilder` rất hữu dụng. [5] Nó đảm bảo việc biên dịch sạch sẽ từ mã nguồn dưới hệ thống biên dịch tự động `sid` cho các nền tảng khác nhau và tránh một lỗi nghiêm trọng FTBFS (Thất Bại Biên Dịch Từ Nguồn) vốn luôn nằm trong thể loại RC (Nghiêm Trọng Cho Phát Hành). [6]

Hãy tùy chỉnh gói phần mềm `pbuilder` như sau:

- Làm cho thư mục `/var/cache/pbuilder/result` có thể ghi được bởi tài khoản của bạn.
- tạo một thư mục, ví dụ như */var/cache/pbuilder/hooks*, có thể ghi được bởi người dùng, để đặt mã kịch bản mồi vào.
- cấu hình `~/.pbuilderrc` hoặc `/etc/pbuilderrc` để bao gồm những thứ sau:

```
AUTO_DEBSIGN=${AUTO_DEBSIGN:-no}
HOOKDIR=/var/cache/pbuilder/hooks
```

Trước tiên hãy khởi động hệ thống `pbuilder` **chroot** nội bộ như sau:

```
$ sudo pbuilder create
```

Nếu bạn đang có một gói phần mềm chứa mã nguồn hoàn chỉnh, chạy các lệnh sau dưới thư mục nơi các tập tin *foo*`.orig.tar.gz`, *foo*`.debian.tar.gz`, và *foo*`.dsc` tồn tại để cập nhật hệ thống `pbuilder` **chroot** nội bộ và để biên dịch các gói phần mềm nhị phân trong nó:

```
$ sudo pbuilder --update
$ sudo pbuilder --build foo_version.dsc
```

Các gói phần mềm vừa mới được biên dịch mà không có chữ ký GPG sẽ được đặt ở `/var/cache/pbuilder/result/` với chủ sở hữu không phải là root.

Các chữ ký GPG trên các tập tin `.dsc` và `.changes` có thể được tạo ra như sau:

```
$ cd /var/cache/pbuilder/result/
$ debsign foo_version_arch.changes
```

Nếu bạn có một cây thư mục chứa mã nguồn đã được cập nhật nhưng chưa tạo ra gói phần mềm chứa mã nguồn tương ứng, thay vì thế hãy chạy các lệnh sau trong thư mục nguồn nơi thư mục `debian` tồn tại:

```
$ sudo pbuilder --update
$ pdebuild
```

Bạn có thể đăng nhập vào môi trường **chroot** với lệnh `pbuilder --login --save-after-login` và cấu hình nó theo ý bạn muốn. Môi trường này có thể được lưu bằng việc rời khỏi dòng lệnh với thao tác `^D` (Control-D).

Phiên bản mới nhất của lệnh **lintian** có thể được chạy từ môi trường `chroot` dùng kịch bản mồi */var/cache/pbuilder/hooks/*`B90lintian` cấu hình như sau: [7]

[5] Bởi vì gói phần mềm `pbuilder` vẫn đang tiếp tục tiến hóa, bạn nên kiểm tra môi trường cấu hình thực sự bằng việc đọc tài liệu hướng dẫn mới nhất.

[6] Xem http://buildd.debian.org/ để biết thêm về việc tự động biên dịch các gói phần mềm Debian.

[7] Điều này giả sử là `HOOKDIR=/var/cache/pbuilder/hooks`. Bạn có thể tìm thấy nhiều ví dụ của các kịch bản mồi trong thư mục `/usr/share/doc/pbuilder/examples`.

```
#!/bin/sh
set -e
install_packages() {
        apt-get -y --allow-downgrades install "$@"
        }
install_packages lintian
echo "+++ lintian output +++"
su -c "lintian -i -I --show-overrides /tmp/buildd/*.changes" - pbuilder
# use this version if you don't want lintian to fail the build
#su -c "lintian -i -I --show-overrides /tmp/buildd/*.changes; :" - pbuilder
echo "+++ end of lintian output +++"
```

Bạn cần phải có kết nối đến môi trường `sid` mới nhất để biên dịch gói phần đúng cách cho `sid`. Trên thực tế, `sid` có thể gặp vấn đề làm cho bạn cảm thấy khó khăn hơn khi di chuyển toàn bộ hệ thống của bạn. Gói phần mềm `pbuilder` có thể giúp bạn đối mặt với loại tình huống này.

Bạn có thể cần phải cập nhật các gói phần mềm `stable` sau khi chúng được tung ra cho `stable-proposed-updates`, `stable/updates`, v.v. [8] Cho những tình huống như thế, việc bạn có thể đang chạy một hệ thống `sid` không phải là một lý do đủ tốt cho việc thất bại trong cập nhật chúng đúng cách. Gói phần mềm `pbuilder` có thể giúp bạn truy cập các môi trường của bất cứ bản phân phối hệ điều hành nào xuất phát từ Debian trên cùng một nền tảng.

Xem http://www.netfort.gr.jp/~dancer/software/pbuilder.html, pdebuild(1), pbuilderrc(5), và pbuilder(8).

6.5 Lệnh git-buildpackage và những lệnh tương tự

Nếu thượng nguồn của bạn dùng một hệ thống quản lý mã nguồn (VCS) [9] để bảo trì mã nguồn của họ, bạn cũng nên xem xét việc sử dụng nó. Nó có thể giúp bạn gộp và chọn các bản vá từ thượng nguồn dễ hơn. Có nhiều gói phần mềm chứa mã kịch bản dùng cho việc đóng gói các gói phần mềm Debian cho từng VCS.

- `git-buildpackage`: a giải pháp để giúp việc đóng gói phần mềm Debian với các kho mã nguồn Git.
- `svn-buildpackage`: các chương trình giúp đỡ để bảo trì các gói phần mềm Debian với Subversion.
- `cvs-buildpackage`: một tập hợp các kịch bản đóng gói Debian cho các cây mã nguồn CVS.

Việc sử dụng `git-buildpackage` đang dần trở nên phổ biến đối với các nhà phát triển Debian để quản lý các gói phần mềm Debian với máy phục vụ Git trên alioth.debian.org (http://alioth.debian.org/) . [10] Gói phần mềm này cung cấp nhiều lệnh để *tự động hóa* các thao tác đóng gói:

- git-import-dsc(1): nhập một gói phần mềm Debian trước đây vào một kho Git.
- git-import-orig(1): nhập một tập tin tar mới từ thượng nguồn vào một kho Git.
- git-dch(1): tạo ra một changelog(nhật ký thay đổi) từ các thông điệp Git.
- git-buildpackage(1): biên dịch các gói Debian từ một kho Git.
- git-pbuilder(1): biên dịch các gói Debian từ một kho Git dùng lệnh **pbuilder**/**cowbuilder**.

Những lệnh sau dùng 3 nhánh để theo dõi thao tác đóng gói:

- `main` cho cây mã nguồn của gói phần mềm Debian.

[8] Có một vài giới hạn cho những cập nhật như vậy đối với gói phần mềm `stable` của bạn.

[9] Xem Các hệ thống quản lý phiên bản (http://www.debian.org/doc/manuals/debian-reference/ch10#_version_control_systems) để biết thêm.

[10] Debian wiki Alioth (http://wiki.debian.org/Alioth) hướng dẫn cách sử dụng dịch vụ alioth.debian.org (http://alioth.debian.org/) .

- `upstream` for cây mã nguồn của thượng nguồn.
- `pristine-tar` cho tập tin nén thượng nguồn tạo ra bởi tham số `--pristine-tar`.[11]

Bạn có thể cấu hình `git-buildpackage` với `~/.gbp.conf`. Xem gbp.conf(5). [12]

6.6 Tái biên dịch nhanh

Với một gói phần mềm lớn, bạn có thể không muốn biên dịch lại từ đầu mỗi lần bạn tinh chỉnh các chi tiết trong `debian/rules`. Dành cho mục đích kiểm tra, bạn có thể tạo một tập tin `.deb` mà không cần tái biên dịch các mã nguồn từ thượng nguồn như sau[13]:

```
$ fakeroot debian/rules binary
```

Hoặc đơn giản chạy lệnh sau để xem nó có biên dịch được hay không:

```
$ fakeroot debian/rules build
```

Khi bạn hoàn tất tinh chỉnh, hãy nhớ tái biên dịch với thủ tục chính quy. Bạn có khả năng không thể tải lên được đúng cách nếu bạn cố gắng tải các tập tin `.deb` được biên dịch theo cách này.

6.7 Sơ đồ lệnh

Đây là một bản tóm tắt cho thấy các lệnh biên dịch gói phần mềm phối hợp như thế nào trong sơ đồ lệnh. Có nhiều cách để làm một thứ.

- `debian/rules` = kịch bản của người bảo trì cho biên dịch gói phần mềm
- **dpkg-buildpackage** = lõi của công cụ biên dịch gói phần mềm
- **debuild** = **dpkg-buildpackage** + **lintian** (biên dịch với các biến môi trường dọn dẹp sạch sẽ)
- **pbuilder** = lõi của công cụ sắp đặt môi trường chroot Debian.
- **pdebuild** = **pbuilder** + **dpkg-buildpackage** (biên dịch trong chroot)
- **cowbuilder** = tăng tốc tốc độ chạy của **pbuilder**
- **git-pbuilder** = lệnh với ngữ pháp dễ sử dụng thay thế cho lệnh**pdebuild** (dùng bởi lệnh**gbp buildpackage**)
- **gbp** = quản lý mã nguồn của gói phần mềm dưới kho git

[11] Tham số `--pristine-tar` chạy lệnh **pristine-tar**, lệnh này tạo ra một bản sao nguyên vẹn của tập tin nén thượng nguồn chưa chỉnh sửa bằng việc sử dụng một tập tin nhị phân delta nhỏ và những nội dung của tập tin nén vốn được giữ trong nhánh `upstream` trong VCS.

[12] Đây là một vài nguồn thông tin điện tử có sẵn cho các đối tượng người đọc thông thái.

- Biên dịch các Gói Phần Mềm Debian với git-buildpackage (`/usr/share/doc/git-buildpackage/manual-html/gbp.html`)
- các gói phần mềm debian trong git (https://honk.sigxcpu.org/piki/development/debian_packages_in_git/)
- Dùng Git cho đóng gói phần mềm Debian (http://www.eyrie.org/~eagle/notes/debian/git.html)
- git-dpm: các gói phần mềm Debian trong công cụ quản lý Git (http://git-dpm.alioth.debian.org/)
- Dùng TopGit để tạo một chuỗi quilt cho đóng gói phần mềm Debian (http://git.debian.org/?p=collab-maint/topgit.git;a=blob_plain;f=debian/HOWTO-tg2quilt;hb=HEAD)

[13] Các biến môi trường vốn được cấu hình bình thường với các giá trị phù hợp không được gán giá trị với phương pháp này. Đừng bao giờ tạo các gói phần mềm thực sự để được tải lên với phương pháp **nhanh** này.

- **gbp buildpackage** = **pbuilder** + **dpkg-buildpackage** + **gbp**

Mặc dù việc sử dụng các lệnh cấp cao hơn như **gbp buildpackage** và **pbuilder** đảm bảo một môi trường biên dịch phần mềm hoàn hảo, hãy đặt sự quan trọng vào việc hiểu các lệnh cấp thấp như `debian/rules` và **dpkg-buildpackage** được thực thi như thế nào dưới các lệnh cấp cao.

Chapter 7

Kiểm tra gói để tìm lỗi

Có vài phương pháp bạn nên biết để dò tìm lỗi trong một gói phần mềm trước khi tải nó lên kho phần mềm công cộng.

Thực hiện việc kiểm tra trên một máy khác không phải của bạn cũng là một ý hay. Ban phải quan sát kỹ bất kỳ cảnh báo hoặc lỗi cho tất cả các bài kiểm tra được mô tả ở đây.

7.1 Những thay đổi đáng nghi

Nếu bạn tìm thấy một tập tin vá mới được tự động tạo ra như là `debian-changes-*` trong thư mục `debian/patches` sau khi biên dịch gói phần mềm Debian không dựa trên nền tảng máy bạn dưới định dạng `3.0 (quilt)`, rất có thể bạn đã thay đổi vài tập tin vô ý hoặc kịch bản biên dịch đã thay đổi mã nguồn của thượng nguồn. Nếu đây là lỗi của bạn, hãy sửa nó. Nếu đây là lỗi gây ra bởi kịch bản biên dịch, sửa nguyên nhân gốc với lệnh **dh-autoreconf** như trong Phần 4.4.3 hoặc đi vòng qua nó với tập tin `source/options` như trong Phần 5.24.

7.2 Kiểm tra việc cài đặt của gói phần mềm

Bạn phải kiểm tra gói phần mềm của bạn xem có lỗi trong quá trình cài đặt hay không. Lệnh debi(1) giúp bạn kiểm tra việc cài đặt của tất cả các gói phần mềm nhị phân được tạo ra.

```
$ sudo debi gentoo_0.9.12-1_i386.changes
```

Để tránh vấn đề trong quá trình cài đặt trên các hệ thống khác nhau, bạn phải chắc chắn rằng không có xung đột về tên tập tin với các gói phần mềm khác đang hiện hữu, dùng tập tin `Contents-`*`i386`* tải về từ kho Debian. Lệnh **apt-file** có thể rất tiện cho công việc này. Nếu có xung đột, xin hãy hành động để tránh vấn đề thực tế này, dù cho bằng cách thay đổi tên tập tin, bằng cách di chuyển một tập tin chung qua một gói phần mềm khác mà nhiều gói phần mềm có thể dựa trên, bằng các phương pháp thay thế (xem update-alternatives(1)) phối hợp với các nhà bảo trì của các gói phần mềm bị ảnh hưởng khác, hoặc khai báo một mối quan hệ `Conflicts` trong tập tin `debian/control`.

7.3 Kiểm tra các kịch bản bảo trì của một gói phần mềm

Tất cả các kịch bản bảo trì (đó là, các tập tin `preinst`, `prerm`, `postinst`, and `postrm`) rất khó để viết đúng cách trừ khi chúng được tạo ra tự động bởi các chương trình `debhelper`. Vì thể đừng sử dụng chúng nếu bạn là một người bảo trì mới vào nghề (xem Phần 5.18).

Nếu gói phần mềm tận dụng các kịch bản bảo trì phức tạp này, hãy chắc chắn rằng bạn kiểm tra không chỉ quá trình cài đặt mà cả công đoạn tháo gỡ phần mềm, xóa hoàn toàn phần mềm, và nâng cấp. Nhiều lỗi kịch bản bảo trì hiện ra khi các phần đã được tháo gỡ hoặc xóa bỏ. Sử dụng lệnh **dpkg** như sau để kiểm tra:

```
$ sudo dpkg -r gentoo
$ sudo dpkg -P gentoo
$ sudo dpkg -i gentoo_version-revision_i386.deb
```

Lệnh đó nên được chạy theo các chuỗi như sau:

- cài đặt phiên bản trước đó (nếu cần thiết).
- nâng cấp nó từ phiên bản trước.
- giảm cấp nó xuống phiên bản thấp hơn (không bắt buộc).
- xóa bỏ hoàn toàn nó.
- cài đặt gói phần mềm mới
- tháo gỡ nó.
- cài đặt nó lại.
- xóa bỏ hoàn toàn nó.

Nếu đây là gói phần mềm đầu tiên của bạn, bạn nên tạo các gói phần mềm giả với các phiên bản khác nhau để kiểm tra gói của bạn trước để tránh các vấn đề trong tương lai.

Hãy nhớ rằng nếu gói phần mềm của bạn đã được tung ra trước đây trong Debian, mọi người sẽ thông thường nâng cấp nó lên gói phần mềm của bạn từ phiên bản vốn nằm trong bản phân phối Debian trước. Hãy nhớ kiểm tra việc nâng cấp từ phiên bản đó.

Mặc dù giảm cấp không được hỗ trợ chính thức, hỗ trợ nó là một cử chỉ thân thiện.

7.4 Sử dụng `lintian`

Chạy lintian(1) qua tập tin `.changes` của bạn. Lệnh **lintian** chạy nhiều kịch bản kiểm tra các lỗi đóng gói thường gặp. [1]

```
$ lintian -i -I --show-overrides gentoo_0.9.12-1_i386.changes
```

Tất nhiên, thay thế tên tập tin với tên của tập tin `.changes` được tạo ra cho gói phần mềm của bạn. Kết quả của lệnh **lintian** sử dụng các cờ sau:

- `E:` cho lỗi; một lỗi đóng gói hoặc một vi phạm chắc chắn về chính sách.
- `W:` cho cảnh báo; một lỗi đóng gói hoặc một vi phạm có thể liên quan đến chính sách.
- `I:` cho thông tin; thông tin về một số mặt của việc đóng gói.
- `N:` cho ghi chú; a thông điệp chi tiết để hỗ trợ tìm lỗi.
- `O:` cho đè lên ngầm định; một thông điệp bị đè lên bởi các tập tin `lintian-overrides` nhưng hiện ra bởi tham số `--show-overrides`.

Khi bạn thấy các cảnh báo, tùy chỉnh gói phần mềm để tránh chúng hoặc kiểm tra để chắc rang các cảnh báo đó là cảnh báo nhầm. Nếu là nhầm, hãy thiết đặt các tập tin `lintian-overrides` như mô tả trong Phần 5.14.

Ghi chú rằng bạn có thể biên dịch gói phần mềm với **dpkg-buildpackage** và chạy **lintian** trên nó trong một dòng lệnh, nếu bạn sử dụng debuild(1) hay pdebuild(1).

[1] Bạn không cần phải cung cấp **lintian** tham số `-i -I --show-overrides` nếu bạn tùy chỉnh `/etc/devscripts.conf` hoặc `~/.devscripts` như mô tả trong Phần 6.3.

7.5 Lệnh debc

Bạn có thể liệt kê các tập tin trong gói phần mềm Debian nhị phân với lệnh debc(1).

```
$ debc package.changes
```

7.6 Lệnh debdiff

Bạn có thể so sánh nội dung tập tin của hai gói phần mềm Debian chứa mã nguồn với lệnh debdiff(1).

```
$ debdiff old-package.dsc new-package.dsc
```

Bạn có thể so sánh danh sách tập tin trong hai gói phần mềm Debian nhị phân với lệnh debdiff(1).

```
$ debdiff old-package.changes new-package.changes
```

Những lệnh này rất hữu dụng để phát hiện cái gì đã thay đổi trong gói phần mềm nguồn và để tìm ra những thay đổi vô ý tạo ra trong khi cập nhật các gói phần mềm nhị phân, như vô ý đặt tập tin sai địa điểm hoặc vô ý xóa tập tin.

7.7 Lệnh interdiff

Bạn có thể so sánh hai tập tin `diff.gz` với lệnh interdiff(1). Lệnh này rất hữu dụng để chắc chắn rằng không có thay đổi vô ý nào được tạo ra trong mã nguồn bởi người bảo trì trong quá trình tạo các gói phần mềm dưới định dạng nguồn cũ `1.0`.

```
$ interdiff -z old-package.diff.gz new-package.diff.gz
```

Định dạng nguồn mới `3.0` lưu trữ những thay đổi trong nhiều tập tin vá như mô tả ở Phần 5.25. Bạn cũng có thể theo dõi những thay đổi của tập tin `debian/patches/*` dùng lệnh **interdiff**.

7.8 Lệnh mc

Rất nhiều thao tác kiểm soát tập tin này có thể được gộp chung vào một quá trình trực quan bởi một trình quản lý tập tin như mc(1) với khả năng cho phép bạn duyệt không chỉ nội dung của các tập tin gói phần mềm `*.deb` mà cả các tập tin `*.udeb`, `*.debian.tar.gz`, `*.diff.gz`, và `*.orig.tar.gz`.

Hãy quan sát cả những tập tin bổ sung không cần thiết và những tập tin không có nội dung, trong cả gói phần mềm nhị phân và nguồn. Thông thường những phần thừa không được dọn dẹp sạch sẽ; thay đổi tập tin `rules` của bạn để bù lại điều này.

Chapter 8

Cập nhật gói phần mềm

Sau khi bạn tung ra một gói phần mềm, bạn sẽ sớm cần phải cập nhật nó.

8.1 Phiên bản tu chỉnh Debian mới

Hãy giả sử là một báo cáo lỗi được gửi về gói phần mềm của bạn như `#654321`, và nó mô tả một vấn đề mà bạn có thể xử lý. Đây là những gì bạn cần làm để tạo một bản tu chỉnh mới của gói phần mềm:

- Nếu nó sẽ được ghi lại như là một bản vá, hãy làm như sau:
  - `dquilt new` *`bugname.patch`* để đặt tên cho bản vá;
  - `dquilt add` *`buggy-file`* để khai báo tập tin sẽ được điều chỉnh;
  - Sửa vấn đề trong nguồn của gói phần mềm cho một lỗi từ thượng nguồn
  - `dquilt refresh` để ghi lại vào *`bugname.patch`*;
  - `dquilt header -e` để thêm vào mô tả của nó;
- Nếu nó dùng để cập nhật một bản vá đang tồn tại, hãy làm như sau:
  - `dquilt pop` *`foo.patch`* để hủy bỏ bản vá hiện tại *`foo.patch`*;
  - Sửa vấn đề trong bản cũ *`foo.patch`*;
  - `dquilt refresh` để cập nhật *`foo.patch`*;
  - `dquilt header -e` để cập nhật mô tả của nó;
  - `while dquilt push; do dquilt refresh; done` để áp dụng tất cả bản vá trong khi tháo bỏ *fuzz*;
- Thêm một bản tu chỉnh mới vào đầu của tập tin Debian `changelog`, ví dụ như chạy lệnh `dch -i`, hoặc cụ thể hơn với lệnh `dch -v` *`version-revision`* và sau đó chèn ghi chú vào bằng trình soạn thảo văn bản bạn thích. [1]
- Chèn một đoạn mô tả ngắn của lỗi và giải pháp trong mục nhật ký thay đổi, theo sau bởi `Closes: #654321`. Bằng cách đó, báo cáo lỗi sẽ được *tự động* đóng bởi phần mềm bảo trì kho lưu trữ ngay lúc gói phần mềm của bạn được nhận vào kho lưu trữ của Debian.
- Lặp lại những gì bạn đã làm phía trên để sửa nhiều lỗi trong quá trình cập nhật tập tin Debian `changelog` với lệnh `dch` nếu cần thiết.
- Lặp lại những gì bạn đã làm ở Phần 6.1 và Chương 7.

[1] Để lấy thông tin ngày dưới định dạng bị bắt buộc, dùng `LANG=C date -R`.

- Khi bạn đã hài lòng, bạn nên thay đổi giá trị bản phân phối trong tập tin `changelog` từ `UNRELEASED` sang giá trị bản phân phối đích `unstable` (hoặc thậm chí `experimental`).[2]
- Tải lên gói phần như ở Chương 9. Sự khác biệt là lần này, tập tin nén chứa mã nguồn nguyên vẹn ban đầu sẽ không được đính kèm, bởi vì nó không có thay đổi gì và nó đã tồn tại trong kho lưu trữ của Debian.

Một tình huống khó có thể xảy ra khi bạn tạo một gói phần mềm tại chỗ để thí nghiệm với quá trình đóng gói trước khi tải phiên bản bình thường lên kho lưu trữ chính thức, ví dụ như *`1.0.1-1`*. Để nâng cấp trơn tru hơn, sẽ là một ý hay nếu tạo một chỉ mục trong tập tin `changelog` với phiên bản là một chuỗi ký tự như *`1.0.1-1~rc1`*. Bạn có thể sắp xếp lại nội dung tập tin `changelog` bằng cách gộp nhiều chỉ mục nhật ký thay đổi được tạo ra tại chỗ thành một chỉ mục duy nhất cho gói phần mềm chính thức. Xem Phần 2.6 để biết thêm về thứ tự của các chuỗi ký tự dành cho phiên bản.

8.2 Thẩm định phiên bản mới phát hành từ thượng nguồn

Khi chuẩn bị các gói phần mềm của một phiên bản mới được phát hành từ thượng nguồn cho kho lưu trữ Debian, bạn phải kiểm tra bản phát hành mới từ thượng nguồn trước tiên.

Bắt đầu bằng việc đọc tập tin thượng nguồn `changelog`, `NEWS`, và bất kỳ tài liệu nào khác họ có thể vừa phát hành với phiên bản mới.

Bạn có thể tiếp theo thẩm định những thay đổi giữa mã nguồn của bản cũ và bản mới từ thượng nguồn như sau, quan sát kỹ những gì đáng nghi:

```
$ diff -urN foo-oldversion foo-newversion
```

Những thay đổi liên quan tới một vài tập tin tự động được tạo ra bởi Autotools như là `missing`, `aclocal.m4`, `config.guess`, `config.h.in`, `config.sub`, `configure`, `depcomp`, `install-sh`, `ltmain.sh`, và `Makefile.in` có thể bị phớt lờ. Bạn có thể xóa nó đi trước khi chạy lệnh **diff** trên mã nguồn để thẩm định.

8.3 Phiên bản phát hành mới từ thượng nguồn.

Nếu một gói phần mềm *`foo`* được cấu hình đúng cách theo các định dạng mới `3.0 (native)` or `3.0 (quilt)`, đóng gói một phiên bản thượng nguồn mới chủ yếu chỉ là di chuyển thư mục `debian` cũ sang mã nguồn mới. Công việc này có thể được thực hiện bằng cách chạy lệnh `tar xvzf` */path/to/foo_oldversion*`.debian.tar.gz` tại nơi giải nén mã nguồn mới.[3] Dĩ nhiên, bạn cần phải làm một số công việc chắc chắn phải làm sau:

- Tạo một bản sao của mã nguồn từ thượng nguồn như là tập tin `foo_`*`newversion`*`.orig.tar.gz`.
- Cập nhật tập tin Debian `changelog` với lệnh `dch -v` *`newversion-1`*.
  - Tạo một mục mới với chuỗi ký tự `New upstream release`
  - Mô tả cụ thể những thay đổi *trong bản phát hành mới từ thượng nguồn* sửa được các lỗi đã báo cáo và đóng các lỗi đó bằng việc chèn thêm `Closes:` *`#bug_number`*.
  - Mô tả cụ thể những thay đổi xảy ra bởi người bảo trì *đối với bản phát hành từ thượng nguồn* mà sửa được các lỗi đã báo cáo và đóng các lỗi đó bằng hành động chèn thêm `Closes:` *`#bug_number`*.
- `while dquilt push; do dquilt refresh; done` để áp dụng tất cả các bản vá trong lúc tháo chúng ra *fuzz*.

Nếu bản vá/ghép không áp dụng được sạch sẽ, thẩm định tình hình (gợi ý được để trong các tập tin `.rej`).

[2] Nếu bạn sử dụng lệnh `dch -r` để tạo thay đổi cuối này, vui lòng đảm bảo bạn đã lưu tệp `changelog` một cách rõ ràng bởi trình soạn thảo

[3] Nếu một gói phần mềm *`foo`* được đóng gói theo định dạng cũ `1.0`, công việc này có thể được hoàn thành bằng việc chạy lệnh `zcat` */path/to/foo_oldversion*`.diff.gz|patch -p1`

- Nếu một bản vá bạn đã áp dụng với mã nguồn được ghép vào mã nguồn từ thượng nguồn,
  - chạy lệnh `dquilt delete` để tháo bỏ nó ra.
- Nếu một bản vá bạn đã áp dụng vào mã nguồn xung đột với những thay đổi mới đến từ mã nguồn mới của thượng nguồn,
  - `dquilt push -f` để áp dụng những bản vá cũ trong khi ép buộc các từ chối vá từ tập tin *`baz`*`.rej`.
  - Sửa tập tin *`baz`* bằng tay để có được hiệu quả mong muốn của tập tin *`baz`*`.rej`.
  - `dquilt refresh` để cập nhật bản vá
- Tiếp tục như bình thường với `while dquilt push; do dquilt refresh; done`.

Quá trình này có thể được tự động hóa bằng cách chạy lệnh uupdate(1) như sau:

```
$ apt-get source foo
...
dpkg-source: info: extracting foo in foo-oldversion
dpkg-source: info: unpacking foo_oldversion.orig.tar.gz
dpkg-source: info: applying foo_oldversion-1.debian.tar.gz
$ ls -F
foo-oldversion/
foo_oldversion-1.debian.tar.gz
foo_oldversion-1.dsc
foo_oldversion.orig.tar.gz
$ wget http://example.org/foo/foo-newversion.tar.gz
$ cd foo-oldversion
$ uupdate -v newversion ../foo-newversion.tar.gz
$ cd ../foo-newversion
$ while dquilt push; do dquilt refresh; done
$ dch
... document changes made
```

Nếu bạn tạo một tập tin `debian/watch` như được mô tả ở Phần 5.21, bạn nên bỏ qua lệnh **wget**. Bạn đơn giản chỉ chạy lệnh uscan(1) trong thư mục *`foo-oldversion`* thay vì lệnh **uupdate**. Hành động này *tự động* tìm mã nguồn mới, tải về, và chạy lệnh **uupdate**. [4]

Bạn có thể phát hành mã nguồn mới này bằng việc lặp lại những gì bạn đã làm ở Phần 6.1, Chương 7, và Chương 9.

8.4 Cập nhật kiểu đóng gói

Cập nhật cách đóng gói không phải là một hành động bắt buộc cho việc cập nhật một gói phần mềm. Tuy nhiên, làm vậy sẽ cho phép bạn sử dụng toàn bộ khả năng của hệ thống `debhelper` hiện đại và định dạng gói nguồn `3.0`. [5]

- Nếu bạn cần tái tạo những tập tin khuôn mẫu đã bị xóa vì lý do gì đó, bạn có thể chạy lệnh **dh_make** lại trong cây thư mục của gói phần mềm nguồn Debian với tham số `--addmissing` option. Sau đó điều chỉnh chúng cho phù hợp.
- Nếu gói phần mềm không được cập nhật để sử dụng cú pháp `debhelper` v7+ **dh** cho tập tin `debian/rules`, cập nhật nó để sử dụng **dh**. Cập nhật tập tin `debian/control` tương ứng.
- Nếu bạn muốn cập nhật tập tin `rules` được tạo với cơ chế đính kèm `Makefile` của Common Debian Build System (`cdbs`) (Hệ Thống Biên Dịch Chung Debian) lên cú pháp **dh**, xem phần sau để hiểu các biến cấu hình `DEB_*`.
  - bản sao tại chỗ của `/usr/share/doc/cdbs/cdbs-doc.pdf.gz`

[4] Nếu lệnh **uscan** tải về mã nguồn mới nhưng nó không chạy lệnh **uupdate**, bạn nên sửa tập tin `debian/watch` để có dòng `debian uupdate` tại cuối của đường dẫn URL.

[5] Nếu nhà tài trợ của bạn hoặc các nhà bảo trì khác phản đối việc cập nhật cách đóng gói hiện tại, đừng phí thời gian tranh cãi. Có nhiều việc quan trọng hơn cần phải làm.

  – The Common Debian Build System (CDBS), FOSDEM 2009 (http://meetings-archive.debian.net/pub/debian-meetings/-2009/fosdem/slides/The_Common_Debian_Build_System_CDBS/)

- Nếu bạn có một gói phần mềm nguồn với định dạng `1.0` mà không có tập tin *foo*`.diff.gz`, bạn có thể cập nhật nó lên định dạng nguồn `3.0 (native)` mới hơn bằng việc tạo tập tin `debian/source/format` chứa dòng `3.0 (native)`. Các tập tin `debian/*` còn lại có thể được sao chép sang.
- Nếu bạn có một gói phần mềm nguồn với định dạng `1.0` mà không có tập tin *foo*`.diff.gz`, bạn có thể cập nhật nó lên định dạng nguồn `3.0 (native)` mới hơn bằng việc tạo tập tin `debian/source/format` chứa dòng `3.0 (native)`. Các tập tin `debian/*` còn lại có thể được sao chép sang. Nhập tập tin `big.diff` được tạo ra bởi lệnh `filterdiff -z -x '*/debian/*' foo.diff.gz > big.diff` vào hệ thống **quilt** của bạn, nếu cần thiết. [6]
- Nếu nó được đóng gói bằng một hệ thống vá khác như là `dpatch`, `dbs`, hay `cdbs` với `-p0`, `-p1`, or `-p2`, chuyển đổi nó sang `quilt` bằng việc sử dụng `deb3` tại http://bugs.debian.org/581186.
- Nếu nó được đóng gói với lệnh **dh** với tham số `--with quilt` hay với các lệnh **dh_quilt_patch** và **dh_quilt_unpatch**, bỏ những thứ này và buộc nó sử dụng định dạng gói nguồn `3.0 (quilt)` mới hơn.

Bạn nên kiểm tra DEP - Debian Enhancement Proposals (http://dep.debian.net/) (Những đề nghị cải tiến Debian) và sử dụng những đề nghị đã được CHẤP THUẬN.

Bạn cũng cần phải thực hiện các công việc khác mô tả ở Phần 8.3.

8.5 Chuyển đổi sang UTF-8

Nếu các văn bản thượng nguồn được mã hóa theo kiểu mã hóa cũ, chuyển đổi nó sang UTF-8 là một ý hay.

- Sử dụng iconv(1) để chuyển đổi mã hóa của các tập tin văn bản thô.

```
iconv -f latin1 -t utf8 foo_in.txt > foo_out.txt
```

- Sử dụng w3m(1) để chuyển đổi các tập tin HTML sang các tập tin văn bản thô UTF-8. Khi bạn làm điều này, hãy chắc chắn rằng bạn chạy nó dưới môi trường UTF-8.

```
LC_ALL=en_US.UTF-8 w3m -o display_charset=UTF-8 \
        -cols 70 -dump -no-graph -T text/html \
        < foo_in.html > foo_out.txt
```

8.6 Các nhắc nhở cho việc cập nhật các gói phần mềm

Sau đây là vài nhắc nhở cho việc cập nhật các phần mềm:

- Bảo tồn các chỉ mục cũ trong tập tin `changelog` (nghe có vẻ quá dĩ nhiên, nhưng đã có các tình huống nhiều người đánh lệnh `dch` trong khi họ đáng lẽ phải đánh `dch -i`.)
- Các thay đổi Debian hiện tại cần phải được đánh giá lại; ném đi những thứ thượng nguồn đã tích hợp (dưới kiểu này hay kiểu khác) và ghi nhớ giữ những thứ mà thượng nguồn chưa tích hợp, trừ khi có một lý do đáng để không làm thế.
- Nếu bất kỳ thay đổi nào được tạo ra đối với hệ thống biên dịch (hi vọng bạn sẽ biết từ việc thẩm định các thay đổi từ thượng nguồn), hãy cập nhật các điều kiện biên dịch của `debian/rules` và `debian/control` nếu cần thiết.

[6] Bạn có thể chặt tập tin `big.diff` ra nhiều bản vá nhỏ liên tiếp dùng lệnh **splitdiff**.

- Kiểm tra Debian Bug Tracking System (BTS) (http://www.debian.org/Bugs/) (Hệ thống theo dõi lỗi Debian) để xem có ai đã cung cấp bản vá cho các lỗi đang mở.
- Kiểm tra những nội dung của tập tin `.changes` để chắc chắn rằng bạn đang tải lên đúng bản phân phối, các lỗi được đóng đúng cách được liệt kê trong trường `Closes`, trường `Maintainer` và `Changed-By` phải khớp, tập tin phải được ký bằng GPG, v.v.

Chapter 9

Tải gói phần mềm lên

Giờ bạn đã kiểm tra gói phần mềm mới của bạn kỹ càng, bạn muốn phát hành nó trên kho công cộng để chia sẻ nó.

9.1 Tải nó lên kho lưu trữ Debian

Khi bạn trở thành một nhà phát triển chính thức, [1] bạn có thể tải gói phần mềm lên kho lưu trữ Debian. [2] Bạn có thể làm điều này bằng tay, nhưng sẽ dễ hơn nếu bạn sử dụng các công cụ tự động sẵn có như dupload(1) hay dput(1). Chúng tôi sẽ mô tả công việc sẽ được thực hiện ra sao với **dupload**. [3]

Đầu tiên bạn phải tạo tập tin cấu hình của **dupload**. Bạn có thể chọn sửa tập tin cấu hình tầm hệ thống `/etc/dupload.conf`, hoặc tạo tập tin `~/.dupload.conf` riêng của bạn vượt quyền cấp trên ở những cài đặt bạn muốn thay đổi.

Bạn có thể đọc trang tham khảo dupload.conf(5) để hiểu ý nghĩa của từng tham số ở đây.

Tham số `$default_host` quyết định hàng đợi tải lên nào sẽ được sử dụng ngầm định. `anonymous-ftp-master` là hàng đợi chính, nhưng có thể bạn sẽ muốn sử dụng một hàng đợi khác. [4]

Khi được kết nối tới Internet, bạn có thể tải gói phần mềm của bạn lên như sau:

```
$ dupload gentoo_0.9.12-1_i386.changes
```

dupload kiểm tra rằng các mã tập tin SHA1/SHA256 khớp với các mã được liệt kê trong tập tin `.changes`. Nếu chúng không khớp, nó sẽ cảnh báo bạn để tái biên dịch như mô tả ở Phần 6.1 để nó có thể được tải lên đúng cách.

Nếu bạn gặp phải một vấn đề tải lên tại ftp://ftp.upload.debian.org/pub/UploadQueue/, bạn có thể sửa nó bằng việc tải lên bằng tay một tập tin `*.commands` ký theo kiểu GPG với **ftp**. [5] Ví dụ, dùng `hello.commands`:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
Uploader: Foo Bar <Foo.Bar@example.org>
Commands:
```

[1] Xem Phần 1.1.

[2] Có các kho lưu trữ truy cập công cộng như là http://mentors.debian.net/ vốn hoạt động gần như không khác gì với kho lưu trữ Debian và cung cấp một khu vực tải lên cho những người không phải DD(nhà phát triển Debian). Bạn có thể tự tạo một kho tương tự sử dụng các công cụ tại http://wiki.debian.org/-HowToSetupADebianRepository. Vì thế phần này cũng sẽ hữu dụng cho những người không phải DD.

[3] Gói phần mềm `dput` có vẻ đến với nhiều tính năng và có vẻ đang trở nên thông dụng hơn gói phần mềm `dupload`. Nó sử dụng tập tin `/etc/dput` cho việc cấu hình mọi nơi của nó và tập tin `~/.dput.cf` cho việc cấu hình từng người dùng. Nó cũng hỗ trợ đầy đủ các dịch vụ liên quan đến Ubuntu.

[4] Xem Debian Developer's Reference 5.6, "Uploading a package" (http://www.debian.org/doc/manuals/developers-reference/pkgs.html#upload) .

[5] Xem ftp://ftp.upload.debian.org/pub/UploadQueue/README. Ngoài ra, bạn có thể dùng lệnh **dcut** từ gói phần mềm `dput`.

```
 rm hello_1.0-1_i386.deb
 mv hello_1.0-1.dsx hello_1.0-1.dsc
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.10 (GNU/Linux)

[...]
-----END PGP SIGNATURE-----
```

9.2 Đính kèm `orig.tar.gz` cho việc tải lên

Khi bạn tải gói phần mềm lên kho lưu trữ lần đầu tiên, bạn cũng cần đính kèm tập tin nguồn nguyên gốc `orig.tar.gz`. Nếu mã phiên bản Debian của gói phần mềm này không phải là `1` hay `0`, bạn phải cung cấp tham số `-sa` của dòng lệnh **dpkg-buildpackage**.

Đối với dòng lệnh **dpkg-buildpackage**:

```
$ dpkg-buildpackage -sa
```

Đối với dòng lệnh **debuild**:

```
$ debuild -sa
```

For the **pdebuild** command:

```
$ pdebuild --debbuildopts -sa
```

Ngược lại, tham số `-sd` sẽ buộc việc loại trừ tập tin nguồn nguyên gốc `orig.tar.gz`.

9.3 Những lần tải lên bị bỏ qua

Nếu bạn tạo quá nhiều mục trong tập tin `debian/changelog` bằng việc bỏ qua các lần tải lên, bạn phải tạo một tập tin đúng quy định `*_.changes` bao gồm tất cả những thay đổi từ lần tải lên cuối cùng. Điều này có thể được thực hiện bằng việc đặt giá trị phiên bản cho tham số `-v` của lệnh **dpkg-buildpackage**, ví dụ, *`1.2`*.

Đối với dòng lệnh **dpkg-buildpackage**:

```
$ dpkg-buildpackage -v1.2
```

Đối với dòng lệnh **debuild**:

```
$ debuild -v1.2
```

For the **pdebuild** command:

```
$ pdebuild --debbuildopts "-v1.2"
```

Appendix A

Đóng gói nâng cao

Sau đây là một vài gợi ý và chỉ dẫn cho các đề tài đóng gói nâng cáo mà bạn rất có thể phải đối mặt. Bạn rất được khuyến khích đọc tất cả những tài liệu liên quan được đề nghị ở đây.

Bạn có thể cần phải thay đổi các tập tin khuôn mẫu đóng gói tạo ra bởi lệnh **dh_make** để biết được các đề tài được nói tới trong chương này. Lệnh **debmake** mới hơn có thể sẽ nói về các đề tài này tốt hơn.

A.1 Các thư viện được chia sẻ

Trước khi đóng gói các thư viện được chia sẻ, bạn nên đọc những tài liệu liên quan sau một cách chi tiết:

- Hướng dẫn về Chính sách Debian, 8 "Các Thư viện Chia sẻ" (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html)
- Hướng dẫn về Chính sách Debian, 9.1.1 "Cấu trúc Hệ thống tập tin" (http://www.debian.org/doc/debian-policy/ch-opersys.html#s-fhs)
- Hướng dẫn về Chính sách Debian, 10.2 "Các Thư viện" (http://www.debian.org/doc/debian-policy/ch-files.html#s-libraries)

Sau đây là một số gợi ý được đơn giản hóa quá mức để giúp bạn bắt đầu:

- Các thư viện chia sẻ là các tập tin đối tượng ELF chứa mã biên dịch.
- Các thư viện chia sẻ được phân phối bởi các tập tin `*.so`. (Không phải các tập tin `*.a` hay các tập tin `*.la`)
- Các thư viện chia sẻ được sử dụng phần lớn để chia sẻ những mã thông dụng giữa nhiều tập tin thực thi với cơ chế **ld**.
- Các thư viện chia sẻ thỉnh thoảng được sử dụng để cung cấp nhiều phần mở rộng có thể cắm vào một tập tin thực thi với cơ chế **dlopen**.
- Các thư viện chia sẻ xuất các ký hiệu đại diện cho các đối tượng được biên dịch như là các tham biến, các hàm, và các class; và cho phép truy cập tới chúng từ các tập tin thực thi được kết nối tới chúng.
- SONAME của một thư viện chia sẻ `lib`*`foo`*`.so.`*`1`*: `objdump -p lib`*`foo`*`.so.`*`1` `| grep SONAME` [1]
- SONAME của một thư viện chia sẻ thường khớp với tên tập tin thư viện (nhưng không phải luôn luôn như vậy).
- SONAME của các thư viện chia sẻ được kết nối tới *`/usr/bin/foo`*: `objdump -p` *`/usr/bin/foo`* `| grep NEEDED` [2]
- `lib`*`foo1`*: gói phần mềm thư viện cho thư viện chia sẻ `lib`*`foo`*`.so.`*`1` với phiên bản SONAME ABI *`1`*.[3]

[1] Ngoài ra: `readelf -d lib`*`foo`*`.so.`*`1` `| grep SONAME`
[2] Ngoài ra: `readelf -d lib`*`foo`*`.so.`*`1` `| grep NEEDED`
[3] Xem Hướng dẫn về Chính sách Debian, 8.1 "Các Thư viện chia sẻ khi chạy" (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-runtime) .

- Các kịch bản bảo trì gói phần mềm của gói phần mềm thư viện phải gọi lệnh **ldconfig** dưới các tình huống cụ thể để tạo các kết nối tượng trưng cho SONAME.[4]
- `lib`*`foo1`*`-dbg`: gói phần mềm ký hiệu dò lỗ chứa các ký hiệu dò lỗi cho gói phần mềm thư viện chia sẻ `lib`*`foo1`*.
- `lib`*`foo`*`-dev`: gói phần mềm phát triển chứa những tập tin tiêu đề v.v. cho thư viện chia sẻ `lib`*`foo`*`.so.`*`1`*.[5]
- Các gói Debian không nên chứa các tập tin lưu trữ `*.la` nói chung.[6]
- Gói phần mềm Debian nói chung không nên sử dụng RPATH.[7]
- Mặc dù nó có thể đã lạc hậu và chỉ tồn tại như là tài liêu tham khảo thứ cấp, Hướng dẫn Đóng gói thư viện Debian (http://www.netfort.gr.jp/~dancer/column/libpkg-guide/libpkg-guide.html) có thể vẫn hữu dụng.

A.2 Quản lý `debian/`*`package`*`.symbols`

Khi bạn đóng gói một thư viện chia sẻ, bạn nên tạo một tập tin `debian/`*`package`*`.symbols` để quản lý phiên bản thấp nhất liên quan tới từng ký hiệu cho các thay đổi ABI tương thích ngược dưới cùng một SONAME của thư viện cho cùng một tên gói phần mềm thư viện.[8] Bạn nên đọc những tài liệu liên quan chính thống sau đây một cách chi tiết:

- Hướng dẫn Chính sách Debian, 8.6.3 "Hệ thống ký hiệu" (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-symbols) [9]
- dh_makeshlibs(1)
- dpkg-gensymbols(1)
- dpkg-shlibdeps(1)
- deb-symbols(5)

Sau đây là một ví dụ sơ khảo để tạo một gói phần mềm `libfoo1` tương ứng với phiên bản thượng nguồn `1.3` với tập tin `debian/libfoo1.symbols` phù hợp:

- Chuẩn bị một sườn cho cây mã nguồn Debian hóa sử dụng tập tin thượng nguồn `libfoo-1.3.tar.gz`.
  - Nếu đây là lần đóng gói đầu tiên của gói phần mềm `libfoo1`, tạo một tập tin `debian/libfoo1.symbols` rỗng.
  - Nếu phiên bản thượng nguồn trước đó `1.2` được đóng gói thành gói phần mềm `libfoo1` với một tập tin `debian/libfoo1.symbols` phù hợp trong gói phần mềm nguồn của nó, hãy sử dụng lại tập tin đó.
  - Nếu bản thượng nguồn trước đó `1.2` không được đóng gói với tập tin `debian/libfoo1.symbols`, tạo một tập tin như là tập tin `symbols` từ tất cả các gói phần mềm nhị phân có sẵn của một tên gói phần mềm thư viện chứa cùng SONAME của một thư viện, ví dụ, các phiên bản `1.1-1` và `1.2-1`. [10]

[4] Xem Hướng dẫn về Chính sách Debian, 8.1.1 "ldconfig" (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-ldconfig) .

[5] Xem Hướng dẫn Chính sách Debian, 10.2 "Các Thư viện tĩnh" (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-static) và Hướng dẫn về Chính sách Debian, 8.4 "Các Tập tin phát triển" (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-dev)

[6] Xem Debian wiki ReleaseGoals/LAFileRemoval (http://wiki.debian.org/ReleaseGoals/LAFileRemoval) .

[7] Xem Debian wiki RpathIssue (http://wiki.debian.org/RpathIssue) .

[8] Các thay đổi ABI không tương thích ngược thông thường cần bạn phải cập nhật SONAME của thư viện và tên gói phần mềm thư viện chia sẻ sang những tên mới.

[9] Với các thư viện C++ và các tình huống khác mà theo dõi từng ký hiệu là vô cùng khó khăn, thay vì như trên hãy làm theo Hướng dẫn Chính sách về Debian, 8.6.4 "Hệ thống shlibs" (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-shlibdeps) ,

[10] Tất cả các phiên bản trước đó của các gói phần mềm Debian có sẵn tại http://snapshot.debian.org/ (http://snapshot.debian.org/) . Phiên bản tu chỉnh Debian được cắt bỏ khỏi phiên bản để giúp việc backport gói phần mềm (tạo gói phần mềm chứa các cập nhật mới cho các bản phân phối Debian cũ) dễ hơn: `1.1` << `1.1-1~bpo70+1` << `1.1-1` và `1.2` << `1.2-1~bpo70+1` << `1.2-1`

```
$ dpkg-deb -x libfoo1_1.1-1.deb libfoo1_1.1-1
$ dpkg-deb -x libfoo1_1.2-1.deb libfoo1_1.2-1
$ : > symbols
$ dpkg-gensymbols -v1.1 -plibfoo1 -Plibfoo1_1.1-1 -Osymbols
$ dpkg-gensymbols -v1.2 -plibfoo1 -Plibfoo1_1.2-1 -Osymbols
```

- Tạo các bản biên dịch thử của cây mã nguồn với các công cụ như là **debuild** và **pdebuild**. (Nếu điều này thất bại do thiếu ký hiệu v.v., đã có một số thay đổi ABI không tương thích ngược buộc bạn phải tăng tên gói phần mềm thư viện chia sẻ lên một tên như là `libfoo1a` và bạn nên bắt đầu lại từ đầu.)

```
$ cd libfoo-1.3
$ debuild
...
dpkg-gensymbols: warning: some new symbols appeared in the symbols file: ...
 see diff output below
--- debian/libfoo1.symbols (libfoo1_1.3-1_amd64)
+++ dpkg-gensymbolsFE5gzx        2012-11-11 02:24:53.609667389 +0900
@@ -127,6 +127,7 @@
  foo_get_name@Base 1.1
  foo_get_longname@Base 1.2
  foo_get_type@Base 1.1
+ foo_get_longtype@Base 1.3-1
  foo_get_symbol@Base 1.1
  foo_get_rank@Base 1.1
  foo_new@Base 1.1
...
```

- Nếu bạn thấy sự khác biệt in ra bởi lệnh **dpkg-gensymbols** như trên, giải nén tập tin `symbols` phù hợp cập nhật mới nhất từ gói phần mềm nhị phân được tạo ra của thư viện chia sẻ. [11]

```
$ cd ..
$ dpkg-deb -R  libfoo1_1.3_amd64.deb libfoo1-tmp
$ sed -e 's/1\.3-1/1\.3/' libfoo1-tmp/DEBIAN/symbols \
        >libfoo-1.3/debian/libfoo1.symbols
```

- Biên dịch các gói phần mềm phát hành với các công cụ như **debuild** và **pdebuild**.

```
$ cd libfoo-1.3
$ debuild -- clean
$ debuild
...
```

Bên cạnh các ví dụ ở trên, chúng ta cần phải kiểm tra tương thích ABI hơn nữa và tăng các phiên bản cho một vài ký hiệu bằng tay nếu cần. [12]

Mặc dù đây chỉ là một tài liệu tham khảo thứ cấp, Debian wiki UsingSymbolsFiles (http://wiki.debian.org/UsingSymbolsFiles) và những trang điện tử kết nối tới nó có thể rất hữu dụng.

[11] Phiên bản tu chỉnh Debian được cắt bỏ khỏi phiên bản để giúp việc backport gói phần mềm (tạo gói phần mềm chứa các cập nhật mới cho các bản phân phối Debian cũ) dễ hơn: `1.3` << `1.3-1~bpo70+1` << `1.3-1`

[12] Xem Hướng dẫn về Chính sách Debian, 8.6.2 "Các Thay đổi ABI của Thư viện chia sẻ" (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-updates)

A.3 Multiarch

Tính năng multiarch được giới thiệu ở Debian wheezy tích hợp hỗ trợ cho cài đặt xuyên nền tảng của các gói phần mềm nhị phân (đặc biệt `i386`<->`amd64`, nhưng cũng một vài kết hợp khác) trong `dpkg` và `apt`. Bạn nên đọc các tài liệu tham khảo sau một cách chi tiết:

- Ubuntu wiki MultiarchSpec (https://wiki.ubuntu.com/MultiarchSpec) (thượng nguồn)
- Debian wiki Multiarch/Implementation (http://wiki.debian.org/Multiarch/Implementation) (tình huống Debian)

Nó sử dụng bộ ba như là `i386-linux-gnu` và `x86_64-linux-gnu` cho đường dẫn cài đặt của các thư viện chia sẻ. Đường dẫn bộ ba thực sự được đặt giá trị linh động theo tham biến `$(DEB_HOST_MULTIARCH)` dùng lệnh dpkg-architecture(1) cho từng lần biên dịch gói phần mềm nhị phân. Ví dụ, đường dẫn để cài đặt các thư viện multiarch được thay đổi như sau:[13]

Đường dẫn cũ	Đường dẫn i386 multiarch	Đường dẫn amd64 multiarch
`/lib/`	`/lib/i386-linux-gnu/`	`/lib/x86_64-linux-gnu/`
`/usr/lib/`	`/usr/lib/i386-linux-gnu/`	`/usr/lib/x86_64-linux-gnu/`

Sau đây là một vài ví dụ về tình huống băm gói phần mềm multiarch tiêu biểu cho các gói:

- một gói phần mềm nguồn cho `lib`*`foo`*`-1.tar.gz`
- một gói phần mềm nguồn công cụ *`bar`*`-1.tar.gz` viết bằng một ngôn ngữ biên dịch.
- một gói phần mềm nguồn công cụ *`baz`*`-1.tar.gz` viết bằng một ngôn ngữ thông dịch

Package	Architecture:	Multi-Arch:	Nội dung gói phần mềm
`lib`*`foo1`*	bất kỳ	không đổi	thư viện được chia sẻ, có thể cài cùng lúc
`lib`*`foo1`*`-dbg`	bất kỳ	không đổi	các ký hiệu dò lỗi của thư viện chia sẻ, có thể cài cùng lúc
`lib`*`foo`*`-dev`	bất kỳ	không đổi	các tập tin tiêu đề của thư viện chia sẻ v.v., có thể cài cùng lúc
`lib`*`foo`*`-tools`	bất kỳ	Xa lạ	Các chương trình hỗ trợ khi chạy, không thể cài đặt cùng lúc
`lib`*`foo`*`-doc`	tất cả	Xa lạ	những tập tin tài liệu hướng dẫn của thư viện chia sẻ
`bar`	bất kỳ	Xa lạ	các tập tin chương trình đã biên dịch, không thể cài đặt cùng lúc
`bar``-doc`	tất cả	Xa lạ	những tập tin tài liệu hướng dẫn cho chương trình
`baz`	tất cả	Xa lạ	các tập tin chương trình được thông dịch

Xin chú ý rằng gói phần mềm phát triển nên chứa một kết nối tượng trưng cho thư viện chia sẻ tương ứng **không có số phiên bản**. Ví dụ: `/usr/lib/x86_64-linux-gnu/libfoo.so` -> `libfoo.so.1`

A.4 Biên dịch một gói phần mềm chia sẻ

Bạn có thể biên dịch một gói phần mềm thư viện Debian cho phép hỗ trợ multiarch dh(1) như sau:

- Cập nhật `debian/control`.
  - Thêm `Build-Depends: debhelper (>=10)` cho phần gói phần mềm nguồn.
  - Thêm `Pre-Depends: ${misc:Pre-Depends}` cho mỗi gói phần mềm nhị phân thư viện chia sẻ.

[13] Các đường dẫn thư viện chuyên dụng cũ như `/lib32/` và `/lib64/` không được sử dụng nữa.

  - Thêm `Multi-Arch:` cho mỗi phần tương ứng với mỗi gói phần mềm nhị phân.

- Đặt giá trị 10 cho `debian/compat`.
- Điều chỉnh đường dẫn từ `/usr/lib/` bình thường sang multiarch `/usr/lib/$(DEB_HOST_MULTIARCH)/` cho tất cả kịch bản đóng gói.
  - Gọi `DEB_HOST_MULTIARCH ?= $(shell dpkg-architecture -qDEB_HOST_MULTIARCH)` trong `debian/rules` để đặt giá trị cho tham biến `DEB_HOST_MULTIARCH` trước tiên.
  - Thay thế `/usr/lib/` với `/usr/lib/$(DEB_HOST_MULTIARCH)/` trong `debian/rules`.
  - Nếu `./configure` được sử dụng trong một phần của mục tiêu `override_dh_auto_configure` trong `debian/rules`, chắc chắn rằng bạn thay thế nó với `dh_auto_configure -- .` [14]
  - Thay thế hết tất cả các sự xuất hiện của `/usr/lib/` với `/usr/lib/*/` trong các tập tin `debian/`*`foo`*`.install`.
  - Tạo ra các tập tin như `debian/`*`foo`*`.links` từ `debian/`*`foo`*`.links.in` một cách tự động bằng thêm một kịch bản vào mục tiêu `override_dh_auto_configure` trong `debian/rules`.

```
override_dh_auto_configure:
        dh_auto_configure
        sed 's/@DEB_HOST_MULTIARCH@/$(DEB_HOST_MULTIARCH)/g' \
                debian/foo.links.in > debian/foo.links
```

Hãy chắc chắn rằng gói phần mềm thư viện chia sẻ chỉ chứa duy nhất các tập tin được trông đợi, và gói phần mềm -dev của bạn vẫn hoạt động tốt.

Tất cả các tập tin được cài đặt cùng lúc với gói phần mềm multiarch vào cùng một đường dẫn nên có cùng một nội dung chính xác như nhau. Bạn phải cẩn thận với những sự khác biệt tạo ra bởi thứ tự byte dữ liệu và thuật toán nén.

A.5 Gói phần mềm Debian native

Nếu một gói phần mề được bảo trì chỉ cho Debian hoặc có thể chỉ cho việc dùng nội bộ, mã nguồn của nó có thể chứa tất các tập tin `debian/*` bên trong nó. Có 2 cách để đóng gói nó.

Bạn có thể tạo một tập tin lưu trữ thượng nguồn bằng việc loại trừ tất cả các tập tin `debian/*` và đóng gói nó như là một gói phần mềm nhị phân Debian không native như ở Phần 2.1. Đây là cách thông thường mọi người khuyến khích sử dụng.

Một cách khác là cách làm của một gói phần mềm Debian native.

- Tạo một gói phần mềm nguồn Debian native dưới định dạng `3.0 (native)` bằng việc sử dụng một tập tin nén tar đơn độc chứa tất cả các tập tin bên trong.
  - *`package_version`*`.tar.gz`
  - *`package_version`*`.dsc`
- Biên dịch các gói phần mềm nhị phân Debian từ gói phần mềm nguồn Debian native.
  - *`package_version_arch`*`.deb`

Ví dụ, nếu bạn có các tập tin nguồn trong thư mục `~/mypackage-1.0` mà không có các tập tin `debian/*`, bạn có thể tạo một gói phần mềm Debian native cho nó bằng việc chạy lệnh **dh_make** như sau:

[14] Ngoài ra, bạn có thể thêm các đối số `--libdir=\$${prefix}/lib/$(DEB_HOST_MULTIARCH)` và `--libexecdir=\$${prefix}/lib/$(DEB_HOST_MULTIARCH)` cho `./configure`. Xin hãy chú ý rằng tham số `--libexecdir` thông báo đường dẫn ngầm định để cài đặt các chương trình thực thi được chạy bởi các chương trình khác thay vì bởi người dùng. Giá trị ngầm định Autotools của nó là `/usr/libexec/` nhưng ngầm định của Debian là `/usr/lib/`.

```
$ cd ~/mypackage-1.0
$ dh_make --native
```

Sau đó thư mục `debian` và nội dụng của nó được tạo như là ở Phần 2.8. Hành động này không tạo một tập tin lưu trữ vì đây là một gói phần mềm Debian native. Nhưng đó là điểm khác biệt duy nhất. Phần còn lại của quá trình đóng gói thực tế là không khác gì.

Sau khi chạy lệnh **dpkg-buildpackage**, bạn sẽ thấy các tập tin sau trong thư mục cha:

- `mypackage_1.0.tar.gz`

  Đây là tập tin mã nguồn tạo ra từ thư mục `mypackage-1.0` bởi lệnh **dpkg-source** command. (Phần hậu tố của nó không phải là `orig.tar.gz`.)

- `mypackage_1.0.dsc`

  Đây là bản tóm tắt của những nội dung của mã nguồn như ở trong gói phần mềm Debian không native. (There có mã tu chỉnh Debian.)

- `mypackage_1.0_i386.deb`

  Đây là gói phần mềm nhị phân hoàn chỉnh của bạn như là gói phần mềm Debian không native. (Không có mã tu chỉnh Debian.)

- `mypackage_1.0_i386.changes`

  Tập tin này mô tả những thay đổi được tạo ra trong phiên bản gói phần mềm hiện tại như ở trong gói phần mềm Debian không native. (Không có bản tu chỉnh Debian.)