

Podręcznik dla nowych opiekunów pakietów Debiana

Josip Rodin <joy-mg@debian.org>

polskie tłumaczenie: Paweł Tęcza <ptecza@debianusers.pl>

korekta tłumaczenia: Marcin Owsiany <porridge@debian.org>

wznowienie tłumaczenia: Wojciech Zaręba <wojtekz@comp.waw.pl>

wersja oryginału: 1.2.11, 12-01-2007, wersja tłumaczenia: 1.2.5, 27-09-2007

Prawa autorskie

Copyright © 1998-2002 Josip Rodin.

Copyright © 2005-2007 Osamu Aoki.

Copyright © polskiego tłumaczenia 2002-2004 Paweł Tęcza, Marcin Owsiany.

Copyright © polskiego tłumaczenia 2007 Wojciech Zaręba.

Ten dokument może być używany zgodnie z zasadami licencji GNU GPL (General Public License) w wersji 2 lub wyższej.

Do stworzenia tego dokumentu wykorzystano, jako przykłady, następujące dokumenty:

Making a Debian Package (znany jako Debmake Manual), copyright © 1997 Jaldhar Vyas.

The New-Maintainer's Debian Packaging Howto, copyright © 1997 Will Lowe.

Spis treści

1	Rozpoczęcie, jak się należy	1
1.1	Programy, których potrzebujesz do dewelopmentu	2
1.2	Oficjalny Deweloper Debiana	4
1.3	Inne informacje	4
2	Pierwsze kroki	7
2.1	Wybierz swój program	7
2.2	Weź program i wypróbuj go	8
2.3	Nazwa pakietu i jego wersja	9
2.4	Wstępna „debianizacja”	10
3	Modyfikacja źródła	11
3.1	Instalacja w podkatalogu	11
3.2	Nie zgadzające się biblioteki	14
4	Rzeczy wymagane w katalogu debian/	15
4.1	Plik ‘control’	15
4.2	Plik ‘copyright’	20
4.3	Plik ‘changelog’	21
4.4	Plik ‘rules’	22
5	Inne pliki z katalogu debian/	27
5.1	Plik ‘README.Debian’	27
5.2	Plik ‘conffiles.ex’	28
5.3	Plik ‘cron.d.ex’	28

5.4	Plik 'dirs'	28
5.5	Plik 'docs'	29
5.6	Plik 'emacsen-*.ex'	30
5.7	Plik 'init.d.ex'	30
5.8	Pliki 'manpage.1.ex', 'manpage.sgml.ex'	30
5.9	Plik 'menu.ex'	31
5.10	Plik 'watch.ex'	32
5.11	Plik 'ex.package.doc-base'	32
5.12	postinst.ex, preinst.ex, postrm.ex, prerm.ex	33
6	Budowanie pakietu	35
6.1	Całkowita przebudowa	35
6.2	Szybka przebudowa	36
6.3	Polecenie <code>debuild</code>	37
6.4	Systemy <code>dpatch</code> i <code>quilt</code>	37
6.5	Dołączanie <code>orig.tar.gz</code> podczas wgrywania	38
7	Sprawdzanie pakietu pod kątem błędów	39
7.1	Pakiety <code>lintian</code>	39
7.2	Polecenie <code>mc</code>	39
7.3	Polecenie <code>debdiff</code>	40
7.4	Polecenie <code>interdiff</code>	40
7.5	Polecenie <code>debi</code>	40
7.6	Pakiet <code>pbuilder</code>	40
8	Wgrywanie pakietu	43
8.1	Wgrywanie pakietu do archiwum Debiana	43
8.2	Wgrywanie do prywatnego archiwum	45
9	Aktualizacja pakietu	47
9.1	Nowa poprawka Debiana	47
9.2	Nowe wydanie autorskie (prosto)	47
9.3	Nowe wydanie autorskie (realistycznie)	48

9.4	Plik <code>orig.tar.gz</code>	50
9.5	Polecenie <code>cvs-buildpackage</code> i jemu podobne	50
9.6	Weryfikowanie uaktualnienia pakietu do nowszej wersji	50
10	Gdzie prosić o pomoc	53
A	Przykłady	55
A.1	Prosty przykład pakowania	55
A.2	Przykład z <code>dpatch</code> i <code>pbuilder</code>	55

Rozdział 1

Rozpoczęcie, jak się należy

Ten dokument próbuje opisać proces budowania pakietów systemu Debian. Jest on przeznaczony dla zwykłych użytkowników Debiana i tych, którzy chcą zostać deweloperami. Używa zwykłego języka i daje dużo działających przykładów. Stare rzymskie przysłowie mówi *Longum iter est per praecepta, breve et efficax per exempla* (Długa jest droga przez nakazy, krótka i skuteczna przez przykłady).

Jedną z rzeczy sprawiających, że Debian jest wyjątkową dystrybucją Linuksa, jest jego system pakietów. Mimo że istnieje ogromna ilość oprogramowania już spakowanego do formatu Debiana, to czasami zachodzi konieczność zainstalowania programu, który nie posiada swojej paczki. Pewnie się dziwisz, że możesz sam zbudować własne pakiety i myślisz, że to bardzo trudne zadanie. No cóż, jeśli jesteś zupełnym nowicjuszem w Linuksie, to rzeczywiście będziesz miał kłopoty, ale czy gdybyś był żółtodziobem, to czytałbyś teraz ten dokument? :-)

Musisz trochę wiedzieć na temat programowania pod Unix'em, ale nie musisz od razu być ekspertem.

Jedna rzecz jest pewna: aby odpowiednio tworzyć i zarządzać pakietami Debiana konieczne są osobogodziny. Staraj się nie popełniać błędów, gdyż nasz system, aby działać, wymaga od opiekunów zarówno kompetencji technicznych jak i rzetelności.

Ten dokument wyjaśni wszystkie kroki potrzebne do zbudowania pakietu (choć niektóre z nich mogą na początku wydać się nieistotne) i pomoże Ci stworzyć Twój pierwszy pakiet. Dzięki niemu nabierzesz trochę doświadczenia, które przyda Ci się w trakcie budowania następnych wydań pakietu, a później może również do tworzenia innych paczek.

Najnowsza wersja tego dokumentu powinna być zawsze dostępna bezpośrednio na stronie <http://www.debian.org/doc/maint-guide/> (<http://www.debian.org/doc/maint-guide/>) oraz w pakiecie 'maint-guide'. Polskie tłumaczenie jest również dostępne w pakiecie 'maint-guide-pl'.

1.1 Programy, których potrzebujesz do dewelopmentu

Zanim zaczniesz cokolwiek robić, powinieneś upewnić się, że masz zainstalowanych kilka dodatkowych pakietów niezbędnych do dewelopmentu. Zwróć uwagę, że na poniższej liście nie ma żadnych pakietów oznaczonych jako 'niezbędne' (essential) lub 'wymagane' (required). Po prostu zakładamy, że masz już je zainstalowane.

Ta wersja podręcznika została uaktualniona z myślą o pakietach wchodzących w skład Debiana 2.2 ('potato') oraz 3.0 ('woody').

Następujące pakiety wchodzą w skład standardowej instalacji Debiana, więc prawdopodobnie masz je (i dodatkowe pakiety, od których one zależą) już zainstalowane. Mimo to powinieneś sprawdzić ich status za pomocą polecenia 'dpkg -s <pakiet>'.
'

- `dpkg-dev` - pakiet zawierający narzędzia niezbędne do rozpakowywania, budowania i wysyłania pakietów źródłowych Debiana (więcej informacji znajdziesz na stronie podręcznika `dpkg-source(1)`).
- `file` - przydatny program do określania typu pliku (zobacz `file(1)`).
- `gcc` - kompilator GNU języka C, niezbędny gdy Twój program, tak jak większość programów, został napisany w języku C (zobacz `gcc(1)`). Pakiet ten jest powiązany z kilkoma innymi pakietami, takimi jak `binutils`, który zawiera zestaw programów służących do asemlacji i konsolidacji plików wynikowych (zobacz 'info binutils' w pakiecie `binutils-doc`) i `cpp`, zawierający preprocesor języka C (zobacz `cpp(1)`).
- `g++` - kompilator GNU języka C++, niezbędny gdy Twój program został napisany w języku C++ (zobacz `g++(1)`).
- `libc6-dev` - biblioteki języka C i pliki nagłówkowe kompilatora gcc niezbędne do konsolidacji plików wynikowych (zobacz 'info libc' w pakiecie `glibc-doc`).
- `make` - na ogół proces tworzenia programu składa się z szeregu kroków. Zamiast ciągłego powtarzania w kółko tych samych komend, możesz posłużyć się programem, który automatyzuje cały proces. Jedyne co musisz zrobić, to stworzyć plik(i) 'Makefile' (zobacz 'info make').
- `patch` - bardzo użyteczne narzędzie służące do tworzenia łat, czyli utworzonych przez program diff plików z różnicami pomiędzy plikami źródłowymi, i tworzenia w ten sposób „załatanych” (poprawionych) wersji programów (zobacz `patch(1)`).
- `perl` - Perl jest jednym z najczęściej stosowanych interpretowanych języków skryptowych w systemach kompatybilnych z systemem Unix. Często określany jest jako „Uniksowy scyzoryk z piłą łańcuchową” (po angielsku znacznie zabawniej: Unix's Swiss Army Chainsaw) - zobacz `perl(1)`.

Najprawdopodobniej przydadzą się również następujące pakiety:

- `autoconf` i `automake` - wiele nowszych programów używa skryptów konfiguracyjnych i plików Makefile przetworzonych za pomocą takich narzędzi (zobacz 'info autoconf' i 'info automake').
- `dh-make` i `debhelper` - pakiet `dh-make` jest niezbędny do stworzenia szablonu naszego przykładowego pakietu. Używa on do tworzenia pakietów niektórych narzędzi z pakietu `debhelper`. Pakiety te nie są niezbędne do budowania paczek, ale są **bardzo** zalecane, szczególnie nowym opiekunom. Dzięki nim o wiele łatwiej rozpocząć proces budowania pakietu i kontrolować go później (zobacz `dh-make(1)`, `debhelper(1)` i plik `/usr/share/doc/debhelper/README`).
- `devscripts` - pakiet zawierający parę użytecznych i pomocnych dla opiekuna skryptów, które nie są jednakże niezbędne do budowania pakietów (więcej informacji znajdziesz w pliku `/usr/share/doc/devscripts/README.gz`).
- `fakeroot` - narzędzie, które pozwala „udawać” bycie administratorem systemu (rotem). Uprawnienia administratora są niezbędne w niektórych etapach procesu budowania pakietu (zobacz `fakeroot(1)`).
- `gnupg` - narzędzie umożliwiające cyfrowe *podpisanie* pakietów. Jest to szczególnie ważne, gdy zamierzasz rozpowszechnić swój pakiet, a na pewno będziesz musiał to zrobić, gdyby Twój pakiet miał być włączony do dystrybucji Debiana (zobacz `gpg(1)`).
- `g77` - kompilator GNU języka Fortran 77, niezbędny gdy Twój program został napisany w języku Fortran (zobacz `g77(1)`).
- `gpc` - kompilator GNU języka Pascal, niezbędny gdy Twój program został napisany w języku Pascal. Warty odnotowania w tym miejscu jest również pakiet `fp-compiler` (Free Pascal Compiler), który także nadaje się do tego celu (zobacz `gpc(1)` i `ppc386(1)`).
- `xutils` - niektóre programy, głównie dla X11, używają tych narzędzi do wygenerowania plików Makefile z zestawu makro-funkcji (zobacz `imake(1)` i `xmkmf(1)`).
- `lintian` - program służący do sprawdzania poprawności pakietów Debiana. Poinformuje Cię, gdy w zbudowanej paczce znajdzie błędy i wyjaśni ich przyczynę (zobacz `lintian(1)` oraz `/usr/share/doc/lintian/lintian.html/index.html`).
- `pbuilder` - ten pakiet zawiera programy, które są używane do tworzenia i zarządzania środowiskiem chroot. Budowanie pakietów Debiana w tym środowisku weryfikuje poprawność zależności i zapobiega powstawaniu błędów FTBFS (zobacz `pbuilder(8)` i `pdebuild(1)`).

Poniżej wymieniamy *bardzo ważną* dokumentację, która powinna być przeczytana razem z tym podręcznikiem:

- `debian-policy` - Polityka Debiana opisuje strukturę i zawartość archiwum Debiana, sprawy dotyczące sposobu projektowania systemu operacyjnego, FHS (Standard Hierarchii Systemu Plików - Filesystem Hierarchy Standard), który mówi, gdzie po-

winy się znajdować pliki i katalogi itd. Dla Ciebie najważniejszy jest opis wymagań, które musi spełnić każdy pakiet, aby mógł być włączony do dystrybucji (zobacz </usr/share/doc/debian-policy/policy.html/index.html>).

- `developers-reference` - opisuje wszystkie zagadnienia nie związane z technicznymi szczegółami procesu tworzenia pakietów, a więc strukturę archiwum, sposób zmian nazw pakietów, procedurę ich osierocania i adopcji, umieszczania pakietów w archiwum Debiana, również nie będąc opiekunem danego pakietu (Non-Maintainer Upload - NMU), jak zarządzać błędami, najlepsze praktyki dotyczące pakowania, kiedy i gdzie umieszczać pakiet itd. (zobacz </usr/share/doc/developers-reference/index.en.html>).

Powyższe krótkie opisy służą jedynie jako wprowadzenie do opisu każdego z pakietów. Zanim przejdiesz dalej, prosimy gruntownie zapoznać się z dokumentacją do każdego z programów, a przynajmniej z ich standardowym użyciem. Być może wydaje Ci się to teraz trudne, ale później będziesz *bardzo* zadowolony z przeczytania tej dokumentacji.

Uwaga: pakiet `debmake` zawiera niektóre programy zbliżone funkcjonalnie do `dh-make`, ale ten dokument **nie** omawia jego użycia, ponieważ jest on *przestarzały*.

1.2 Oficjalny Deweloper Debiana

Po zbudowaniu swojego pierwszego pakietu (albo w czasie budowania) być może zechcesz zostać oficjalnym Deweloperem Debiana, aby wprowadzić swój pakiet do kolejnej dystrybucji (jeśli program jest użyteczny, to czemu nie?).

Nie możesz zostać oficjalnym Deweloperem Debiana w ciągu jednej nocy, gdyż do tego potrzebne jest coś więcej, niż umiejętności techniczne. Niech Cię to jednak nie zniechęca. Możesz wysłać swój pakiet, jeśli jest użyteczny dla innych, jako opiekun sponsorowany przez zapisanie się na stronie Nowych Opiekunów (<http://nm.debian.org/>). Sponsor jest oficjalnym Deweloperem Debiana, który pomaga opiekunom włączać pakiety do archiwum Debiana. Więcej o procedurze przyjmowania nowych opiekunów jest na stronie `debian-mentors FAQ` (http://people.debian.org/~mpalmer/debian-mentors_FAQ.html).

Zwracamy uwagę, że nie ma potrzeby tworzenia nowego pakietu, aby stać się oficjalnym Deweloperem Debiana. Rozwój istniejących pakietów jest również drogą mogącą prowadzić do zostania oficjalnym Deweloperem.

1.3 Inne informacje

Istnieją dwa rodzaje pakietów, jakie możesz stworzyć: źródłowe i binarne. Pakiet źródłowy zawiera kod, który możesz skompilować, aby otrzymać binarną postać programu. Pakiet binarny zawiera natomiast już gotowy do użycia program. Prosimy nie mylić takich pojęć, jak źródło programu i pakiet źródłowy! Więcej szczegółów na temat terminologii jest w innych podręcznikach.

W Debianie termin 'opiekun' (maintainer) oznacza osobę, która tworzy pakiety (pakuje programy), 'autor' (upstream author) - osobę, która tworzy program, a 'zewnętrzny opiekun' (upstream maintainer) - osobę, która aktualnie opiekuje się programem, pozostając poza projektem Debian. Zwykle autor i zewnętrzny opiekun są tą samą osobą - czasem nawet tą samą osobą jest opiekun. Jeśli napisałeś jakiś program i chcesz, żeby wszedł w skład Debiana, przyslij swoje zgłoszenie i zostań opiekunem.

Rozdział 2

Pierwsze kroki

2.1 Wybierz swój program

Prawdopodobnie wybrałeś już pakiet, który chcesz zbudować. Pierwszą rzeczą, którą powinieneś zrobić, to sprawdzić, czy pakiet znajduje się już w dystrybucji, przy pomocy programu `aptitude`. Jeśli używasz dystrybucji 'stabilnej', zapewne najlepiej przejść do strony wyszukiwania pakietów (<http://www.debian.org/distrib/packages>).

Jeśli pakiet już istnieje, to cóż, zainstaluj go! :-). Jeśli przypadkiem jest on osierocony (jego opiekunem jest „Debian QA Group”), to być może możesz się nim zaopiekować.

Sprawdź na stronie Pakietów Rokujących i Wymagających Pracy (<http://www.debian.org/devel/wnpp/>) i stronach powiązanych ostatni status adopcji/osierocenia pakietu.

Jeśli możesz zaadoptować pakiet, pobierz jego źródła (poleceniem `apt-get source nazwa_pakietu`) i przetestuj go. Niestety ten dokument nie zawiera informacji na temat adopcji pakietów. Za to nie musisz męczyć się, rozpracowując działanie danego pakietu, gdyż ktoś już wcześniej dokonał wstępnych ustawień. Ale czytaj dalej, bo poniższe porady będą z pewnością wartościowe również dla Ciebie.

Jeśli pakiet jest nowy i chciałbyś, żeby został włączony do dystrybucji Debiana, wykonaj poniższe instrukcje:

- sprawdź na stronie lista pakietów w opracowaniu (http://www.de.debian.org/devel/wnpp/being_packaged), czy ktoś już nie pracuje nad tym pakietem. Jeśli ktoś już to robi, to możesz się z nim skontaktować, jeśli sądzisz, że mógłbyś mu pomóc. Jeśli nie - znajdź jakiś inny interesujący Cię program, który nie ma jeszcze swojego opiekuna.
- program **musi** mieć licencję i, jeśli to możliwe, najlepiej zgodną z Wytycznymi Debiana dotyczącymi Wolnego Oprogramowania (http://www.debian.org/social_contract.html#guidelines) oraz **nie może** wymagać pakietów spoza sekcji `main` do kompilacji lub wykonania, bo jest to niezgodne z Polityką Debiana. Jeśli nie zgadza się to z którąś z powyższych zasad, program może być włączony do sekcji 'contrib' lub 'non-free', zależnie od sytuacji. Gdy nie jesteś pewny, do której sekcji można włączyć

program, wyślij tekst licencji na listę <debian-legal@lists.debian.org> i poproś o poradę.

- program z pewnością **nie** powinien być uruchamiany z ustanowionym identyfikatorem administratora systemu (setuid root), a jeszcze lepiej - nie powinien potrzebować ustanowionego żadnego identyfikatora użytkownika lub grupy.
- program nie powinien być demonem ani innym programem umieszczanym w katalogu */sbin, ani nie powinien otwierać portu jako administrator systemu.
- program powinien mieć binarną, wykonywalną formę, biblioteki są trudniejsze w utrzymaniu.
- program powinien być dobrze udokumentowany, a kod zrozumiały (np. nie zagmatwany).
- powinieneś skontaktować się z autorem(ami) programu, aby sprawdzić czy zgadzają się na jego zapakowanie. Ważną rzeczą jest możliwość konsultacji z autorem w razie wystąpienia jakichś specyficznych problemów. Nie próbuj pakować oprogramowania, którym się nikt nie zajmuje.
- i na końcu, choć wcale nie jest to najmniej ważne, musisz wiedzieć jak program działa i wypróbować go przez pewien czas.

Oczywiście powyższe zalecenia to po prostu zabezpieczenia, które mają na celu uchronić Cię przed gniewem użytkowników, gdy zrobisz coś źle w jakimś demonie z ustanowionym identyfikatorem użytkownika... Gdy nabierzesz już więcej doświadczenia, będziesz mógł pakować nawet takie programy, ale nawet najbardziej doświadczeni deweloperzy konsultują się na liście dyskusyjnej Mentorów Debiana, gdy mają jakieś wątpliwości. Ludzie stamtąd z pewnością chętnie pomogą.

Więcej informacji na te tematy znajdziesz w dokumencie Developer's Reference.

2.2 Weź program i wypróbuj go

Pierwszą rzeczą, którą powinieneś zrobić, to odnalezienie i pobranie oryginalnego pakietu. Zakładam, że już masz plik źródłowy, który pobrałeś ze strony domowej jego autora. Źródła z wolnym oprogramowaniem dla Uniksa są zwykle rozprowadzane w formacie tar/gzip, z rozszerzeniem .tar.gz. Pliki te na ogół zawierają podkatalog o nazwie program-wersja, w którym znajdują się wszystkie pliki źródłowe. Jeśli źródła wybranego przez Ciebie programu są rozprowadzane w innego rodzaju archiwum (na przykład pliki kończące się na „.Z” lub „.zip”), to wypakuj je przy pomocy odpowiedniego narzędzia. Gdy nie jesteś pewien, jak zrobić to poprawnie, zapytaj (po angielsku) na liście dyskusyjnej Mentorów Debiana (wskazówka: temat 'file archive.extension').

Jako przykładu będziemy używać programu o nazwie 'gentoo' - menadżera plików dla systemu X Windows, który wykorzystuje bibliotekę GTK+. Zwróć uwagę, że program ten jest już zapakowany i znacznie się zmienił od czasu, gdy pisany był ten tekst.

W swoim katalogu domowym utwórz podkatalog o nazwie 'debian', 'deb' lub jakkolwiek uważasz za właściwe (np. po prostu ~/gentoo/ jest w tym przypadku dobrym rozwiązaniem). Umieść w nim pobrane archiwum i rozpakuj je (za pomocą 'tar xzf gentoo-0.9.12.tar.gz'). Upewnij się, że nie ma żadnych błędów, nawet jakichś nieistotnych, ponieważ najprawdopodobniej pojawią się problemy w czasie rozpakowywania w systemach innych użytkowników, których narzędzia do wypakowywania mogą, ale nie muszą, ignorować takie nieprawidłowości.

Teraz w katalogu tym powinieneś mieć podkatalog o nazwie 'gentoo-0.9.12'. Wejdź do niego i **dokładnie** przeczytaj znajdującą się tam dokumentację. Zwykle powinny tam być pliki o nazwach README*, INSTALL*, *.lsm lub *.html. Odszukaj w dokumentacji instrukcji, jak poprawnie skompilować i zainstalować program (najprawdopodobniej będą one zakładać, że chcesz zainstalować program do katalogu /usr/local/bin; ale Ty nie rób tak, więcej o tym w rozdziale 'Instalacja w podkatalogu' na 11 stronie).

Proces instalacji różni się w zależności od programu, ale wiele nowoczesnych programów jest dostarczanych ze skrypcem 'configure', który konfiguruje źródła pod Twoim systemem i sprawdza, czy spełniają one warunki niezbędne do poprawnej kompilacji. Po zakończeniu konfiguracji wykonywanej za pomocą polecenia './configure', programy są na ogół kompilowane przy użyciu komendy 'make'. Niektóre z nich pozwalają także na użycie 'make check', który uruchamia procedurę samosprawdzającą. Instalacja w katalogu przeznaczenia następuje zwykle po wydaniu polecenia 'make install'.

Teraz spróbuj skompilować i uruchomić program, aby upewnić się czy działa on prawidłowo i nic się nie psuje w czasie instalacji lub wykonywania.

Możesz też zazwyczaj użyć polecenia 'make clean' (lub lepiej 'make distclean'), aby posprzątać w katalogu, w którym kompilowałeś. Czasem można nawet posłużyć się poleceniem 'make uninstall', które usunie wszystkie zainstalowane pliki.

2.3 Nazwa pakietu i jego wersja

Powinieneś rozpocząć pakowanie z zupełnie wyczyszczonym (ang. pristine - pierwotnym) katalogiem źródłowym, ewentualnie ze świeżo rozpakowanymi źródłami.

Aby prawidłowo zbudować pakiet, musisz tak zmienić nazwę oryginalnego programu, żeby występowały w niej tylko małe litery (o ile występują tam jakieś wielkie litery). Powinieneś także przenieść katalog ze źródłami do katalogu <nazwa_pakietu>-<wersja>.

Jeśli nazwa programu składa się z więcej niż jednego wyrazu, to skróć ją do jednego wyrazu albo utwórz skrót. Na przykład program „John's little editor for X” powinien się nazywać johnledx, jle4x lub jakoś tak. Pamiętaj jednak, aby jego długość nie przekraczała jakiegś rozsądnej wartości, np. 20 znaków.

Sprawdź także dokładnie wersję programu (będzie ona włączona do nazwy pakietu). Jeśli do oznaczenia jego wersji autor nie użył konwencji X.Y.Z, ale posłużył się datą, to Ty również możesz jej użyć do określenia wersji pakietu, poprzedzając ją ciągiem „0.0.” (na wypadek gdyby autor programu zdecydował się kiedyś wydać wersję o miło brzmiącym numerze 1.0). Zatem,

jeśli datą wydania danej wersji programu było 19 grudnia 1998 r., to powinieneś oznaczyć jego wersję jako 0.0.19981219.

Niektóre programy nie są jednak w żaden sposób numerowane. W takich przypadkach powinieneś skontaktować się z zewnętrznym opiekunem, aby dowiedzieć się, czy używa on jakiejś innej metody do oznaczania kolejnych poprawek programu.

2.4 Wstępna „debianizacja”

Upewnij się, że jesteś w katalogu ze źródłami programu i wydaj następujące polecenie:

```
dh_make -e twój.adres@opiekuna.pl -f ../gentoo-0.9.12.tar.gz
```

Oczywiście musisz zastąpić ciąg „twój.adres@opiekuna.pl” adresem Twojej skrzynki pocztowej, gdyż zostanie on wpisany do pliku ze zmianami (changelog) i innych plików, oraz zastąpić nazwę pliku nazwą Twojego archiwum źródłowego. Więcej szczegółów znajdziesz na stronie podręcznika `dh_make(1)`.

Trzeba też wprowadzić kilka dodatkowych informacji. Zostaniesz poproszony o podanie typu pakietu, który tworzysz. Gentoo to pojedynczy pakiet binarny - tworzy on tylko plik binarny, a zatem jeden plik `.deb`. W takim wypadku zaznaczamy pierwszą opcję za pomocą klawisza ‘s’. Następnie sprawdzamy informacje na ekranie i, jeśli wszystko się zgadza, potwierdzamy je naciskając <enter>.

Po wykonaniu `dh_make` jest tworzona kopia autorskiego archiwum jako `gentoo_0.9.12.orig.tar.gz`, w katalogu nadrzędnym, w celu dostosowania, jako niedebianowego pakietu źródłowego, do `diff.gz`. Nazwa tego pliku ma 2 zasadnicze właściwości:

- Nazwa pakietu i wersja są rozdzielone przez „_”.
- Został dodany sufiks „orig.” przed „tar.gz”.

Powtórzymy: ponieważ jesteś jeszcze nowym opiekunem, odradzamy Ci tworzenie skomplikowanych pakietów, takich jak:

- wielokrotne pakiety binarne
- pakiety bibliotek
- z plików źródłowych innych niż `tar.gz` lub `tar.bz2`, czy też
- z archiwów, których zawartość nie może być dystrybuowana.

Prosimy zauważyć, że powinno się uruchomić program `dh_make` **tylko jeden raz**, gdyż nie zachowa się on poprawnie, gdy uruchomisz go ponownie w tym samym już „zdebianizowanym” katalogu. Oznacza to również, że będziesz musiał użyć innej metody, aby wprowadzić w przyszłości nową poprawkę lub nową wersję pakietu. Więcej informacji na ten temat znajdziesz dalej w rozdziale ‘Aktualizacja pakietu’ na 47 stronie.

Rozdział 3

Modyfikacja źródła

Normalnie programy instalują się w podkatalogach katalogu `/usr/local`. Pakiety Debiana nie mogą jednak używać tego katalogu, gdyż jest on zarezerwowany do prywatnego użycia przez administratora (lub użytkowników) systemu. Oznacza to, że musisz się przyjrzeć, jak budowany jest Twój program, zwykle za pomocą pliku `Makefile`. Jest to skrypt programu `make(1)` używanego do automatycznego budowania programu. Więcej szczegółów na temat plików `Makefile` znajdziesz w rozdziale ‘Plik ‘rules’” na 22 stronie.

Zwróć też uwagę na to, czy Twój program używa programów GNU `automake(1)` i/lub `autoconf(1)`, czyli czy źródła programu zawierają plik `Makefile.am` i/lub `Makefile.in`, to je będziesz musiał wtedy modyfikować. Dzieje się tak, ponieważ każde wywołanie programu `automake` powoduje ponowne utworzenie pliku `Makefile.in` z informacjami wygenerowanymi z pliku `Makefile.am`. Także każde wywołanie skryptu `./configure` robi to samo z plikiem `Makefile`, na podstawie pliku `Makefile.in`. Edycja plików `Makefile.am` wymaga pewnej wiedzy na temat programu `automake`, możesz o nim poczytać za pomocą komendy ‘`info automake`’. Edytowanie plików `Makefile.in` odbywa się niemal tak samo, jak w przypadku plików `Makefile`, po prostu trzeba zwracać uwagę na zmienne, tzn. wszystkie łańcuchy otoczone przez znaki ‘@’, dla przykładu zmienna `@CFLAGS@` lub `@LN_S@` będzie zastępowana odpowiednią wartością przy każdym wywołaniu skryptu `./configure`. Przeczytaj dokumentację `/usr/share/doc/autotools-dev/README.Debian.gz`, zanim zaczniesz modyfikacje.

Zauważ również, że nie ma tu wystarczającego miejsca, aby opisać *wszystkie* szczegóły na temat poprawiania zewnętrznych źródeł. Przedstawiam tu jedynie kilka problemów, z którymi często można się spotkać.

3.1 Instalacja w podkatalogu

Większość programów posiada swój własny sposób instalowania się w istniejącej strukturze katalogów systemu. Binaria trafiają do katalogów określonych zmienną środowiskową `$PATH`, zaś dokumentacja i strony podręcznika są umieszczane w zwykle stosowanych miejscach. Jednakże, jeśli pozwolisz na takie działanie, program może się zainstalować w każdym

miejscu systemu. Może sprawić to problemy narzędziom do obsługi pakietów, gdyż nie będą one wiedziały, które pliki należą do Twojej paczki, a które nie.

Zatem musisz zrobić coś innego: zainstalować swój program w tymczasowym katalogu, z którego narzędzia opiekuna zbudują działający pakiet `.deb`. Wszystko co zawiera ten katalog, zostanie zainstalowane w systemie użytkowników, gdy zdecydują się oni zainstalować Twój pakiet, z tą tylko różnicą, że program `dpkg` zainstaluje pliki względem katalogu głównego systemu.

Ten tymczasowy katalog jest zazwyczaj tworzony wewnątrz Twojego katalogu `debian/` w rozpakowanym drzewie ze źródłami. Na ogół ma on nazwę `debian/nazwa_pakietu`.

Pamiętaj, że nie wystarczy, żeby program zachowywał się poprawnie, gdy zostanie on zainstalowany się w katalogu `debian/nazwa_pakietu`. Musi on zachowywać się właściwie także, gdy zostanie umieszczony w głównym katalogu systemu. Zatem nie wolno pozwolić, żeby w plikach pakietu zostały „zaszyte” pełne ścieżki, np. `/home/ja/deb/gentoo-0.9.12/usr/share/gentoo`.

Sprawa jest prosta, gdy programy używają narzędzia GNU `autoconf`. Większość z nich posiada pliki `Makefile`, które domyślnie są ustawione w taki sposób, aby zezwalać na instalację w dowolnym podkatalogu zakładając, że katalog `/usr` (dla przykładu) jest kanonicznym prefiksem. Gdy zostanie wykryte, że Twój program używa `autoconfa`, to `dh_make` ustawi odpowiednie polecenia tak, żeby wszystko zostało zrobione automatycznie. W takich przypadkach możesz nawet ominąć dalsze czytanie tej sekcji. Jednak z innymi programami będziesz miał więcej pracy i prawdopodobnie będziesz musiał przejrzeć i wyedytować pliki `Makefile`.

Poniżej znajduje się odpowiednia część pliku `Makefile` programu `gentoo`:

```
# Katalog binariów dla 'make install'
BIN      = /usr/local/bin

# Katalog ikon dla 'make install'
ICONS    = /usr/local/share/gentoo
```

Widzimy, że pliki te zostaną zainstalowane w katalogu `/usr/local`. Zmieńmy zatem ścieżki na następujące:

```
# Katalog binariów dla 'make install'
BIN      = $(DESTDIR)/usr/bin

# Katalog ikon dla 'make install'
ICONS    = $(DESTDIR)/usr/share/gentoo
```

Ale dlaczego właśnie w tym katalogu, a nie w jakimś innym? Ponieważ pakiety Debiana nigdy nie instalują plików w katalogu `/usr/local`. Ten katalog jest zarezerwowany na potrzeby administratora systemu. W systemie Debian zaś takie pliki trafiają do katalogu `/usr`.

Dokładniejsze informacje na temat położenia binariów, ikon, dokumentacji itd. są opisane w dokumencie Filesystem Hierarchy Standard (zobacz w katalogu `/usr/share/doc/debian-policy/fhs/`). Polecamy Ci przejrzanie tego dokumentu i przeczytanie tych sekcji, które mogą dotyczyć Twojego pakietu.

Zatem powinniśmy instalować pliki binarne w katalogu `/usr/bin`, a nie w `/usr/local/bin`, strony podręcznika w katalogu `/usr/share/man/man1`, a nie w `/usr/local/man/man1` itd. Zauważ, że w pliku Makefile programu gentoo nie wspomniano o jego stronie podręcznika, ale ponieważ Polityka Debiana wymaga, żeby każdy program posiadał taką stronę, to stworzymy ją później i zainstalujemy w katalogu `/usr/share/man/man1`.

Niektóre programy nie używają zmiennych w plikach Makefile do definiowania ścieżek, takich jak te powyżej. Oznacza to, że będziesz musiał wyedytować niektóre źródła napisane w języku C i tak je poprawić, aby używały właściwych katalogów. Ale gdzie i czego właściwie szukać? Możesz to odnaleźć wydając polecenie:

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

Program `grep` przeszuka rekursywnie całe drzewo ze źródłami i wypisze nazwy plików i numery linii, gdy odnajdzie szukany wzorzec.

Wyedytuj te pliki i w odnalezionych liniach zastąp ciąg `/usr/local/*` ciągiem `usr/*` — i to już wszystko. Zrób to uważnie, aby nie zepsuć pozostałej części kodu! :-)

Następnie powinieneś odnaleźć w pliku Makefile cel `'install'` (szukaj linii rozpoczynającej się od ciągu `„install:”`) i zmienić wszystkie odniesienia do katalogów innych, niż te zdefiniowane na początku pliku Makefile. Przed zmianą, cel `'install'` programu gentoo był następujący:

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

Po wykonaniu zmian wygląda on następująco:

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

Na pewno zauważyłeś, że teraz przed pozostałymi poleceniami w tej regule będzie wykonywana komenda `install -d`. Oryginalny plik Makefile nie miał jej, gdyż zwykle katalog `/usr/local/bin` i inne katalogi istnieją już w systemie, zanim wyda się polecenie `'make install'`. Jednakże, gdy instalujemy program do własnego, pustego (lub nawet nie istniejącego) katalogu, to będziemy musieli utworzyć każdy z tych katalogów.

Możemy również dodać inne rzeczy na końcu tej reguły, na przykład instalację dodatkowej dokumentacji, którą autor programu czasem pomija:

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

Uważny czytelnik zauważy, że zmieniliśmy 'gentoo' na 'gentoo-target' w linii rozpoczynającej się od ciągu 'install:'. To jest tzw. poprawka nie dotycząca (unrelated bug fix) :-)

Ilekoć dokonasz zmian, które nie są ściśle związane z pakietem Debiana, poinformuj o nich opiekuna zewnętrznego, aby mógł on je włączyć do następnej wersji programu i by również ktoś inny mógł z nich skorzystać. Pamiętaj także, żeby tworzyć poprawki nie specyficzne dla Debiana, Linuksa (lub nawet Uniksa!) przed ich wysłaniem — niech będą one przenośne. Dzięki temu Twoje łaty będzie łatwiej nałożyć.

Zauważ, że nie musisz wysłać autorowi programu plików debian/*.

3.2 Nie zgadzające się biblioteki

Tutaj mamy pewien dość powszechny problem: biblioteki często różnią się pomiędzy platformami. Dla przykładu, plik Makefile może zawierać odwołania do biblioteki, której nie ma Debiana. W takim przypadku musimy zamienić ją na bibliotekę, która jest dostępna w Debianie i służy tym samym celom.

Jeśli zatem w pliku Makefile (lub Makefile.in) Twojego programu istnieje linia podobna do tej poniżej (i Twój program nie chce się skompilować):

```
LIBS = -lcurses -lcoś -lcoś_innego
```

to zmień ją w pokazany sposób i to powinno pomóc:

```
LIBS = -lncurses -lcoś -lcoś_innego
```

(Autor zdaje sobie sprawę, że to nie jest najlepszy przykład, zważywszy na to, że pakiet libncurses jest teraz dostarczany wraz z dowiązaniem symbolicznym do biblioteki libcurses.so, ale nie mógł wymyślić niczego lepszego. Sugestie będą bardzo mile widziane :-)

Rozdział 4

Rzeczy wymagane w katalogu debian/

W katalogu ze źródłami Twojego programu powstał nowy podkatalog, który nazywa się 'debian'. Zawiera on kilka plików, które powinniśmy wyedytować, aby umożliwić działanie pakietu. Najważniejszymi z nich są pliki 'control', 'changelog', 'copyright' i 'rules', które są wymagane w każdym pakiecie.

4.1 Plik 'control'

Ten plik zawiera różne informacje, których używają programy dpkg, dselect oraz inne narzędzia służące do zarządzania pakietem.

Poniżej przedstawiono plik control utworzony przez program dh_make:

```
1 Source: gentoo
2 Section: unknown
3 Priority: optional
4 Maintainer: Josip Rodin <joy-mg@debian.org>
5 Build-Depends: debhelper (>> 3.0.0)
6 Standards-Version: 3.6.2
7
8 Package: gentoo
9 Architecture: any
10 Depends: ${shlibs:Depends}
11 Description: <insert up to 60 chars description>
12 <insert long description, indented with spaces>
```

(dodałem numery linii)

Linie 1-6 zawierają informacje kontrolne dla pakietu źródłowego.

Linia 1. zawiera nazwę pakietu źródłowego.

Linia 2. oznacza sekcję dystrybucji, do której należy pakiet źródłowy.

Być może zauważyłeś, że Debian jest podzielony na następujące sekcje: 'main' (zawiera wolne oprogramowanie), 'non-free' (zawiera oprogramowanie, które nie jest wolne) i 'contrib' (zawiera wolne oprogramowanie, które zależy od oprogramowania, które nie jest wolne). Dodatkowo każda z sekcji dzieli się na logiczne podsekcje, które skrótowo opisują, do czego służy dany pakiet. Mamy zatem sekcję 'admin', która zawiera programy przeznaczone tylko dla administratora systemu, 'base' z podstawowymi narzędziami, 'devel' z narzędziami programistów, 'doc' z dokumentacją, 'libs' z bibliotekami, 'mail' z programami do obsługi poczty elektronicznej, 'net' z aplikacjami sieciowymi i demonami usług sieciowych, 'x11' z programami dla systemów X11, które nie pasują nigdzie indziej i wiele innych.

Zmierzymy ją zatem na x11. Prefiks "main/„ jest przyjmowany domyślnie, więc możemy go pominąć.

Linia 3. opisuje, jak ważne jest to, aby użytkownik zainstalował dany pakiet. Więcej informacji na temat wartości, jakie może przyjmować to pole znajdziesz w podręczniku Polityki Debiana. Dla nowych pakietów zazwyczaj może ono przyjmować wartość "optional„.

Sekcja (Section) i priorytet (Priority) są używane przez nakładki, jak program `dselect`, które używają ich do sortowania pakietów i wyboru domyślnego zestawu pakietów do zainstalowania. Gdy będziesz umieszczał swój pakiet w archiwum Debiana, wartość tych dwóch pól może być zmieniona przez opiekunów archiwum. W takich przypadkach zostaniesz o tym powiadomiony e-mailem.

Ponieważ jest to pakiet o normalnym priorytecie i nie jest w konflikcie z innym pakietem, to pozostawiamy tam wartość "optional„.

Linia 4. zawiera imię i nazwisko oraz adres e-mail opiekuna pakietu. Upewnij się, że pole to zawiera wartość odpowiednią dla nagłówka "To: „ wiadomości pocztowej, gdyż po umieszczeniu pakietu w archiwum system śledzenia błędów użyje tego pola do wysyłania Ci e-maili ze zgłoszeniami błędów. Nie stosuj przecinków, ampersandów ('&') i nawiasów.

Linia 5. zawiera listę pakietów wymaganych do zbudowania Twojego pakietu. Niektóre pakiety, na przykład `gcc` czy `make`, są założone z góry, więcej szczegółów na temat znajdziesz w pakiecie `build-essential`. Jeśli do zbudowania Twojego pakietu jest potrzebny jakiś niestandardowy kompilator lub inne narzędzie, to powinieneś dodać tutaj linię 'Build-Depends'. Wpisy są oddzielane od siebie za pomocą przecinków; przeczytaj objaśnienia na temat zależności binariów, aby dowiedzieć się więcej na temat składni tego pola.

Możesz także użyć w tym miejscu takich pól jak `Build-Depends-Indep`, `Build-Conflicts` i innych. Dane te są używane przez oprogramowanie do automatycznego budowania pakietów Debiana w celu stworzenia pakietów binarnych przeznaczonych dla innych platform komputerowych. Więcej informacji na temat zależności budowania pakietów znajdziesz w podręczniku Polityki. Dokument `Developers' Reference` zawiera szczegóły na temat innych platform (architektur) oraz adaptowania (ang. porting) do nich oprogramowania.

Poniżej pokazano sztuczkę, dzięki której odszukasz pakiety, których potrzebuje do zbudowania Twój pakiet:

```
strace -f -o /tmp/log ./configure
```

```
# or make instead of ./configure, if the package doesn't use autoconf
for x in `dpkg -S $(grep open /tmp/log|\
    perl -pe 's!.* open\(\\("[^\\"]*\).*!$1!' |\
    grep "^/" | sort | uniq|\
    grep -v "^\( /tmp\| /dev\| /proc\) " ) 2>/dev/null|\
    cut -f1 -d":" | sort | uniq`; \
do \
    echo -n "$x (>=" `dpkg -s $x|grep ^Version|cut -f2 -d":" ` ` " ), "; \
done
```

Aby ręcznie znaleźć kompletny zestaw zależności dla programu `/usr/bin/foo`, wykonaj

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

a dla każdej znalezionej biblioteki, np. `libfoo.so.6`, wykonaj

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

Potem tylko weź wersję `-dev` każdej z nich jako pozycję w `'Build-deps'`. Jeśli używasz do tego celu `ldd`, pokazuje on również zależności niebezpośrednie, co skutkuje zbyt dużą liczbą wykazywanych zależności.

Tak więc program `gentoo` wymaga do zbudowania pakietów `xlibs-dev`, `libgtk1.2-dev` i `libglib1.2-dev`, więc dodajmy je za pakietem `debhelper`.

Linia 6. jest wersją standardów polityki Debiana, którą dany pakiet spełnia, wersję podręcznika Polityki, który czytasz w trakcie tworzenia Twojego pakietu.

Linia 8. to nazwa pakietu binarnego. Zwykle jest ona taka sama jak nazwa pakietu źródłowego, ale nie musi to być regułą.

Linia 9. opisuje architekturę procesora, dla którego może być skompilowany pakiet. Pozostawimy w niej `"any"`, gdyż pakiet `dpkg-gencontrol(1)` sam wstawi w tym miejscu odpowiednią wartość dla każdego typu maszyny, na której kompilowany jest pakiet.

Jeśli Twój pakiet jest niezależny od architektury procesora (dla przykładu skrypt powłoki lub Perla, albo jakiś dokument), wpisz tutaj `"all"`, i poczytaj później w sekcji 'Plik `'rules'`' na 22 stronie na temat używania reguły `'binary-indep'` zamiast `'binary-arch'` do budowania pakietu.

Linia 10. pokazuje jedną z najpotężniejszych cech systemu pakietów Debiana. Pakiety mogą znajdować się w różnych relacjach z innymi pakietami. Oprócz pola `Depends:` mogą też występować pola opisujące inne związki: `Recommends:`, `Suggests:`, `Pre-Depends:`, `Conflicts:`, `Provides:` i `Replaces:`.

Narzędzia do zarządzania pakietami zwykle zachowują się w ten sam sposób w czasie ustalania relacji między pakietami. Jeśli tak nie jest, zostanie to wkrótce wyjaśnione (zobacz `dpkg(8)`, `dselect(8)`, `apt(8)`, `aptitude(1)`, itd.).

Pola te oznaczają:

- Depends: (Wymaga)

Pakiet nie zostanie zainstalowany o ile pakiety, których on wymaga nie są już zainstalowane w systemie. Użyj tego pola, gdy Twój program absolutnie nie może być uruchomiony (lub z dużymi trudnościami), jeśli któryś z tych pakietów nie jest obecny w systemie.

- Recommends: (Zaleca)

Nakładki takie jak `dselect` czy `aptitude` zachęcą Cię do zainstalowania zalecanych pakietów wraz z Twoim pakietem; `dselect` będzie nawet na to nalegać. Programy `dpkg` i `apt-get` jednak zignorują te pole. Użyj go dla pakietów, które nie są niezbędne, ale są zwykle używane razem z Twoim programem.

- Suggests: (Poleca)

Gdy użytkownik instaluje Twój program, wszystkie nakładki zachęcą go także do zainstalowania pakietów, które on poleca. Programy `dpkg` i `apt-get` nie będą się o to troszczyć. Użyj tego pola dla pakietów, które lepiej działają z Twoim programem, ale nie są dla niego niezbędne.

- Pre-Depends: (Przed-Wymaga)

Jest to silniejsza relacja niż `Depends`. Pakiet nie zostanie zainstalowany o ile pakiety, od których jest on przed-zależny nie są zainstalowane w systemie i *poprawnie skonfigurowane*. Używaj tego pola **bardzo** oszczędnie i jedynie po przedyskutowaniu tego na liście `debian-devel`. Czytaj: nie używaj go nigdy. :-)

- Conflicts: (PowodujeKonflikt)

Pakiet nie zostanie zainstalowany, dopóki wszystkie pakiety, które powodują konflikt nie zostaną wcześniej usunięte z systemu. Użyj tego pola, gdy Twój program absolutnie nie może być uruchomiony lub spowoduje jakieś problemy, jeśli jakiś inny pakiet jest obecny w systemie.

- Provides: (Dostarcza)

Dla niektórych rodzajów pakietów zostało zdefiniowanych wiele alternatywnych nazw wirtualnych. Pełną listę tych pakietów znajdziesz w pliku `/usr/share/doc/debian-policy/virtual-package-names-list.txt.gz`. Użyj tego pola, jeśli Twój program dostarcza funkcjonalności istniejącego już pakietu wirtualnego.

- Replaces: (Zastępuje)

Użyj tego pola, gdy Twój program zastępuje pliki jakiegoś innego pakietu lub zupełnie zastępuje jakiś pakiet (używane łącznie z polem `Conflicts`:). Pliki z wymienionych pakietów zostaną nadpisane przez pliki z Twojego pakietu.

Wszystkie te pola mają jednolitą składnię. Jest to lista nazw pakietów oddzielonych za pomocą przecinka. Nazwy pakietów mogą również być listami alternatywnych nazw pakietów oddzielonych przy pomocy symbolu `|` (symbol potoku).

Pola mogą ograniczać swoje zastosowanie tylko do szczególnych wersji każdego wymienionego pakietu. Wersje te są umieszczone w nawiasach po każdej nazwie pakietu i powinny zawierać relacje między numerami wersji pakietów. Dozwolonymi relacjami są: <<, <=, =, >= i >>, odpowiednio: wcześniejszy, wcześniejszy lub równy, dokładnie równy, późniejszy lub równy i późniejszy. Dla przykładu:

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

Ostatnią cechą, o której powinieneś wiedzieć, jest `$(shlibs:Depends)`. Gdy Twój pakiet zostanie zbudowany i zainstalowany w tymczasowym katalogu, program `dh_shlibdeps(1)` "prześwietli", go w poszukiwaniu binariów i bibliotek, określi jakich bibliotek współdzielonych wymaga i wykryje, w których pakietach się one znajdują, na przykład `libc6` lub `xlib6g`. Następnie program `dh_gencontrol(1)` umieści ich nazwy we właściwym miejscu, więc nie musisz się o to martwić.

Skoro już wszystko to zostało powiedziane, możemy pozostawić linię 10. w takiej postaci jak teraz i wstawić po niej `Suggests: file`, ponieważ `gentoo` może użyć niektórych funkcjonalności dostarczanych przez ten program/pakiet.

Linia 11. jest krótkim opisem pakietu. Większość ekranów tekstowych ma szerokość 80 kolumn, więc nie powinna ona zawierać więcej niż 60 znaków. Ja wpisałem w niej "fully GUI configurable X file manager using GTK+," (w pełni konfigurowalny okienkowy manager plików używający GTK+).

Od linii 12. zaczyna się dłuższy opis pakietu. Powinien to być akapit z większą liczbą szczegółów na temat pakietu. Pierwsza kolumna każdej linii długiego opisu powinna być pusta. Ponieważ opis ten nie może zawierać pustych linii, wszędzie tam gdzie chciałbyś je wstawić, musisz umieścić znak . (kropka) w kolumnie nr 2. Także na końcu długiego opisu nie może się pojawić więcej niż jedna pusta linia.

A oto końcowa postać uaktualnionego pliku 'control':

```
1 Source: gentoo
2 Section: x11
3 Priority: optional
4 Maintainer: Josip Rodin <joy-mg@debian.org>
5 Build-Depends: debhelper (>> 3.0.0), xlibs-dev, libgtk1.2-dev, libglib1.
6 Standards-Version: 3.5.2
7
8 Package: gentoo
9 Architecture: any
10 Depends: ${shlibs:Depends}
11 Suggests: file
```

```
12 Description: fully GUI configurable X file manager using GTK+
13 gentoo is a file manager for Linux written from scratch in pure C. It
14 uses the GTK+ toolkit for all of its interface needs. gentoo provides
15 100% GUI configurability; no need to edit config files by hand and re-
16 start the program. gentoo supports identifying the type of various
17 files (using extension, regular expressions, or the 'file' command),
18 and can display files of different types with different colors and icons
19 .
20 gentoo borrows some of its look and feel from the classic Amiga file
21 manager "Directory OPUS" (written by Jonathan Potter).
```

(numery linii zostały dodane przeze mnie)

4.2 Plik 'copyright'

Plik ten zawiera informacje o zewnętrznych (ang. upstream) zasobach pakietu, prawach autorskich i licencji. Jego format nie jest narzucony przez Politykę Debiana, ale jego zawartość już tak (zobacz sekcję 12.6 "Informacje o prawach autorskich,,).

Program dh_make stworzył już domyślny plik, którego zawartość jest podobna do tej poniżej:

```
1 This package was debianized by Josip Rodin <joy-mg@debian.org> on
2 Wed, 11 Nov 1998 21:02:14 +0100.
3
4 It was downloaded from <fill in ftp site>
5
6 Upstream Author(s): <put author(s) name and email here>
7
8 Copyright:
9
10 <Must follow here>
```

(numery linii zostały dodane przeze mnie)

Ważnymi rzeczami, które powinieneś dodać do tego pliku, jest miejsce, z którego pobrałeś pakiet ze źródłami oraz informacje o prawach autorskich i licencji. Musisz dołączyć kompletną treść licencji, chyba że jest to jedna z popularnych licencji wolnego oprogramowania, takich jak GNU GPL czy LGPL, BSD lub licencja Artystyczna. W takiej sytuacji możesz po prostu odesłać do odpowiedniego pliku w katalogu /usr/share/common-licenses/, który występuje w każdym systemie Debian.

Poniżej pokazano w skrócie, jak powinien wyglądać plik 'copyright' dla programu gentoo:

```
1 This package was debianized by Josip Rodin <joy-mg@debian.org> on
2 Wed, 11 Nov 1998 21:02:14 +0100.
```

```
3
4 It was downloaded from: ftp://ftp.obsession.se/gentoo/
5
6 Upstream author: Emil Brink <emil@obsession.se>
7
8 This software is copyright (c) 1998-99 by Emil Brink, Obsession
9 Development.
10
11 You are free to distribute this software under the terms of
12 the GNU General Public License either version 2 of the License,
13 or (at your option) any later version.
14 On Debian systems, the complete text of the GNU General Public
15 License can be found in the file '/usr/share/common-licenses/GPL-2'.
```

(numery linii zostały dodane przeze mnie)

Prosimy postępować zgodnie z plikiem HOWTO z listy debian-devel-announce: <http://lists.debian.org/debian-devel-announce/2006/03/msg00023.html>.

4.3 Plik 'changelog'

Jest plikiem wymaganym, którego format opisano w Polityce Debiana (sekcja 4.4 "debian/changelog"). Format ten jest wykorzystywany przez dpkg i inne programy do uzyskiwania informacji o numerze wersji, numerze rewizji (poprawki), dystrybucji i pilności Twojego pakietu.

Jest on także ważny dla Ciebie, ponieważ dobrze jest mieć udokumentowane wszystkie zmiany, których dokonałeś. Pomaga to ludziom pobierającym Twój pakiet zorientować się, czy nie zrobiłeś z pakietem czegoś, o czym powinni oni wiedzieć. Zmiany te zostaną zapisane do pliku '/usr/share/doc/gentoo/changelog.Debian.gz' w pakiecie binarnym.

Program dh_make również tworzy ten plik, którego zawartość wygląda mniej więcej tak:

```
1 gentoo (0.9.12-1) unstable; urgency=low
2
3 * Initial Release.
4
5 -- Josip Rodin <joy-mg@debian.org> Wed, 11 Nov 1998 21:02:14 +0100
6
```

(numery linii zostały dodane przeze mnie)

Linia 1. zawiera nazwę pakietu, wersję, dystrybucję i pilność. Nazwa musi się zgadzać z nazwą pakietu źródłowego, dystrybucja powinna mieć wartość albo 'unstable' (albo nawet 'experimental'), zaś pilności nie powinienes zmieniać na wartość większą niż 'low' (niska). :-)

Linie 3-5 to wpisy dziennika, w którym dokumentujesz zmiany dokonane w każdej z poprawek pakietu (ale nie zmiany zewnętrzne - do tego celu służy specjalny plik stworzony przez autorów programu, który później zainstalujesz jako `/usr/share/doc/gentoo/changelog.gz`). Nowe linie muszą być umieszczone przed znajdującą się na górze linią, która rozpoczyna się od gwiazdki (*). Możesz to zrobić przy pomocy `dch(1)` lub używając jakiegoś edytora tekstu.

Poprawiony będzie wyglądał jakoś tak:

```
1 gentoo (0.9.12-1) unstable; urgency=low
2
3  * Initial Release.
4  * This is my first Debian package.
5  * Adjusted the Makefile to fix $DESTDIR problems.
6
7 -- Josip Rodin <joy-mg@debian.org> Wed, 11 Nov 1998 21:02:14 +0100
8
```

(numery linii zostały dodane przeze mnie)

Więcej na temat pliku 'changelog' bedziesz mógł przeczytać dalej w rozdziale 'Aktualizacja pakietu' na [47](#) stronie.

4.4 Plik 'rules'

Teraz musimy się przyjrzeć regułom (ang. rules), których użyje program `dpkg-buildpackage(1)` do zbudowania naszego pakietu. Plik ten jest właściwie odmianą pliku Makefile, lecz różni się od tego/tych z programu źródłowego. Inaczej niż pozostałe pliki znajdujące się w katalogu `debian/`, ma on ustawiony atrybut wykonywalności.

Każdy plik 'rules', tak samo jak inne pliki Makefile, zawiera różne reguły, które określają, jak postępować ze źródłem. Każda reguła z kolei zawiera cele (targets), czyli nazwy plików bądź akcji, które powinny być stworzone lub wykonane (na przykład 'build:' lub 'install:'). Reguły, które chcesz wykonać są wywoływane z linii komend jako argumenty poleceń (dla przykładu `./debian/rules build` albo `make -f rules install`). Po nazwie celu możesz wymienić zależność, program lub plik, który od tej reguły zależy. W kolejnych liniach można wymienić dowolną liczbę komend, rozpoczynając je od znaku `<tab>`. Nowa reguła zaczyna się od deklaracji w pierwszej kolumnie. Puste linie i linie rozpoczynające się od znaku '#' (hash) są traktowane jako komentarz i ignorowane.

Pewnie jesteś teraz nieco zagubiony, ale wszystko stanie się jasne w czasie przeglądania pliku 'rules', który domyślnie jest tworzony przez program `dh_make`. Powinieneś też przeczytać o programie 'make' (poprzez 'info make'), aby uzyskać więcej informacji na jego temat.

Ważne jest, aby pamiętać, że plik 'rules' tworzony przez `dh_make` jest tylko propozycją. Działa on z prostymi pakietami, ale w przypadku bardziej skomplikowanych nie obawiaj się go modyfikować, zależnie od potrzeb. Jediną rzeczą, której nie możesz zmieniać to nazwy reguł, gdyż używają ich wszystkie narzędzia, zgodnie z wytycznymi zawartymi w Polityce Debiana.

Poniżej pokazano przykładowy domyślny plik `debian/rules`, który został wygenerowany przez program `dh_make`:

```
1  #!/usr/bin/make -f
2  # -*- makefile -*-
3  # Sample debian/rules that uses debhelper.
4  # This file was originally written by Joey Hess and Craig Small.
5  # As a special exception, when this file is copied by dh-make into a
6  # dh-make output file, you may use that output file without restriction.
7  # This special exception was added by Craig Small in version 0.37 of
8  # Uncomment this to turn on verbose mode.
9  #export DH_VERBOSE=1
10 configure: configure-stamp
11 configure-stamp:
12     dh_testdir
13     # Add here commands to configure the package.
14     touch configure-stamp
15 build: build-stamp
16 build-stamp: configure-stamp
17     dh_testdir
18     # Add here commands to compile the package.
19     $(MAKE)
20     #docbook-to-man debian/testpack.sgml > testpack.1
21     touch $@
22 clean:
23     dh_testdir
24     dh_testroot
25     rm -f build-stamp configure-stamp
26     # Add here commands to clean up after the build process.
27     $(MAKE) clean
28     dh_clean
29 install: build
30     dh_testdir
31     dh_testroot
32     dh_clean -k
33     dh_installdirs
34     # Add here commands to install the package into debian/testpa
35     $(MAKE) DESTDIR=$(CURDIR)/debian/testpack install
36 # Build architecture-independent files here.
37 binary-indep: build install
38 # We have nothing to do by default.
39 # Build architecture-dependent files here.
40 binary-arch: build install
41     dh_testdir
42     dh_testroot
43     dh_installchangelogs
```

```
44         dh_installdocs
45         dh_installexamples
46 #       dh_install
47 #       dh_installmenu
48 #       dh_installdebconf
49 #       dh_installogrotate
50 #       dh_installemacsen
51 #       dh_installpam
52 #       dh_installemime
53 #       dh_python
54 #       dh_installinit
55 #       dh_installcron
56 #       dh_installinfo
57         dh_installman
58         dh_link
59         dh_strip
60         dh_compress
61         dh_fixperms
62 #       dh_perl
63 #       dh_makeshlibs
64         dh_installdeb
65         dh_shlibdeps
66         dh_gencontrol
67         dh_md5sums
68         dh_builddeb
69 binary: binary-indep binary-arch
70 .PHONY: build clean binary-indep binary-arch binary install configure
```

(numery linii zostały dodane przeze mnie; w rzeczywistym pliku `debian/rules` wiodące białe znaki są tabulatorami)

Z liniami takimi jak linia nr 1 prawdopodobnie spotkałeś się już w skryptach powłoki albo Perla. Mówi ona systemowi operacyjnemu, że plik ten ma być przetwarzany przez program `"/usr/bin/make"`.

Znaczenie zmiennych `DH_*`, których użyto w liniach 8. i 9. powinno być zrozumiałe dzięki krótkiemu opisowi. Więcej informacji na temat zmiennej `DH_COMPAT` znajdziesz w sekcji "Debhelper compatibility levels," na stronie podręcznika programu `debhelper(1)`.

Linie 11-16 to szablon obsługujący parametry `DEB_BUILD_OPTIONS`, które opisano w Polityce Debiana (sekcja 10.1 "Binaries,,"). Po prostu mówią one, czy w binaria mają być wbudowane symbole służące do odpluskwiania (ang. *debugging*) i czy powinny one być usunięte przy instalacji. I znów: to jest tylko szablon, wskazówka, którą powinieś uwzględnić. Powinienieś sprawdzić, w jaki sposób autor programu obsługuje włączanie symboli odpluskwiających oraz usuwanie ich po instalacji i zaimplementować to samemu.

Zwykle możesz nakazać kompilatorowi `gcc` użycie opcji `"-g"`, przy pomocy zmiennej `CFLAGS`. Jeśli tak jest w przypadku Twojego pakietu, przekaz wartość tej zmiennej przez *dodanie* łańcu-

cha `CFLAGS="$ (CFLAGS) „` do wywołania `$(MAKE)` w regule `'build'` (zobacz poniżej). Jeśli zaś Twój pakiet używa skryptu konfiguracyjnego `autoconfa`, to możesz zmodyfikować konfigurację przez *poprzedzenie* powyższym łańcuchem wywołania skryptu `./configure` w regule `'build'`.

Jeśli chodzi o pozbywanie się symboli odpluskwiających, to programy są na ogół tak skonfigurowane, że instalują się z nimi i często nie mają opcji umożliwiającej zmianę tego stanu. Na szczęście mamy program `dh_strip(1)`, który wykryje, gdy ustawiona jest opcja `DEB_BUILD_OPTIONS=nostrip` i zakończy swe działanie.

Linie 18-26 opisują regułę `'build'` (i jej regułę potomną `'build-stamp'`), która uruchamia program `make` na oryginalnym pliku `Makefile` aplikacji, aby skompilować program. Jeśli pakiet używa narzędzi konfiguracyjnych GNU do zbudowania binariów, koniecznie przeczytaj `/usr/share/doc/autotools-dev/README.Debian.gz`. O zakomentowanym przykładzie `docbook-to-man` opowiemy dalej w rozdziale 'Pliki `'manpage.1.ex'`, `'manpage.sgml.ex'`' na 30 stronie.

Reguła `'clean'` zawarta w liniach 28-36 czyści wszystkie niepotrzebne pliki binarne i automatycznie wygenerowane rzeczy, które zostały po zbudowaniu pakietu. Reguła ta musi działać przez cały czas (nawet, gdy drzewo ze źródłami *jest wyczyszczone!*), zatem prosimy używać opcji wymuszającej (na przykład dla polecenia `rm` jest nią opcja `'-f'`) lub ignorującej zwracane wartości (błędy) poprzez zastosowanie `'-'` przed poleceniem.

Reguła `'install'`, która odpowiada za proces instalacji, rozpoczyna się w linii nr 38. Uruchamia ona po prostu regułę `'install'` z pliku `Makefile` programu i instaluje go w katalogu `$(CURDIR)/debian/gentoo` - oto dlaczego określiliśmy zmienną `$(DESTDIR)` jako katalog bazowy instalacji w pliku `Makefile` programu `gentoo`.

Jak tłumaczy komentarz, reguła `'binary-indep'`, która znajduje się w linii 48., jest używana do budowania pakietów niezależnych od architektury procesora. Jeśli nie mamy takiego pakietu, żadna akcja nie zostanie przedsięwzięta.

Następną regułą jest `'binary-arch'` znajdująca się w liniach 52-79. Uruchamia ona kilka małych programów narzędziowych z pakietu `debhelper`, które wykonują różne operacje z plikami pakietu, aby uczynić go zgodnym z Polityką Debiana.

Gdy określiłeś architekturę Twojego pakietu jako `'Architecture: all'`, będziesz musiał umieścić w tej regule wszystkie komendy do budowania pakietu i pozostawić pustą regułę `'binary-arch'`.

Nazwy programów wchodzących w skład pakietu `debhelper` rozpoczynają się od `dh_`. Reszta jest opisem tego, co dane narzędzie robi. Mimo, że dość dobrze same się one objaśniają, poniżej zamieszczono dodatkowe opisy:

- `dh_testdir(1)` sprawdza, czy jesteś we właściwym katalogu (tzn. na samej górze katalogu ze źródłami)
- `dh_testroot(1)` sprawdza, czy masz uprawnienia administratora systemu, których wymagają cele `'binary-arch'`, `'binary-indep'` i `'clean'`

- `dh_installman(1)` kopiuje strony podręcznika systemowego we właściwe miejsce w katalogu przeznaczenia. Musisz tylko powiedzieć, gdzie one się znajdują, względem głównego katalogu ze źródłami
- `dh_strip(1)` usuwa z plików wykonywalnych i bibliotek nagłówki służące do odpluskwania, aby uczynić je mniejszymi
- `dh_compress(1)` pakuje programem `gzip(1)` strony podręcznika i dokumentację większą niż 4 kB
- `dh_installdeb(1)` kopiuje pliki związane z pakietem (na przykład skrypty opiekuna) do katalogu `debian/gentoo/DEBIAN`
- `dh_shlibdeps(1)` wylicza zależności bibliotek i plików wykonywalnych od bibliotek współdzielonych
- `dh_gencontrol(1)` instaluje finalną wersję pliku 'control' w katalogu `debian/gentoo/DEBIAN`
- `dh_md5sums(1)` generuje sumy kontrolne MD5 dla każdego pliku zawartego w pakiecie

Pełniejsze informacje na temat działania każdego ze skryptów `dh_*` i ich parametrów wywołania znajdziesz na odpowiednich stronach podręcznika. Oprócz powyższych istnieją również inne użyteczne skrypty `dh_*`, które nie zostały tu wspomniane. Jeśli są potrzebne, czytaj dokumentację do pakietu `debhelper`.

W sekcji 'binary-arch' powinieneś wykomentować lub usunąć linie z tymi skryptami `dh_*`, których nie chcesz wywoływać. Dla pakietu `gentoo` wykomentowałem wywołanie skryptów `examples`, `cron`, `init`, `man` i `info`, gdyż `gentoo` ich po prostu nie używa. W linii 68. zamieniłem 'ChangeLog' na 'FIXES', ponieważ jest to rzeczywista nazwa autorskiego pliku z dziennikiem zmian.

Dwie ostatnie linie (i pozostałe nie opisane tutaj) są mniej lub bardziej niezbędne. Na ich temat możesz poczytać na stronie podręcznika do programu `make` oraz w Polityce Debiana. W tym momencie są na tyle mało ważne, że nie będziemy ich opisywać.

Rozdział 5

Inne pliki z katalogu debian/

Jak zobaczysz, w katalogu debian/ znajdują się jeszcze różne inne pliki, większość z nich kończy się przyrostkiem '.ex', oznaczającym, że są to przykłady. Przyjrzyj się im wszystkim. Jeśli chcesz lub musisz użyć którejś z ich funkcjonalności, to:

- zajrzyj do odpowiedniej dokumentacji (wskazówka: podręcznik Polityki Debiana)
- jeśli to konieczne, zmodyfikuj zawartość plików według potrzeb
- usuń z ich nazwy przyrostek '.ex', jeśli taki posiadają
- usuń z ich nazwy przedrostek 'ex.', jeśli taki posiadają
- zmodyfikuj plik 'rules' według potrzeb

Niektóre z tych plików, najczęściej używane, są objaśnione w poniższych sekcjach.

5.1 Plik 'README.Debian'

W tym pliku powinny być udokumentowane dodatkowe szczegóły lub rozbieżności pomiędzy oryginalnym pakietem i Twoją "zdebianaizowaną" wersją.

Program dh_make tworzy domyślny plik README.Debian, który wygląda jakoś tak:

```
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
```

Ponieważ nie musimy niczego umieszczać w tym pliku, możemy go skasować.

5.2 Plik 'conffiles.ex'

Jedną z najbardziej irytujących rzeczy związanych z oprogramowaniem jest to, że po poświęceniu dużej ilości czasu i wysiłku na dostosowanie programu, jego aktualizacja wszystko "zadeptuje". Debian rozwiązał ten problem przez znakowanie plików konfiguracyjnych. Zatem jeśli uaktualniasz program do nowszej wersji, to zostaniesz zapytany o to, czy chcesz zachować swoją starą konfigurację, czy nie.

Aby to uzyskać, musisz wprowadzić pełną ścieżkę do każdego pliku konfiguracyjnego (zwykle znajdują się w katalogu /etc), jednego w każdej linii, w pliku nazwanym `conffiles`. Program gentoo ma jeden plik konfiguracyjny, `/etc/gentoorc`, zatem podamy ścieżkę do niego w pliku `conffiles`.

Jeśli Twój program używa plików konfiguracyjnych, ale sam zmienia ich zawartość, to najlepiej będzie nie umieszczać ich w pliku `conffiles`, ponieważ program `dpkg` będzie za każdym razem prosił użytkowników o weryfikację zmian.

Jeśli program, który pakujesz, wymaga od każdego użytkownika modyfikacji pliku konfiguracyjnego, aby w ogóle zadziałać, to również powinieneś zrezygnować z umieszczenia go w pliku `conffiles`.

Przykładowe pliki konfiguracyjne możesz znaleźć w 'skryptach opiekuna pakietu'. Więcej szczegółów na ich temat zamieszczono w sekcji 'postinst.ex, preinst.ex, postrm.ex, prerm.ex' na [33](#) stronie.

Jeśli Twój program nie posiada plików konfiguracyjnych, to możesz bez obaw wykasować plik `conffiles` z katalogu debian/.

5.3 Plik 'cron.d.ex'

Jeśli Twój pakiet do prawidłowego działania wymaga regularnie wykonywanych zadań, to możesz do tego celu wykorzystać właśnie plik `cron.d`.

Zwróć uwagę, że to nie obejmuje zagadnień związanych z rotacją plików logów. Więcej informacji na ten temat znajdziesz na stronach podręcznika `dh_installlogrotate(1)` i `logrotate(8)`.

Jeśli nie potrzebujesz tego pliku, usuń go.

5.4 Plik 'dirs'

Plik ten określa katalogi, które są potrzebne, ale których normalna procedura instalacyjna (`make install`) nie tworzy.

Domyślnie plik ten wygląda następująco:

```
usr/bin
usr/sbin
```

Zwróć uwagę, iż przed nazwami katalogów nie występują znaki ukośników ('/'). Normalnie zmienilibyśmy go w następujący sposób:

```
usr/bin
usr/man/man1
```

ale ponieważ katalogi te są tworzone przez Makefile, to nie potrzebujemy pliku 'dirs' i możemy go usunąć.

5.5 Plik 'docs'

Ten plik określa nazwy plików z dokumentacją, którą program `dh_installdocs` zainstaluje w tymczasowym katalogu.

Domyślnie obejmuje to także pliki istniejące już w katalogu głównym ze źródłami programu, takie jak "BUGS", "README*", "TODO", itp.

Dla programu `gentoo` dołączyłem również inne pliki:

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

Możemy też usunąć ten plik i, zamiast podawać listę plików, użyć ich nazw jako argumentów wejściowych dla programu `dh_installdocs` wywoływanego w pliku `rules`:

```
dh_installdocs BUGS CONFIG-CHANGES CREDITS ONEWS README \
               README.gtkrc TODO
```

Może się tak zdarzyć, że nie będziesz mieć żadnego z tych plików w źródłach Twojego pakietu. W takim przypadku możesz bezpiecznie usunąć plik `docs`. Nie usuwaj jednak wywołań programu `dh_installdocs` z pliku `rules`, ponieważ jest on używany również do instalacji pliku `copyright` i innych rzeczy.

5.6 Plik 'emacsens*.ex'

Jeśli Twój pakiet zawiera pliki Emacsa, które mogą być skompilowane do kodu bajtowego w czasie instalacji, to możesz użyć tych plików właśnie w tym celu.

Pliki te są instalowane w katalogu tymczasowym przez program `dh_installemacsens(1)`, zatem jeśli chcesz go wywołać, nie zapomnij odkomentować odpowiedniej linii w pliku `rules`.

Jeśli zaś nie potrzebujesz tych plików, możesz je usunąć.

5.7 Plik 'init.d.ex'

Jeśli Twój pakiet jest demonem, który musi być uruchamiany w czasie startu systemu, to znaczy, że nie posłuchałeś moich zaleceń we wstępie do tego podręcznika, nieprawdaż? :-)

Plik ten jest prostym szablonem skryptu umieszczanego w katalogu `/etc/init.d/`, zatem będziesz musiał znacznie go przerobić. Zostanie on zainstalowany w katalogu tymczasowym przez program `dh_installinit(1)`.

Jeśli nie potrzebujesz tego pliku, usuń go.

5.8 Pliki 'manpage.1.ex', 'manpage.sgml.ex'

Twój program(y) powinien mieć stronę podręcznika systemowego. Jeśli jeszcze jej nie ma, to możesz użyć tych plików jako szablonów.

Strony podręcznika są zwykle napisane w formacie `nroff(1)`. W formacie tym napisano właśnie przykładowy plik `manpage.1.ex`. Zobacz stronę podręcznika programu `man(7)`, jest tam krótki opis, jak poprawiać tego typu pliki.

Jeśli zamiast formatu `nroff` wolisz pisać dokumenty w formacie SGML, to możesz wykorzystać szablon `manpage.sgml.ex`. W takim przypadku musisz:

- zainstalować pakiet `docbook-to-man`
- dopisać `docbook-to-man` do linii `Build-Depends` w pliku `control`
- usunąć znak komentarza przed wywołaniem programu `docbook-to-man` w regule 'build' pliku `rules`

Pamiętaj o zmianie nazwy pliku na coś w stylu `gentoo.sgml!`

Docelowa nazwa pliku ze stroną podręcznika systemowego powinna zawierać nazwę programu, który opisuje, zatem zmień ją z "manpage,, na "gentoo,,. Nazwa tego pliku zawiera także przyrostek ".1,, który mówi, że jest to strona z sekcji poleceń użytkownika. Upewnij się, do której sekcji powinna należeć strona Twojego pakietu. Poniżej zamieszczono krótką listę sekcji podręcznika:

Sekcja	Opis	Uwagi
1	Polecenia użytkownika	Wykonywalne komendy lub skrypty.
2	Wywołania systemowe	Funkcje jądra systemu.
3	Wywołania biblioteczne	Funkcje bibliotek systemowych.
4	Pliki specjalne	Zwykle umieszczone w katalogu /dev.
5	Formaty plików	Na przykład format pliku /etc/passwd.
6	Gry	Lub inne programy rozrywkowe.
7	Pakiety makr	Takie jak makra programu man.
8	Administracja systemem	Programy zwykle uruchamiane tylko przez adm
9	Procedury jądra	Niestandardowe wywołania i procedury wewnętrzne

Zatem podręcznik programu gentoo powinien nazywać się `gentoo.1`. Ponieważ w oryginalnych źródłach nie było strony podręcznika 'gentoo.1', napisałem ją sam, używając informacji znalezionych w przykładach i dokumentacji dołączonej do źródeł.

5.9 Plik 'menu.ex'

Użytkownicy systemu X Window zwykle posługują się menadżerami okien, umożliwiającymi uruchamianie programów poprzez rozwijalne menu, które można dostosowywać do własnych potrzeb. Jeśli zainstalowali oni pakiet `menu`, to zostanie utworzony zestaw menu służący do uruchamiania programów w systemie.

Poniżej pokazano plik `menu.ex`, domyślnie utworzony przez program `dh_make`:

```
?package (gentoo) : needs="X11|text|vc|wm" section="Apps/see-menu-manual" \
  title="gentoo" command="/usr/bin/gentoo"
```

Pierwszym polem po znaku dwukropka jest pole "needs,,", które określa, jakiego rodzaju interfejsu wymaga program. Zmień je na jedną z wymienionych możliwości, na przykład "text,,", lub "X11,,",

Następnym polem jest "section,,", które mówi, w jakim menu i podmenu powinien znaleźć się wpis z programem `gentoo`. Aktualną listę sekcji można znaleźć na stronie `/usr/share/doc/debian-policy/menu-policy.html/ch2.html#s2.1`.

Pole "title,,", to nazwa programu. Jeśli chcesz, możesz rozpocząć ją od wielkiej litery. Powinna być krótka.

Wreszcie pole "command,,", to nazwa polecenia, które uruchamia program.

Po zmianach wpis do menu wygląda następująco:

```
?package (gentoo) : needs="X11" section="Apps/Tools" title="Gentoo" command="
```

Możesz również dodać inne pola, na przykład "longtitle,,", "icon,,", "hints,,", itd. Więcej informacji możesz znaleźć na stronach podręcznika `menufile (5)`, `update-menus (1)` i w katalogu `/usr/share/doc/debian-policy/menu-policy.html/`.

5.10 Plik 'watch.ex'

Plik ten jest używany do konfigurowania programów `uscan(1)` i `uupdate(1)` (zawartych w pakiecie `devscripts`). Są one używane do sprawdzania strony internetowej, z której pobrałeś źródła dla swojego pakietu.

W tym pliku umieściłem:

```
# watch control file for uscan
# Site          Directory  Pattern          Version  Script
ftp.obsession.se /gentoo   gentoo-(.*)\.tar\.gz  debian   uupdate
```

Wskazówka: połącz się z internetem i spróbuj uruchomić program "uscan,, w katalogu, w którym stworzyłeś plik 'watch'. I czytaj strony podręczników! :)

5.11 Plik 'ex.package.doc-base'

Jeśli Twój pakiet ma dokumentację w postaci innej niż strony podręcznika i dokumentacja przeglądana za pomocą programu "info,, to powinieneś użyć pliku 'doc-base', aby ją zarejestrować. Użytkownik będzie mógł ją wtedy znaleźć, na przykład za pomocą `dhelptest(1)`, `dwww(1)` lub `doccentral(1)`.

Na ogół obejmuje to pliki HTML, PS i PDF umieszczone w katalogu `/usr/share/doc/nazwa_pakietu/`.

Plik `gentoo.doc-base` dla programu `gentoo` wygląda następująco:

```
Document: gentoo
Title: Gentoo Manual
Author: Emil Brink
Abstract: This manual describes what Gentoo is, and how it can be used.
Section: Apps/Tools

Format: HTML
Index: /usr/share/doc/gentoo/html/index.html
Files: /usr/share/doc/gentoo/html/*.html
```

Informacje na temat formatu tego pliku znajdziesz na stronie podręcznika `install-docs(8)` oraz w podręczniku `doc-base` (katalog `/usr/share/doc/doc-base/doc-base.html/`).

Więcej szczegółów nt. instalowania dokumentacji w sekcji 'Instalacja w podkatalogu' na [11](#) stronie.

5.12 `postinst.ex`, `preinst.ex`, `postrm.ex`, `prerm.ex`

Pliki te są nazywane skryptami opiekuna. Umieszczone są one w obszarze kontrolnym pakietu i uruchamiane przez program `dpkg`, gdy Twój pakiet jest instalowany, uaktualniany do nowszej wersji lub usuwany.

Na razie powinieneś unikać ręcznych modyfikacji tych skryptów, ponieważ często są one skomplikowane. Więcej informacji znajdziesz w podręczniku Polityki Debiana, w rozdziale 6. Zerknij także na przykładowe pliki wygenerowane przez program `dh_make`.

Rozdział 6

Budowanie pakietu

Teraz już powinniśmy być gotowi do zbudowania pakietu.

6.1 Całkowita przebudowa

Wejdź do katalogu głównego programu i wydaj w nim polecenie:

```
dpkg-buildpackage -rfakeroot
```

Wykona ono wszystko, to znaczy:

- wyczyści drzewo źródeł programu (debian/rules clean) używając programu `fakeroot`
- zbuduje pakiet źródłowy (`dpkg-source -b`)
- zbuduje program (debian/rules build)
- zbuduje pakiet binarny (debian/rules binary), używając programu `fakeroot`
- podpisze źródłowy plik `.dsc`, używając programu `gnupg`
- utworzy i podpisze umieszczany w archiwum Debiana plik `.changes` przy pomocy programów `dpkg-genchanges` i `gnupg`

Będziesz musiał tylko dwukrotnie wprowadzić hasło do Twojego prywatnego klucza GPG.

Po zakończeniu procesu zobaczysz następujące pliki w katalogu nadrzędnym (`~/gentoo/`):

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

To archiwum z oryginalnym kodem źródłowym programu. Jego nazwa została zmieniona w powyższy sposób, aby zachować standard Debiana. Zwróć uwagę, że plik ten został utworzony przy użyciu opcji `'-f'` przez program `dh_make`, gdy na początku go uruchomiliśmy.

- *gentoo_0.9.12-1.dsc*

To jest streszczenie zawartości kodu źródłowego. Plik ten jest generowany na podstawie pliku 'control' i używany w czasie rozpakowywania źródła przez program `dpkg-source(1)`. Jest on podpisany cyfrowo, aby inni mogli być pewni, że jest naprawdę Twój.

- *gentoo_0.9.12-1.diff.gz*

Ten plik jest skompresowany i zawiera wszystkie zmiany, których dokonałeś w oryginalnym kodzie źródłowym. Zmiany te są zapisane w formacie znanym jako "unified diff,,". Plik jest utworzony i używany przez program `dpkg-source(1)`. Uwaga: jeśli nie nazwałeś oryginalnego archiwum ze źródłami programu w sposób: `nazwapakietu_wersja.orig.tar.gz`, to program `dpkg-source` nie wygeneruje poprawnego pliku `.diff.gz`!

Gdyby ktoś jeszcze chciał ponownie utworzyć Twój pakiet zaczynając procedurę od początku, to może łatwo to zrobić używając trzech powyższych plików. Procedura postępowania w takich przypadkach jest wręcz banalna: po prostu należy gdzieś skopiować te trzy pliki i wydać komendę `dpkg-source -x gentoo_0.9.12-1.dsc`.

- *gentoo_0.9.12-1_i386.deb*

To kompletny pakiet binarny. Możesz użyć programu `dpkg`, aby zainstalować go lub usunąć w taki sam sposób, jak każdy inny pakiet.

- *gentoo_0.9.12-1_i386.changes*

Plik ten opisuje wszystkie zmiany dokonane w obecnej poprawce pakietu. Używają go programy obsługi archiwów FTP Debiana do zainstalowania pakietów binarnych i źródłowych. Jest on częściowo generowany z plików 'changelog' i `.dsc`. Plik ten jest podpisany cyfrowo, aby inni mogli być pewni, że jest naprawdę Twój.

W czasie, gdy będziesz się zajmował pakietem, zmieni się pewnie jego działanie i dodane zostaną nowe funkcjonalności. Ludzie pobierający Twój pakiet mogą w tym pliku szybko zobaczyć, co się zmieniło. Programy zarządzające archiwum Debiana wyślą również zawartość tego pliku na listę dyskusyjną `debian-devel-changes`.

Długie łańcuchy liczb w plikach `.dsc` i `.changes` to sumy kontrolne MD5 wspomnianych plików. Osoby pobierające Twoje pliki mogą sprawdzić je używając programu `md5sum(1)` i jeśli sumy nie będą się zgadzać, będą wiedzieć, że plik jest uszkodzony lub został przez kogoś zmieniony.

6.2 Szybka przebudowa

Gdy masz duży pakiet, to możesz nie chcieć budować go od nowa za każdym razem, gdy zmienisz jakiś szczegół w pliku `debian/rules`. Dla celów testowych możesz stworzyć plik `.deb` bez przebudowywania źródeł programu:

```
fakeroot debian/rules binary
```

Gdy już zakończyłeś szlifowanie Twojego pakietu, pamiętaj o przebudowaniu go zgodnie z powyższą, pełną procedurą. Może Ci się nie udać umieścić go w archiwum Debiana, gdy próbujesz zamieścić tam pliki .deb zbudowane w skrócony sposób.

6.3 Polecenie `debuild`

Możesz zautomatyzować proces budowania pakietu za pomocą polecenia `debuild`. Zobacz `debuild(1)`.

Można skonfigurować działanie programu `debuild` poprzez pliki `/etc/devscripts.conf` lub `~/.devscripts`. Chciałbym zasugerować co najmniej:

```
DEBSIGN_KEYID="Your_GPG_keyID"  
DEBUILD_DPKG_BUILDPACKAGE_OPTS="-i -ICVS -I.svn"
```

W ten sposób możesz budować pakiet zawsze ze swoim kluczem GPG i bez niepotrzebnych komponentów (zwłaszcza pomocne przy sponsoringu). Przykładowo czyszczenie źródeł i przebudowa pakietu z konta użytkownika jest tak proste:

```
debuild clean  
debuild
```

6.4 Systemy `dpatch` i `quilt`

Proste użycie `dh_make` i `dpkg-buildpackage` spowoduje utworzenie pojedynczego dużego pliku `diff.gz`, zawierającego pliki obsługujące pakiet w katalogu `debian/` i plików łańcuchów źródeł. Taki pakiet jest uciążliwy podczas późniejszego sprawdzania dla kogoś, kto powinien zrozumieć wszystkie wprowadzane do źródeł zmiany. Nie jest to zbyt przyjemne¹.

Zaproponowano różne metody obsługi łańcuchów źródeł i używa się ich w pakietach Debiana. Systemy `dpatch` i `quilt` są najprostszymi z nich. Z innych wymienimy `dbs`, `cdbs` itd.

Pakiet, który został poprawnie zbudowany za pomocą systemu `dpatch` lub `quilt`, posiada dobrze udokumentowane poprawki w źródłach jako zbiór plików łańcuchów `-pl` z nagłówkiem w katalogu `debian/patches/` i niezmiennym drzewem źródeł poza katalogiem `debian/`. Kiedy prosisz sponsora o wgranie Twojego pakietu, tego rodzaju wyraźne rozdzielanie i udokumentowanie Twoich zmian jest bardzo ważne dla sprawnego przejrzania ich przez sponsora. Użycie `dpatch` oraz `quilt` jest opisane w podręcznikach `dpatch(1)`, `dpatch-edit-patch(1)`

¹Jeśli nie jesteś jeszcze Deweloperem Debiana i prosisz swojego sponsora o wgranie pakietu po jego sprawdzeniu, powinieneś przygotować pakiet w sposób możliwie czytelny.

i quilt (1). Oba programy udostępniają wygodne do umieszczenia w `debian/rules` pliki `/usr/share/dpatch/dpatch.make` i `/usr/share/quilt/quilt.make`.

Kiedy ktoś (włączając Ciebie) chce wdrożyć poprawki do źródeł, modyfikacja pakietu jest całkiem prosta:

- Wyedytuj łąkę, tak aby otrzymać plik `-pl` jako poprawkę do źródeł.
- W przypadku `dpatch` dodaj nagłówek poleceniem `'dpatch patch-template'`.
- Przenieś nagłówki do katalogu `debian/patches`.
- Dodaj nazwy plików do pliku `debian/patches/00list` (dla `dpatch`) lub `debian/patches/series` (dla `quilt`).

`dpatch` posiada też możliwość wykonywania poprawek zależnych od architektury za pomocą makra `CPP`.

6.5 Dołączanie `orig.tar.gz` podczas wgrywania

Kiedy pierwszy raz wgrywasz pakiet do archiwum, powinieneś załączyć oryginalny plik źródłowy `orig.tar.gz`. Jeśli wersja pakietu nie jest rewizją `-0` lub `-1`, musisz wykonać polecenie `dpkg-buildpackage` z opcją `"-sa,,`. Z drugiej strony opcja `"-sd,,` wymusi wyłączenie oryginalnego źródła `orig.tar.gz`.

Rozdział 7

Sprawdzanie pakietu pod kątem błędów

7.1 Pakiety `lintian`

Uruchom programy `lintian(1)` podając jako argument swój plik `.changes`. Programy ten sprawdzają pakiet pod kątem wielu błędów występujących podczas pakowania. Robi się to poleceniami:

```
lintian -i gentoo_0.9.12-1_i386.changes
```

Oczywiście zastąp nazwę pliku `.changes` nazwą pliku wygenerowanego dla Twojego pakietu. Jeśli pojawią się informacje o błędach (linie rozpoczynające się od "E:,,), to przeczytaj ich objaśnienie (linie rozpoczynające się od "N:,,), popraw błędy i ponownie zbuduj pakiet w sposób opisany w sekcji 'Całkowita przebudowa' na 35 stronie. Linie, które zaczynają się od "W:,,), to ostrzeżenia. Oczywiście powinieneś albo coś poprawić, albo upewnić się, że nie są ważne (i w takim przypadku wymusić na Lintianie ich ignorowanie; więcej szczegółów znajdziesz w dokumentacji).

Zwróć uwagę, że możesz w jednym kroku zbudować pakiet za pomocą programu `dpkg-buildpackage` i uruchomić programy `lintian` korzystając z narzędzia `debuild(1)`.

7.2 Polecenie `mc`

Możesz rozpakować zawartość pliku `*.deb` przy pomocy polecenia `dpkg-deb(1)`. Możesz też zobaczyć zawartość utworzonego pakietu Debiana używając `debc(1)`.

Można to również osiągnąć w sposób bardziej intuicyjny za pomocą menedżera plików np. `mc(1)`, który pozwala przeglądać nie tylko zawartość pliku `*.deb`, ale również pliki `*.diff.gz` i `*.tar.gz`.

Zwróć przy tym uwagę na zbędne pliki o zerowej wielkości, zarówno binarne, jak i źródłowe. Często nie są one kasowane w sposób prawidłowy; popraw plik `rules`, tak aby usunąć tę wadę.

Porada: `'zgrep ^+++ ../gentoo_0.9.12-1.diff.gz'` poda listę Twoich zmian w plikach źródłowych, a `'dpkg-deb -c gentoo_0.9.12-1_i386.deb'` lub `'debc gentoo_0.9.12-1_i386.changes'` - listę plików w pakiecie binarnym.

7.3 Polecenie `debdiff`

Możesz porównać listy plików w dwóch pakietach binarnych Debiana za pomocą polecenia `debdiff(1)`. Jest to bardzo użyteczne do sprawdzenia, że żadne pliki nie zostały błędnie przesunięte lub wykasowane, lub też nie zostały dokonane inne niepożądane zmiany podczas modyfikacji pakietu. Możesz sprawdzić całą grupę plików `*.deb` przez wywołanie `'debdiff old-package.change new-package.change'`.

7.4 Polecenie `interdiff`

Możesz porównać dwa pliki `diff.gz` za pomocą polecenia `interdiff(1)`. Jest to pomocne w celu sprawdzenia, czy nie zostały przez nieuwagę opiekuna wprowadzone jakieś zmiany do plików źródłowych podczas modyfikacji pakietów. Uruchom po prostu `'interdiff -z old-package.diff.gz new-package.diff.gz'`.

7.5 Polecenie `debi`

Zainstaluj pakiet dla przetestowania go, np. za pomocą polecenia `debi(1)`, wywołanego jako `root`. Spróbuj zainstalować i uruchomić go na innych maszynach, kontrolując, czy w czasie instalacji lub działania nie pojawiają się komunikaty o błędach lub ostrzeżenia.

7.6 Pakiet `pbuilder`

Pakiet `pbuilder` jest bardzo użyteczny w przypadku weryfikacji zależności dla programów uruchamianych w tzw. środowisku czystej przestrzeni (`chroot`). Zapewnia on "czystą" budowę pakietu ze źródeł za pomocą auto-buildera dla różnych architektur i zapobiega powstawaniu szeregu błędów FTBFS (Fails To Build From Source), które zawsze należą do kategorii RC (release critical - krytyczne dla wydania). Zobacz stronę <http://buildd.debian.org/> o debianowym pakiecie auto-builder.

Najbardziej podstawowym użyciem pakietu `pbuilder` jest bezpośrednio wywołanie polecenia `pbuilder` jako `root`. Dla przykładu spróbuj wywołać poniższe polecenia w katalogu, gdzie znajdują się pliki `.orig.tar.gz`, `.diff.gz` i `.dsc`:

```
root # pbuilder create # za drugim razem: pbuilder update
root # pbuilder build foo.dsc
```

Nowozbudowany pakiet znajdzie się w katalogu `/var/cache/pbuilder/result/`, właścicielem będzie root.

Polecenie `pdebuild` pomaga wykorzystywać funkcje pakietu `pbuilder` z konta zwykłego użytkownika. Z katalogu źródeł pakietu, gdzie w katalogu nadrzędnym znajduje się plik `orig.tar.gz`, wykonaj polecenia:

```
$ sudo pbuilder create # za drugim razem: sudo pbuilder update
$ pdebuild
```

Nowozbudowany pakiet znajdzie się w katalogu `/var/cache/pbuilder/result/`, właścicielem będzie użytkownik (nie-root) ¹.

Jeśli chcesz dodać nowe źródło apta, tak aby było używane przez pakiet `pbuilder`, ustaw `OTHERMIRROR` w `~/pbuilder.conf` lub `/etc/pbuilder.conf` i wykonaj (dla `sarge`):

```
$ sudo pbuilder update --distribution sarge --override-config
```

Użycie `--override-config` jest konieczne do zaktualizowania źródeł apt w środowisku `chroot`.

Zobacz <http://www.netfort.gr.jp/~dancer/software/pbuilder.html>, `pdebuild(1)`, `pbuilder.conf(5)` i `pbuilder(8)`.

¹W chwili obecnej zalecam dostosowanie systemu przez umożliwienie zapisu w katalogu `/var/cache/pbuilder/result/` przez użytkowników i dodanie w pliku `~/pbuilder.conf` lub `/etc/pbuilder.conf` wpisu

```
AUTO_DEBSIGN=yes
```

Pozwoli to na podpisywanie tworzonych pakietów Twoim kluczem GPG z pliku `~/gnupg/`. Ponieważ pakiet `pbuilder` jest ciągle rozwijany, sprawdź w jego bieżącej dokumentacji, w jaki sposób powinieneś go skonfigurować.

Rozdział 8

Wgrywanie pakietu

Gdy już gruntownie przetestowałeś swój nowy pakiet, jesteś gotowy, aby wziąć udział w procesie przyjmowania do Debiana nowego opiekuna pakietów. Jest to opisane na stronie <http://www.debian.org/devel/join/newmaint>.

8.1 Wgrywanie pakietu do archiwum Debiana

Gdy już zostałeś oficjalnym deweloperem, powinieneś umieścić swój pakiet w archiwum Debiana. Możesz zrobić to ręcznie, ale łatwiej jest użyć specjalnie do tego celu stworzonych narzędzi, które automatyzują cały proces. Należą do nich takie programy, jak `dupload(1)` i `dput(1)`. Opiszemy tutaj, w jaki sposób posługiwać się programem `dupload`.

Pierwszą rzeczą, którą powinieneś zrobić, jest edycja jego pliku konfiguracyjnego. Możesz wyedytować zarówno przeznaczony dla całego systemu plik `/etc/dupload.conf`, jak i swój własny plik `~/dupload.conf`, który nadpisuje te rzeczy, które chcesz zmienić. Umieść w nim coś takiego:

```
package config;

$default_host = "anonymous-ftp-master";

$cfg{'anonymous-ftp-master'} = {
    fqdn => "ftp-master.debian.org",
    method => "ftp",
    incoming => "/pub/UploadQueue/",
    # files pass on to dinstall on ftp-master which sends emails itself
    # pliki przekazywane do dinstall na ftp-master, które wysyłają maile
    dinstall_runs => 1,
};

1;
```

Możesz przeczytać stronę podręcznika `dupload.conf(5)`, aby zrozumieć, co oznacza każda z użytych opcji.

Uwagi wymaga zmienna `$default_host` – określa ona, która z kolejek służących do umieszczania pakietów jest używana domyślnie. Główną kolejką jest "anonymous-ftp-master", ale możliwe jest, że będziesz chciał użyć innej, szybszej. Więcej informacji na temat kolejek znajdziesz w dokumencie Developers' Reference, w sekcji "Uploading a package", która znajduje się w dokumencie `/usr/share/doc/developers-reference/ch-pkgs.en.html#s-upload`.

Następnie połącz się z internetem i wydaj polecenie:

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

Program `dupload` sprawdzi, czy zgadzają się sumy kontrolne MD5 plików z sumami zapisanymi w pliku `.changes`. Jeśli sumy kontrolne pasują do siebie, pakiet może być umieszczony w archiwum. Jeśli sumy się nie zgadzają, zostaniesz ostrzeżony, aby móc przebudować pakiet zgodnie z procedurą opisaną w rozdziale 'Całkowita przebudowa' na 35 stronie.

Jeśli stwierdzisz jakiś problem z wgrywaniem w kolejce <ftp://ftp-master.debian.org/pub/UploadQueue/>, możesz naprawić go ręcznie poprzez wgranie podpisanego pliku `*.commands` do <ftp://ftp-master.debian.org/pub/UploadQueue/> za pomocą programu `ftp`¹. Przykładowy plik `hello.commands`:

```
-----BEGIN PGP SIGNED MESSAGE-----

Uploader: Roman Hodek <Roman.Hodek@informatik.uni-erlangen.de>
Commands:
  rm hello_1.0-1_i386.deb
  mv hello_1.0-1.dsx hello_1.0-1.dsc

-----BEGIN PGP SIGNATURE-----
Version: 2.6.3ia

iQCVAwUBNFiQSXVhJ0HiWnvJAQG58AP+IDJVeSWmDvzMUphScg1EK0mvChgnuD7h
BRiVQubXkB2DphLJW5UUSRnjlwFcyWwH/lFpNpl7XP95LkLX3iFza9qItw4k2/q
tvylZkmIA9jxCyv/YB6zZCbHmbvUnL473eLRoxlnYZd3JFaCZMJ86B0Ph4GFNPaf
Z4jxNrgh7Bc=
=pH94
-----END PGP SIGNATURE-----
```

¹Zobacz <ftp://ftp-master.debian.org/pub/UploadQueue/README>. Możesz też użyć programu `dcut` z pakietu `dput`.

8.2 Wgrywanie do prywatnego archiwum

Jeśli chcesz utworzyć swoje prywatne archiwum na stronie URL="http://people.debian.org/~*account_name*," jako deweloper, a potem w prosty sposób wywoływać `dupload -t target_name`, powinieneś dodać następujące linie do pliku `/etc/dupload.conf`:

```
# Developer account
$cfg{'target_name'} = {
    fqdn => "people.debian.org",
    method => "scpb",
    incoming => "/home/account_name/public_html/package/",
    # I do not need to announce
    dinstall_runs => 1,
};
$cfg{'target_name'}{preupload}{'changes'} = "
    echo 'mkdir -p public_html/package' | ssh people.debian.org 2>/dev/n
    echo 'Package directory created!';

$cfg{'target_name'}{postupload}{'changes'} = "
    echo 'cd public_html/package ;
    dpkg-scanpackages . /dev/null >Packages || true ;
    dpkg-scansources . /dev/null >Sources || true ;
    gzip -c Packages >Packages.gz ;
    gzip -c Sources >Sources.gz ' | ssh people.debian.org 2>/dev/null ;
    echo 'Package archive created!';
```

Tutaj archiwum APT jest budowane poprzez szybkie i nie sprawdzane zdalne wywołanie powłoki przez SSH. Nadpisywane pliki wymagane przez `dpkg-scanpackages` i `dpkg-scansources` są podstawiane przez `/dev/null`. Technika ta może być stosowana przez nie-deweloperów Debiana do wprowadzania ich pakietów na ich osobiste strony internetowe. Można też używać `apt-ftparchive` albo innych skryptów do tworzenia archiwów APT.

Rozdział 9

Aktualizacja pakietu

9.1 Nowa poprawka Debiana

Powiedzmy, że do Twojego pakietu został zgłoszony raport o błędzie o numerze #54321, opisujący problem, który możesz rozwiązać. Aby stworzyć nową poprawkę (revision) pakietu Debiana, musisz wykonać następujące czynności:

- Oczywiście najpierw popraw błąd w źródłach pakietu.
- Dodaj nową poprawkę na początku pliku 'changelog', na przykład za pomocą 'dch -i' lub wręcz 'dch -v <wersja>-<rewizja>' i za pomocą ulubionego edytora tekstu wstaw komentarze.
Porada: w jaki sposób najłatwiej pobrać datę w wymaganym formacie? Użyj komendy '822-date' lub 'date -R'.
- Dołącz krótki opis błędu i jego rozwiązania do pliku 'changelog' oraz napis: "Closes: #54321,..". W ten sposób raport o błędzie zostanie automatycznie "zamknięty" przez oprogramowanie obsługujące archiwum Debiana w chwili, gdy pakiet zostanie w nim zaakceptowany.
- Powtórz kroki wykonywane w rozdziałach 'Całkowita przebudowa' na 35 stronie, 'Sprawdzanie pakietu pod kątem błędów' na 39 stronie i 'Wgrywanie pakietu' na 43 stronie. Jedyną różnicą będzie to, że nie zostaną wgrane oryginalne źródła, gdyż nie zmieniły się i znajdują się już w archiwum Debiana.

9.2 Nowe wydanie autorskie (prosto)

Rozważmy teraz trochę inną, troszkę bardziej skomplikowaną sytuację - została wydana nowa, zewnętrzna wersja programu i oczywiście chcemy ją zapakować. Trzeba wykonać następujące czynności:

- Pobierz archiwum z nowymi źródłami (na przykład 'gentoo-0.9.13.tar.gz') i umieść je w katalogu nadrzędnym do katalogu ze starym drzewem źródeł (dla przykładu ~/gentoo/).
- Wejdź do katalogu ze starymi źródłami i wykonaj:

```
uupdate -u gentoo-0.9.13.tar.gz
```

Oczywiście musisz zastąpić nazwę pliku nazwą archiwum ze źródłami Twojego programu. Program `uupdate(1)` odpowiednio zmieni nazwę tego archiwum, spróbuje nałożyć wszystkie zmiany z Twojego poprzedniego pliku `.diff.gz` i uaktualni plik `debian/changelog`.

- Zmień katalog na `'.. /gentoo-0.9.13'`, czyli drzewo z nowym źródłem pakietu i powtórz to, co robiłeś w rozdziałach 'Całkowita przebudowa' na 35 stronie, 'Sprawdzanie pakietu pod kątem błędów' na 39 stronie i 'Wgrywanie pakietu' na 43 stronie.

Zauważ, że jeśli skonfigurowałeś plik 'debian/watch', jak to opisano w sekcji 'Plik 'watch.ex'' na 32 stronie, to możesz uruchomić program `uscan(1)`, aby automatycznie odzukiwać poprawione źródła, pobierać je i uruchamiać program `uupdate`.

9.3 Nowe wydanie autorskie (realistycznie)

Kiedy przygotowujesz pakiety dla archiwum Debiana, musisz szczegółowo sprawdzać rezultaty swoich działań. Przedstawiam poniżej bardziej realistyczny przykład takiej procedury.

1 Sprawdź zmiany w źródłach zewnętrznych

- Przeczytaj autorskie pliki `changelog`, `NEWS` i inną dokumentację, która może się odnosić do nowej wersji.
- Wykonaj `'diff -urN'` pomiędzy starymi i nowymi źródłami autorskimi, aby zapoznać się z dokonanymi zmianami (a które mogą powodować potencjalne błędy), miej oczy otwarte na wszystko, co wygląda podejrzanie.

2 Zaadaptuj stary pakiet do nowej wersji.

- Rozpakuj archiwum źródłowe i zmień nazwę jego głównego katalogu na `<nazwa_pakietu>-<wersja_autora>/` i przejdź `'cd'` do tego katalogu.
- Skopiuuj archiwum źródłowe w katalogu nadrzędnym z nową nazwą `<nazwa_pakietu>_<wersja_autora>.orig.tar.gz`.
- Wykonaj takie same czynności na nowych źródłach, jakie wykonałeś na starych. Są możliwe następujące sposoby:
 - polecenie `'zcat /path/to/<nazwa_pakietu>_<stara_wersja>.diff.gz | patch -p1'`
 - polecenie `'uupdate'`
 - polecenie `'svn merge'`, jeśli źródła są w repozytorium Subversion lub
 - po prostu przekopiuj katalog `debian/` ze starych źródeł, jeśli były spakowane przy pomocy `dpatch` lub `quilt`.

- Zachowaj stare wpisy w pliku changelog (wydaje się oczywiste, ale różnie to bywa...)
 - Nowa wersja pakietu będzie miała numer autorski uzupełniony przez `-1` - numer rewizji Debiana, np. `'0.9.13-1'`.
 - Dodaj wpis "New upstream release,, na początku pliku `debian/changelog`. Możesz to zrobić np. przez wywołanie `'dch -v 0.9.13-1'`.
 - Zwięźle opisz (po angielsku oczywiście [*uwaga tłumacza*]) zmiany dokonane w nowej wersji autorskiej, które naprawiają zgłoszone błędy i zamknij te zgłoszenia w pliku `changelog`.
 - Zwięźle opisz zmiany dokonane przez opiekuna *odnoszące się do* nowej wersji autorskiej, które naprawiają zgłoszone błędy i zamknij zgłoszenia.
 - Jeśli poprawka (`patch/merge`) nie może być zaaplikowana w sposób bezpośredni, zbadaj sytuację, żeby stwierdzić jaka jest przyczyna powstawania trudności (na podstawie wpisów w plikach `.rej`). W większości przypadków problem polega na tym, że poprawka została uwzględniona przez autora i nie jest już potrzebna.
 - Aktualizacja do nowej wersji powinna przebiegać w sposób nieabsorbujący użytkownika (użytkownicy nie powinni zauważać tego faktu, poza stwierdzeniem, że błędy zostały poprawione lub dodano nowe funkcje) ¹.
 - Jeśli z jakiegoś powodu trzeba dodać skasowane wcześniej pliki szablonów, możesz uruchomić program `dh_make` w już wcześniej "zdebianizowanym,, katalogu z opcją `-o`, a następnie uważnie je wyedytować.
 - Uprzednio istniejące zmiany Debiana powinny zostać ponownie wprowadzone; odrzuć poprawki, które zostały już uwzględnione (w ten czy inny sposób) przez autora zewnętrznego, ale pamiętaj o tych pozostałych, aż do chwili, kiedy naprawę przestaną być potrzebne.
 - Jeśli były jakieś zmiany w systemie budującym (mam nadzieję, że wiesz o tym z kroku 1.), popraw plik `debian/rules` i, jeśli to konieczne, zależności w pliku `debian/control`.
- 3 Zbuduj nowy pakiet jak w rozdziale 'Polecenie `debuild`' na 37 stronie lub 'Pakiet `pbuilder`' na 40 stronie. Użycie pakietu `pbuilder` jest wskazane.
- 4 Sprawdź, czy nowe pakiety zostały zbudowane w sposób prawidłowy.
- Przeprowadź 'Sprawdzanie pakietu pod kątem błędów' na 39 stronie.
 - Wykonaj 'Weryfikowanie uaktualnienia pakietu do nowszej wersji' na następnej stronie.
 - Sprawdź ponownie, czy nie ma poprawionych błędów otwartych w Debian Bug Tracking System (BTS) (<http://www.debian.org/Bugs/>).
 - Sprawdź zawartość pliku `.changes` i upewnij się, że wgrywasz zmiany do odpowiedniej dystrybucji, zostały ujęte właściwe zgłoszenia błędów w polu `Closes:`, pola `Maintainer:` i `Changed-By:` są prawidłowe, istnieje podpis GPG itp.

¹Proszę przygotować również prawidłową aktualizację pliku konfiguracyjnego Twojego pakietu za pomocą dobrze napisanych skryptów `postinst` itp., tak aby **nie** działały się rzeczy zaskakujące użytkowników! To m. in. jest jednym z czynników, dla których ludzie wybierają Debiana. Jeśli rzeczywiście istnieje konieczność ręcznej interwencji (np. konfiguracja dla różnych katalogów domowych o całkowicie odmiennej strukturze), pomyśl o domyślnym skonfigurowaniu pakietu w sposób najbardziej bezpieczny (np. przez wyłączenie usługi) i przygotowaniu właściwej dokumentacji (pliki `README.Debian` i `NEWS.Debian`), jako ostateczność. Nie zajmuj jednak użytkownika komunikatami `debconfa`.

- 5 Jeśli wykonywałeś jakieś zmiany, aby coś poprawić, wróć do kroku 2., aż do osiągnięcia pozytywnego rezultatu.
- 6 Jeśli wgrywanie odbywa się przez sponsora, odnotuj wszelkie dodatkowe opcje wymagane do zbudowania pakietu (np. `'dpkg-buildpackage -sa -v ...'`) i poinformuj o nich sponsora, tak aby mógł również prawidłowo go zbudować.
- 7 Jeśli sam wykonujesz wgrywanie, przeprowadź 'Wgrywanie pakietu' na 43 stronie.

9.4 Plik `orig.tar.gz`

Jeśli próbujesz zbudować pakiet tylko z nowego katalogu źródłowego z podkatalogiem `debian/` bez pliku `orig.tar.gz` w katalogu nadrzędnym, zakończy się to utworzeniem pakietu źródłowego, bez powstania pliku `diff.gz`. Taki sposób pakowania jest prawidłowy jedynie dla specyficznych pakietów Debiana, które są bezużyteczne w innych dystrybucjach.²

W celu uzyskania nie-natywnego pakietu źródłowego, który pozwala na uzyskanie zarówno pliku `orig.tar.gz` jak i `diff.gz`, musisz ręcznie skopiować archiwum autorskie do katalogu nadrzędnego i zmienić jego nazwę na `<nazwa_pakietu>_<wersja_autorska>.orig.tar.gz`, jak zrobił to program `dh_make` w rozdziale 'Wstępna "debianizacja,"' na 10 stronie.

9.5 Polecenie `cvs-buildpackage` i jemu podobne

Rozważ użycie systemu zarządzania kodem źródłowym do zarządzania plikami zmienianymi w czasie pakowania. Jest kilka skryptów do pakowania przystosowanych do najbardziej popularnych z nich.

- CVS
 - `cvs-buildpackage`
- Subversion
 - `svn-buildpackage`
- Git (`git-core`)
 - `git-buildpackage`

Polecenia te automatyzują też pakowanie nowych wydań autorskich.

9.6 Weryfikowanie uaktualnienia pakietu do nowszej wersji

Kiedy już zbudowałeś nową wersję pakietu, powinieneś wykonać następującą procedurę, żeby upewnić się, że aktualizacja pakietu do nowej wersji przebiega bezbłędnie:

²Niektórzy twierdzą, że nawet dla specyficznych pakietów Debiana jest lepszą praktyką przechowywanie zawartości katalogu `debian/` w pliku `diff.gz`, niż w `orig.tar.gz`.

- uaktualnij pakiet z poprzedniej wersji
- powróć ponownie do poprzedniej wersji (downgrade), a następnie usuń go
- zainstaluj pakiet jako nowy pakiet
- odinstaluj go i następnie zainstaluj ponownie
- wyczyść (purge) pakiet

Jeśli pakiet zawiera nietrywialne skrypty pre/post/inst/rm, przetestuj ścieżki działania każdego z nich.

Miej świadomość, że jeśli Twój pakiet był poprzednio wydany w Debianie, to ludzie często będą go uaktualniać z wersji, która była w ostatnim wydaniu Debiana. Pamiętaj, żeby przetestować także uaktualnianie do nowszej wersji z tamtej wersji.

Rozdział 10

Gdzie prosić o pomoc

Zanim zdecydujesz się zadać pytanie w jakimś publicznym miejscu, proszę najpierw zajrzeć do odpowiedniego podręcznika. Dokumentacja do programów wymienionych w tym dokumencie znajduje się w `/usr/share/doc/dpkg`, `/usr/share/doc/debian`, `/usr/share/doc/autotools-dev/README.Debian.gz`, `/usr/share/doc/package/*`, a także w plikach `info` i `man` omawianych programów. Zobacz też informacje na <http://nm.debian.org/> i http://people.debian.org/~mpalmer/debian-mentors_FAQ.html.

Jeśli masz pytanie na temat pakowania, na które nie znalazłeś odpowiedzi w powyższej dokumentacji, powinieneś je zadać na liście Mentorów Debiana, która dostępna jest pod adresem `<debian-mentors@lists.debian.org>`. Bardziej doświadczeni deweloperzy Debiana chętnie Ci pomogą, ale przed zadaniem pytania przeczytaj dokumentację!

Więcej informacji na temat tej listy dyskusyjnej znajdziesz na stronie <http://lists.debian.org/debian-mentors/>.

Kiedy odbierzesz raport o błędzie (tak, prawdziwy raport o błędzie!), to znak, że czas zaznajomić się z Systemem śledzenia błędów Debiana (BTS) (<http://www.debian.org/Bugs/>) i przeczytać znajdującą się tam dokumentację, aby móc sprawnie radzić sobie z takimi raportami. Polecam głównie przeczytanie rozdziału "Handling Bugs," z dokumentu `Developers' Reference` w `/usr/share/doc/developers-reference/ch-pkgs.en.html#s-bug-handling`.

Jeśli wciąż masz pytania, pytaj na liście Deweloperów Debiana, która jest dostępna pod adresem `<debian-devel@lists.debian.org>`. Więcej informacji na temat tej listy dyskusyjnej znajdziesz na stronie <http://lists.debian.org/debian-devel/>.

Nawet, gdy wszystko działa dobrze, to czas, żeby zacząć się modlić. Dlaczego? Ponieważ już za kilka godzin (lub dni) użytkownicy z całego świata zaczną używać Twojego pakietu i jeśli popełniłeś jakiś krytyczny błąd, wielu rozgniewanych użytkowników Debiana zasypie Cię listami... To tylko żart. :-)

Zrelaksuj się i bądź gotowy na raporty o błędach, ponieważ masz jeszcze sporo do zrobienia, zanim Twój pakiet będzie w pełni zgodny z Polityką Debiana (powtarzam: przeczytaj *prawdziwą dokumentację*, aby dowiedzieć się więcej szczegółów). Powodzenia!

Dodatek A

Przykłady

Mamy do zapakowania archiwum autorskie *gentoo-1.0.2.tar.gz* i wgrywamy wszystkie pakiety do *nm_target*.

A.1 Prosty przykład pakowania

```
$ mkdir -p /path/to # nowy pusty katalog
$ cd /path/to
$ tar -xvzf /path/from/gentoo-1.0.2.tar.gz # rozpakowujemy źródła
$ cd gentoo-1.0.2
$ dh_make -e name@domain.dom -f /path/from/gentoo-1.0.2.tar.gz
... Odpowiedz na pytania
... Popraw drzewo źródeł
... Jeśli jest to pakiet skryptowy, wpisz "Architecture: all" do debian/con
... Nie kasuj ../gentoo_1.0.2.orig.tar.gz
$ debuild
... Nie powinno być ostrzeżeń
$ cd ..
$ dupload -t nm_target gentoo_1.0.2-1_i386.changes
```

A.2 Przykład z *dpatch* i *pbuilder*

```
$ mkdir -p /path/to # nowy pusty katalog
$ cd /path/to
$ tar -xvzf /path/from/gentoo-1.0.2.tar.gz
$ cp -a gentoo-1.0.2 gentoo-1.0.2-orig
$ cd gentoo-1.0.2
$ dh_make -e name@domain.dom -f /path/from/gentoo-1.0.2.tar.gz
... Odpowiedz na pytania
```

Tu mamy fragment oryginalnego pliku `debian/rules`:

```
configure: configure-stamp
configure-stamp:
    dh_testdir
    # Add here commands to configure the package.
    touch configure-stamp
build: build-stamp
build-stamp: configure-stamp
    dh_testdir
    # Add here commands to compile the package.
    $(MAKE)
    #docbook-to-man debian/gentoo.sgml > gentoo.1
    touch $@
clean:
    dh_testdir
    dh_testroot
    rm -f build-stamp configure-stamp
    # Add here commands to clean up after the build process.
    -$(MAKE) clean
    dh_clean
```

Za pomocą edytora zmień `debian/rules` w następujący sposób, tak aby używać `dpatch` oraz dodaj `dpatch` w linii `Build-Depends`: pliku `debian/control`:

```
configure: configure-stamp
configure-stamp: patch
    dh_testdir
    # Add here commands to configure the package.
    touch configure-stamp
build: build-stamp
build-stamp: configure-stamp
    dh_testdir
    # Add here commands to compile the package.
    $(MAKE)
    #docbook-to-man debian/gentoo.sgml > gentoo.1
    touch $@
clean: clean-patched unpatch
    dh_testdir
    dh_testroot
    rm -f build-stamp configure-stamp
    # Add here commands to clean up after the build process.
    -$(MAKE) clean
    dh_clean
patch: patch-stamp
```

```
patch-stamp:
    dpatch apply-all
    dpatch call-all -a=pkg-info >patch-stamp
unpatch:
    dpatch deapply-all
    rm -rf patch-stamp debian/patched
```

W ten sposób możesz już pakować źródła za pomocą systemu dpatch używając programu dpatch-edit-patch.

```
$ dpatch-edit-patch patch 10_firstpatch
... Popraw źródła edytorem
$ exit 0
... Spróbuj zbudować pakiety przy użyciu "debuild -us -uc"
... Wyczyść źródła wywołując "debuild clean"
... Powtarzaj dpatch-edit-patch aż do zbudowania pakietów ze źródeł
$ sudo pbuilder update
$ pdebuild
$ cd /var/cache/pbuilder/result/
$ dupload -t nm_target gentoo_1.0.2-1_i386.changes
```