

Podręcznik Systemów Live

Projekt Systemów Live<debian-live@lists.debian.org>

Copyright © 2006-2014 Projekt Systemów Live

Ten program jest wolnym oprogramowaniem: możesz go rozprowadzać dalej i / lub modyfikować zgodnie z warunkami Powszechnej Licencji Publicznej GNU opublikowanej przez Free Software Foundation, według wersji 3 tej Licencji lub (według Twojego wyboru) dowolnej późniejszej wersji

Ten program jest rozpowszechniany w nadziei, że będzie użyteczny, ale BEZ JAKIEJKOLWIEK GWARANCJI; bez nawet domyślnej gwarancji PRZYDATNOŚCI HANDLOWEJ albo PRZYDATNOŚCI DO OKREŚLONYCH ZASTOSOWAŃ. Patrz GNU General Public License aby uzyskać więcej szczegółów.

Powinieneś otrzymać kopię Licencji GNU General Public License wraz z tym programem. Jeśli nie, odwiedź <<http://www.gnu.org/licenses/>>.

Pełny tekst licencji GNU General Public można znaleźć w pliku /usr/share/common-licenses/GPL-3.

Contents

O tym podręczniku	2
O tym podręczniku	3
1. O tym podręczniku	3
1.1 Dla niecierpliwych	3
1.2 Definicje	3
1.3 Autorzy	4
1.4 Wnoszenie wkładu do tego dokumentu	5
1.4.1 Nanoszenie zmian	5
1.4.2 Tłumaczenie	6
O Projekt Systemów Live	8
2. O Projekt Systemów Live	8
2.1 Motywacja	8
2.1.1 Co jest nie tak w moim dotychczasowym systemie live?	8
2.1.2 Czemu tworzyć nasz własny system live?	8
2.2 Filozofia	8
2.2.1 Tylko niezmiennione pakiety z działu Debian "main"	8
2.2.2 Bez konfiguracji pakietów systemu live	9
2.3 Kontakt	9
Użytkownik	10

Instalacja	11
3. Instalacja	11
3.1 Wymagania	11
3.2 Instalowanie live-build	11
3.2.1 Z repozytorium Debiana	11
3.2.2 Ze źródła	11
3.2.3 Ze zrzutów deweloperskich	12
3.3 Instalowanie live-boot i live-config	12
3.3.1 Z repozytorium Debiana	12
3.3.2 Ze źródła	12
3.3.3 Ze zrzutów deweloperskich	13
Podstawy	14
4. Podstawy	14
4.1 Co to jest system live?	14
4.2 Pobieranie prekompilowanych obrazów	14
4.3 Using the web live image builder	15
4.3.1 Używanie i przestrogi dotyczące Web buildera	15
4.4 Pierwsze kroki: budowanie obrazu ISO-hybrydy	15
4.5 Korzystanie z hybrydowego obrazu ISO live	16
4.5.1 Wypalanie obrazu ISO na fizycznym nośniku	16
4.5.2 Kopiowanie obrazu ISO-hybrydy na nośnik USB	16
4.5.3 Wykorzystanie przestrzeni pozostałej na nośniku USB	17
4.5.4 Uruchamianie nośnika live	17
4.6 Używanie wirtualnej maszyny do testowania	17
4.6.1 Testowanie obrazu ISO z użyciem QEMU	18
4.6.2 Testowanie obrazu ISO z użyciem VirtualBox'a	18
4.7 Budowanie i używanie obrazu HDD	18

4.8 Budowanie obrazu netboot	19	Dostosowywanie zawartości	28
4.8.1 Serwer DHCP	20	7. Opis dostosowywania	28
4.8.2 Serwer TFTP	20	7.1 Konfiguracja podczas kompilacji vs. podczas	
4.8.3 Serwer NFS	21	uruchamiania systemu	28
4.8.4 Netboot testing HowTo	21	7.2 Etapy kompilacji	28
4.8.5 Qemu	21	7.3 Uzupełnienie lb config plikami	28
4.9 Webbooting	21	7.4 Zadania dostosowywania	29
4.9.1 Getting the webboot files	22		
4.9.2 Uruchamianie obrazów webboot	22	Dostosowywanie instalacji pakietów	30
Przegląd narzędzi	23	8. Dostosowywanie instalacji pakietów	30
5. Przegląd narzędzi	23	8.1 Źródła pakietu	30
5.1 Pakiet live-build	23	8.1.1 Dystrybucja, działy archiwum i tryb	30
5.1.1 Polecenie lb config	23	8.1.2 Serwery lustrzane dystrybucji	31
5.2 Wcięcia	23	8.1.3 Serwery lustrzane dystrybucji używane	
5.2.1 Polecenie lb build	24	podczas budowania obrazu	31
5.2.2 Polecenie lb clean	24	8.1.4 Serwery lustrzane dystrybucji użyte podczas	
5.3 Pakiet live-boot	24	uruchomienia	31
5.4 Pakiet live-config	24	8.1.5 Dodatkowe repozytoria	31
		8.2 Wybieranie pakietów do instalacji	32
Zarządzanie konfiguracją	25	8.2.1 Lista pakietów	32
6. Zarządzanie konfiguracją	25	8.2.2 Używanie metapakietów	32
6.1 Radzenie sobie ze zmianami konfiguracji	25	8.2.3 Lokalna lista pakietów	33
6.1.1 Cemu używać automatycznych skryptów? Co		8.2.4 Lokalna lista pakietów binarnych	33
one robią?	25	8.2.5 Wygenerowana lista pakietów	33
6.1.2 Użyj przykładowych automatycznych skryptów	25	8.2.6 Używanie instrukcji warunkowych w listach	
6.2 Klonowanie konfiguracji opublikowanej przez Git . .	26	pakietów.	33
		8.2.7 Usuwanie pakietu podczas instalacji	34
		8.2.8 Pulpit i zadania językowe	34
		8.2.9 Rodzaj jądra i wersja	35
		8.2.10 Niestandardowe jądra	35

8.3 Instalowanie zmodyfikowanych pakietów lub pakietów innych firm	36	10.2 Ustawianie lokalizacji i języka	43
8.3.1 Używanie packages.chrootdo instalacji niestandardowych pakietów	36	10.3 Persistence	45
8.3.2 Używanie repozytorium APT aby zainstalować niestandardowe pakiety	36	10.3.1 Plik persistence.conf	46
8.3.3 Niestandardowe pakiety i APT	36	10.3.2 Używanie więcej niż jednego magazynu persistence	46
8.4 Konfigurowanie APT podczas kompilacji	37	10.4 Używanie opcji persistence z szyfrowaniem	47
8.4.1 Wybieranie apt lub aptitude	37		
8.4.2 Używanie serwera proxy z APT	37	Dostosowywanie obrazu binarnego	50
8.4.3 Podkręcanie APT celu zaoszczędzenia miejsca	37	11. Dostosowywanie obrazu binarnego	50
8.4.4 Przekazywanie opcji do apt lub aptitude	38	11.1 Programy ładujące (ang. Bootloadery)	50
8.4.5 Pinning APT	38	11.2 Metadane ISO	50
Dostosowywanie zawartości	40	Dostosowywanie Instalatora Debiana	51
9. Dostosowywanie zawartości	40	12. Dostosowywanie Instalatora Debiana	51
9.1 Uwzględnianie	40	12.1 Typy Instalatora Debiana	51
9.1.1 Lokalnie uwzględniane w chroot/live	40	12.2 Dostosowywanie Instalatora Debiana przez preseeding	52
9.1.2 Lokalnie uwzględniane dane binarne	41	12.3 Dostosowywanie zawartości Instalatora Debiana	52
9.2 Haki	41		
9.2.1 Lokalne haki chroot/live	41	Projekt	53
9.2.2 Haki podczas uruchamiania	41		
9.2.3 Lokalne haki binarne	41	Wnoszenie wkładu do tego projektu	54
9.3 Wstępne ustawienie pytań Debconfa (Preseeding)	42	13. Wnoszenie wkładu do tego projektu	54
Dostosowywanie zdarzeń podczas uruchamiania systemu	43	13.1 Wprowadzanie zmian	54
10. Dostosowywanie zdarzeń podczas uruchamiania systemu	43		
10.1 Personalizacja użytkownika live	43		

Zgłaszanie błędów	56	16.2 Wydanie Docelowe	62
14. Zgłaszanie błędów	56	16.2.1 Ostatnie Wydanie Docelowe Debiana	62
14.1 Znane problemy	56	16.2.2 Szablon obwieszczenia dla wydania docelowego	62
14.2 Przebuduj od zera	56	Repozytorium Git	64
14.3 Używaj aktualnych pakietów	56	17. Repozytorium Git	64
14.4 Zbierz potrzebne informacje	56	17.1 Obsługa wielu repozytoriów	64
14.5 Wyizoluj prawdopodobną wadę, jeśli to możliwe	57	Przykłady	66
14.6 Wybierz odpowiedni pakiet dla którego zgłaszasz błąd	57	Przykłady	67
14.6.1 W czasie budowania podczas ładowania początkowego (bootstrapping)	58	18. Przykłady	67
14.6.2 W czasie budowania podczas instalacji pakietów	58	18.1 Używanie przykładów	67
14.6.3 W czasie uruchamiania	58	18.2 Samouczek 1: Domyślny obraz	67
14.6.4 W czasie gdy system jest już uruchomiony	58	18.3 Samouczek 2: Narzędzie przeglądarka	68
14.7 Spróbuj wykonać parę kroków	58	18.4 Samouczek 3: Spersonalizowany obraz	68
14.8 Gdzie zgłaszać błędy	59	18.4.1 Pierwsza zmiana	68
Styl Kodowania	60	18.4.2 Druga zmiana	69
15. Styl Kodowania	60	18.5 Kiosk-klient serwera VNC	70
15.1 Kompatybilność	60	18.6 Bazowy obraz dla nośnika USB z 128MB pamięci.	71
15.2 Wcięcia	60	18.7 Pulpit GNOME w lokalnym języku oraz instalator	72
15.3 Zawijanie	60	Dodatek	74
15.4 Zmienne	60		
15.5 Różne	61		
Procedury	62		
16. Procedury	62		
16.1 Główne wydanie	62		

Przewodnik redakcyjny	75
19. Przewodnik redakcyjny	75
19.1 Wytyczne dla autorów	75
19.1.1 Funkcje językowe	75
19.1.2 Procedury	77
19.2 Wytyczne dla tłumaczy	78
19.2.1 Wskazówki tłumaczenia	79
SiSU Metadata, document information	80

1	Podręcznik Systemów Live
---	---------------------------------

2 **O tym podręczniku**

O tym podręczniku

1. O tym podręczniku

Podręcznik ten służy jako centralny punkt dostępu do całej dokumentacji związanej z Projekt Systemów Live a w szczególności odnosi się do oprogramowania wytworzonego przez projekt dla wydania Debian 8.0 "jessie". Aktualną wersję można zawsze znaleźć na <http://live-systems.org/>

While *live-manual* is primarily focused on helping you build a live system and not on end-user topics, an end user may find some useful information in these sections: <The Basics> covers downloading prebuilt images and preparing images to be booted from media or the network, either using the web builder or running *live-build* directly on your system. <Customizing run time behaviours> describes some options that may be specified at the boot prompt, such as selecting a keyboard layout and locale, and using persistence.

Niektóre z poleceń zawartych w tekście muszą być wykonywane z uprawnieniami superużytkownika, które mogą być uzyskane przez stanie się użytkownikiem root poprzez su lub używając sudo. Aby odróżnić te polecenia, które mogą być wykonywane przez nieuprzywilejowanego użytkownika i te wymagające praw administratora, polecenia zostały poprzedzone przez odpowiednio by \$ or # . Symbol ten nie jest częścią polecenia.

1.1 Dla niecierpliwych

Wierzmy, że wszystko w tym podręczniku jest ważne, przynajmniej dla niektórych z naszych użytkowników. Zdajemy sobie sprawę również, że zawiera on dużo materiału do sprostania. I że może chciałbyś doświadczyć szybkich postępów w używaniu

oprogramowania przed zagłębianiem się w szczegóły. Dlatego sugerujemy czytanie w następującej kolejności.

Najpierw przeczytaj rozdział, <O tym podręczniku>, od początku, kończąc na sekcji <Warunki>. Następnie przejdź do trzech ćwiczeń na początku sekcji <Przykłady> mającej na celu nauczyć Cię podstaw budowania obrazu i dostosowywania. Najpierw przeczytaj <Używanie przykładów>, następnie <Tutorial 1: Domyślny obraz>, <Tutorial 2: Narzędzie przeglądarka> i wreszcie <Tutorial 3: Spersonalizowany obraz>. Pod koniec tych tutoriali, będziesz mieć ogólny zarys tego, co można zrobić z systemami live.

Zachęcamy do powrotu i bardziej dogłębnej analizy podręcznika, być może następnym razem czytanie <Podstaw>, przejrzanie lub pominięcie <Budowanie obrazu netboot>, a skończywszy czytając <Omówienie dostosowywania> i rozdziałów, które po nim następują. W tym momencie, mamy nadzieję, że są zostaleś zainteresowany tym, co można zrobić z systemami live i zmotywowany, aby przeczytać resztę tego podręcznika, od deski do deski.

1.2 Definicje

- **System live** : System operacyjny, który można uruchomić bez instalacji na dysku twardym. Systemy live nie zmieniają lokalnego systemu(-ów) lub pliku(-ów) już zainstalowany na dysku twardym komputera, chyba że zostało to specjalnie ustawione. Systemy live są zwykle uruchamiane z nośników takich jak płyty CD, DVD lub pamięci USB. Niektóre mogą się także uruchamiać z sieci (za pośrednictwem obrazów netboot, patrz <Budowanie obrazu netboot>) i przez Internet (za pomocą parametru startowego fetch=URL, patrz <Webbooting>). webbooting).
- **Nośnik live** : W odróżnieniu od systemu live, nośnik live odnosi się do dysku CD, DVD lub pamięci USB, gdzie znajdują się

pliki binarne stworzone przez *live-build*, które są używane do uruchamiania systemu live. Szerzej, termin ten odnosi się również do każdego miejsca, w którym znajdują się te pliki binarne, które są potrzebne do uruchomienia systemu live, również takich miejsc jak miejsca dla plików startowych w sieci.

- **Projekt Systemów Live** : Projekt, który dostarcza, między innymi pakiety: *live-boot*, *live-build*, *live-config*, *live-tools* i *live-manual*.
- **System hosta** : Środowisko użyte do stworzenia systemu live.
- **System docelowy** : Środowisko użyte do uruchomienia systemu live.
- **live-boot** : Zbiór skryptów wykorzystywanych do uruchamiania systemów live.
- **live-build** : Zbiór skryptów wykorzystywanych do budowy niestandardowych systemów live.
- **live-config** : Zbiór skryptów używane do konfiguracji systemu live w czasie procesu bootowania.
- **live-tools** : Zbiór dodatkowych skryptów wykorzystywanych do wykonywania pożytecznych zadań w ramach uruchomionego systemu live.
- **live-manual** : Dokument ten jest tworzony w pakiecie o nazwie *live-manual*.
- **Debian Installer (d-i)** : Oficjalny system instalacyjny dystrybucji Debian.
- **Parametry startowe** : Parametry, które mogą być wprowadzone w wierszu bootloadera, aby wpłynąć na jądro lub *live-config*.
- **chroot** : Program *chroot*, *chroot(8)*, pozwala na uruchomienie różnych instancji środowiska GNU / Linux na jednym systemie bez ponownego uruchomienia go.

- **Obraz binarny** : Plik zawierający system live, takie jak *live-image-i386.hybrid.iso* lub *live-image-i386.img*.

- **Dystrybucja docelowa** : dystrybucja, na której opierać się będzie system żywo. To może się różnić od dystrybucji systemu hosta.

- **stable/testing/unstable** : Dystrybucja **stable** , obecnie nazwa kodowa to **wheezy** , zawiera najnowsze oficjalnie wydanie dystrybucji Debian. Dystrybucja **testing** , tymczasowo nadana nazwa kodowa to **jessie** , to obszar postępu przed następnym wydaniem **stable** . Główną zaletą korzystania z tej dystrybucji jest to, że zawiera nowsze wersje oprogramowania w stosunku do wydania **stable** . Dystrybucja **unstable** , trwale nazwana **sid** , to ta gdzie zachodzi aktywny rozwój Debian. Ogólnie rzecz biorąc, to dystrybucja prowadzona jest przez deweloperów i tych, którzy lubią życie na krawędzi. W tym podręczniku, mamy tendencję do korzystania z nazw kodowych wydań, takich jak **jessie** lub **sid** , bo głównie w tych wydaniach jest zawarte to co aktualnie zawierają narzędzia same w sobie.

1.3 Autorzy

Lista Autorów (w kolejności alfabetycznej):

- Ben Armstrong
- Brendan Sleight
- Carlos Zuferri
- Chris Lamb
- Daniel Baumann
- Franklin Piat
- Jonas Stein

- Kai Hendry
- Marco Amadori
- Mathieu Geli
- Matthias Kirschner
- Richard Nelson
- Trent W. Buck

1.4 Wnoszenie wkładu do tego dokumentu

Intencją niniejszy podręcznik jest działanie jako projekt społeczny, a więc wszelkie propozycje usprawnień i zmian są bardzo mile widziane. Proszę przejrzysz sekcję **«Przyczynianie się do projektu»** w celu uzyskania szczegółowych informacji na temat sposobu pobrania klucza do wysyłania zmian i jak wносить dobre zmiany.

1.4.1 Nanoszenie zmian

W celu dokonania zmian w podręczniku angielskim musisz edytować odpowiednie pliki w `manual/en/`, ale przed złożeniem swojego wkładu, należy przejrzeć swoją pracę. Aby wyświetlić podgląd *live-manual*, upewnij się, że pakiety potrzebne do budowy to są zainstalowane przez wykonanie:

```
# apt-get install make po4a ruby ruby-nokogiri sisu-complete
```

Możesz zbudować *live-manual* z głównego katalogu swojego zapytania GIT przez wykonanie:

38

```
$ make build
```

Ponieważ zbudowanie podręcznika we wszystkich obsługiwanych językach zajmuje trochę czasu, autorzy mogą zdecydować, że wygodniej będzie skorzystać z jednego z szybko korygujących skrótów podczas zamieszczania nowej dokumentacji, którą dodali do podręcznika angielskiego. Używanie `PROOF=1` tworzy *live-manual* w formacie HTML, ale bez posegmentowanych plików HTML, a przy użyciu `PROOF=2` tworzy się *live-manual* w formacie PDF, ale tylko sformatowane jako A4 i listowy pionowy. Dlatego korzystając z jednej z możliwości opcji `PROOF=` można zaoszczędzić sporo czasu, np.:

```
$ make build PROOF=1
```

Gdy zatwierdzasz jedno z tłumaczeń możliwe jest zbudowanie tylko dla jednego języka, przez wykonanie, np.:

```
$ make build LANGUAGES=pl
```

Jest też możliwe, aby zbudować po typie dokumentu, np:

```
$ make build FORMATS=pdf
```

Lub kombinacja obu, np:

```
$ make build LANGUAGES=pl FORMATS=html
```

Po rewizji swojej pracy i upewnieniu się, że wszystko jest w porządku, nie należy używać `make commit` chyba aktualizujesz już istniejące tłumaczenia, i w tym przypadku, nie należy mieszać zmian do instrukcji angielskiej i tłumaczeń w tej samej wysłanej zmianie, ale należy użyć osobnych zgłoszeń zmian dla każdego tłumaczenia. Zobacz sekcję [«Tłumaczenie»](#), aby uzyskać więcej szczegółów.

1.4.2 Tłumaczenie

W celu przetłumaczenia *live-manual*, wykonaj następujące kroki, w zależności od tego, czy rozpoczynasz tłumaczenie od zera czy też kontynuujesz pracę na już istniejącym tłumaczeniu:

- Rozpocznij nowe tłumaczenie od zera
 - Przetłumacz pliki **about_manual.ssi.pot**, **about_project.ssi.pot** i **index.html.in.pot** w `manual/pot/` na swój język używając swojego ulubionego edytora (np. *poedit*) i wyślij przetłumaczony plik `.po` do listy mailingowej, aby sprawdzić ich integralność. Sprawdzanie integralności *live-manual* sprawdza, nie tylko czy pliki `.po` są w 100% przetłumaczone, ale również wykrywa ewentualne błędy.
 - Po sprawdzeniu, aby włączyć nowy język w autobuild to wystarczy dodać początkowo przetłumaczone pliki do `manual/po/${LANGUAGE}/` i uruchomić `make commit`. A następnie, edytować `manual/_sisu/home/index.html` dodając nazwę języka i jego nazwy w języku angielskim w nawiasach.
- Kontynuuj pracę z już rozpoczętym tłumaczeniem
 - Jeśli Twój język docelowy został już dodany, można losowo kontynuować tłumaczenia pozostałych plików `.po` w

`manual/po/${LANGUAGE}/` za pomocą dowolnego edytora (np. *poedit*).

- Nie zapomnij, że musisz uruchomić `make commit` w celu zapewnienia, że przetłumaczone podręczniki są aktualizowane z plików `.po` i że możesz przeglądać swoje zmiany uruchamiając `make build` przed `git add .`, następnie `git commit -m "Translating..."` (Tłumaczenie...) i `git push`. Pamiętaj, że polecenie `make build` może zająć dużo czasu, możesz wybrać indywidualnie korygowane języki jak wyjaśniono w [«Zatwierdzanie zmian»](#)

Po uruchomieniu `make commit` zobaczysz jakiś przewijający się tekst. Są to przede wszystkim informacyjne komunikaty o statusie przetwarzania, a także niektóre wskazówki na temat tego, co mogłoby być zrobione, aby poprawić *live-manual*. Chyba, że widzisz błąd krytyczny, zazwyczaj w takim wypadku możesz kontynuować i wysłać swój wkład w tłumaczenie.

live-manual składa się z dwóch narzędzi, które mogą w znacznym stopniu pomóc tłumaczom znaleźć nieprzetłumaczone i zmienione ciągi znaków. Pierwszy z nich jest "make translate". Uruchamia skrypt, który mówi dokładnie ile nie przetłumaczonych ciągów jest w każdym pliku `.po`. Drugi, "make fixfuzzy", działa tylko na zmienionych ciągach znaków, ale to pomaga znaleźć je i edytować jeden po drugim.

Należy pamiętać, że nawet jeśli te narzędzia mogą być bardzo pomocne do pracy z tłumaczeniami w linii poleceń, to korzystanie z wyspecjalizowanego narzędzia jak *poedit* jest zalecanym sposobem, aby wykonać zadanie. Jest to również dobry pomysł, aby zapoznać się z dokumentacjami Debian o lokalizacjach (l10n) i tymi specyficznymi dla *live-manual*: [«Poradnik dla tłumaczy»](#).

Uwaga: Możesz użyć `make clean` aby oczyścić swoje drzewo

git przed zamieszczeniem. Ten krok nie jest obowiązkowy, dzięki plikowi .gitignore, ale dobrą praktyką jest uniknąć zatwierdzania plików odruchowo.

72	O Projekt Systemów Live	85	2.1.2 Czemu tworzyć nasz własny system live?
73	2. O Projekt Systemów Live	86	Debian to Uniwersalny system operacyjny: Debian będzie posiadał system live do przetestowania i dokładnego zaprezentowania systemu Debian z następującymi głównymi zaletami:
74	2.1 Motywacja		<ul style="list-style-type: none"> • Będzie pod-projektem Debian. 87 • Będzie odzwierciedlał stan aktualnej dystrybucji. 88 • Będzie wspierał tak wiele architektur jak to tylko możliwe. 89 • Będzie się składał tylko z niezmiennych pakietów Debian. 90 • Nie będzie zawierał żadnych pakietów, które nie są w archiwum Debian. 91 • Będzie korzystał z niezmiennego jądra Debian bez dodatkowych poprawek. 92
75	2.1.1 Co jest nie tak w moim dotychczasowym systemie live?		
76	Gdy Projekt Systemów Live został zainicjowany, było już dostępnych kilka systemów live opartych na Debianie i które teraz świetnie się spisują. Z punktu widzenia Debian większość z nich ma jednak jedną lub więcej z następujących wad:		
77	<ul style="list-style-type: none"> • Nie są to projekty Debian i dlatego nie posiadają także wsparcia ze strony Debian. 		
78	<ul style="list-style-type: none"> • Mieszą różne dystrybucje, np.: testing i unstable. 		
79	<ul style="list-style-type: none"> • Wspierają tylko i386. 		
80	<ul style="list-style-type: none"> • Modyfikują zachowanie i/lub wygląd pakietów przez obcinanie ich w celu zaoszczędzenia miejsca. 		
81	<ul style="list-style-type: none"> • Zawierają pakiety z poza archiwum Debian. 		
82	<p>_ * Są dostarczane z jądrem systemu zawierającym dodatkowe łaty, które nie są częścią Debian.</p>		
83	<ul style="list-style-type: none"> • Są duże i powolne a ze względu na swoją zwykłą wielkość, a zatem nie nadają się do zagadnień odzyskiwania. 		
84	<ul style="list-style-type: none"> • Nie są one dostępne na różnych nośnikach, np. CD, DVD, USB-stick czy obrazy netboot. 		
			2.2 Filozofia 93
			2.2.1 Tylko niezmiennione pakiety z działu Debian "main" 94
			Będziemy używać tylko pakietów z repozytorium Debian w sekcji "main". Sekcja "non-free" nie jest częścią Debian, a zatem nie może być używana do budowania oficjalnych obrazów systemu live. 95
			Nie będziemy zmieniać żadnych pakietów. Zawsze, gdy będziemy musieli coś zmienić, zrobimy to w porozumieniu z opiekunem tego pakietu w Debianie. 96
			W drodze wyjątku, nasze własne pakiety, takie jak <i>live-boot</i> , <i>live-build</i> lub <i>live-config</i> mogą być zastosowane tymczasowo z własnego repozytorium z przyczyn rozwojowych (np. do tworzenia zrzutów rozwojowych). Zostaną one przesłane do Debian na bieżąco. 97

2.2.2 Bez konfiguracji pakietów systemu live

Na tym etapie nie będziemy dostarczać lub też instalować przykładowych lub alternatywnych konfiguracji pakietów. Wszystkie pakiety będą użyte w u ich podstawowych konfiguracjach takich jakie są po normalnej instalacji Debian.

Za każdym razem, gdy potrzebujemy innej domyślnej konfiguracji, zrobimy to w porozumieniu z opiekunem pakietu w Debianie.

System do konfigurowania pakietów jest dostarczony przez użycie debconf'a; umożliwiając instalację niestandardowo skonfigurowanych pakietów w Twoim niestandardowo stworzonym obrazie systemu live. A dla <prekompilowanych obrazów live> zdecydowaliśmy, aby pozostawić pakiety w swojej domyślnej konfiguracji, chyba że będzie to absolutnie niezbędne do pracy w środowisku live. Wszędzie tam, gdzie to możliwe, wolimy dostosowywać pakiety w archiwum Debian, aby lepiej pracowały w systemie live w porównaniu do dokonywania zmian w live toolchain lub {konfiguracji obrazu prekompilowanego} # klonuj-konfiguracja-przez-git. Aby uzyskać więcej informacji, zobacz <Omówienie dostosowywania>.

2.3 Kontakt

- **Lista Mailingowa** : Podstawowym kontaktem do projektu jest lista mailingowa na <<https://lists.debian.org/debian-live/>>. Możesz napisać do listy bezpośrednio przez zadresowanie poczty do <debian-live@lists.debian.org> Archiwa listy dostępne są na <<https://lists.debian.org/debian-live/>>.
- **IRC** : Wielu użytkowników i deweloperów jest obecnych na kanale #debian-live na irc.debian.org (OFTC). Kiedy zadajesz pytanie na IRC, prosimy o cierpliwie czekać na odpowiedź. W przypadku, gdy odpowiedź nie pojawi się, napisz na listę

mailingową.

- **BTS** : {Debian Bug Tracking System} <<https://www.debian.org/Bugs/>> (BTS) zawiera informacje o błędach zgłoszonych przez użytkowników i deweloperów. Każdy błąd ma przypisany swój numer i jest przechowywany, dopóki nie zostaje on oznaczony jako rozwiązany. Aby uzyskać więcej informacji, zobacz <Zgłaszanie błędów>.

105

Użytkownik

107 Instalacja

108 3. Instalacja

109 3.1 Wymagania

110 Budowanie obrazów systemu live ma bardzo niewielkie wymagania systemowe:

- 111 • Dostęp do konta (root) super-użytkownika
- 112 • Aktualna wersja *live-build*
- 113 • Powłoka zgodna z POSIX, taka jak *bash* lub *dash*
- 114 • *debootstrap* lub *cdebootstrap*
- 115 • Linux 2.6 lub nowszy.

116 Należy pamiętać, że użycie Debiana lub dystrybucji pochodzącej od Debian nie jest wymagane - *live-build* będzie działać na prawie każdej dystrybucji spełniającej powyższe wymagania.

117 3.2 Instalowanie live-build

118 Możesz zainstalować *live-build* na wiele różnych sposobów:

- 119 • Z repozytorium Debiana
- 120 • Ze źródła
- 121 • Ze zrzutów deweloperskich

122 Jeśli używasz Debiana, zalecanym sposobem jest zainstalowanie *live-build* poprzez repozytorium Debiana.

123 3.2.1 Z repozytorium Debiana

124 Zwyczajnie zainstaluj *live-build* jak każdą inną paczkę:

125

```
# apt-get install live-build
```

52.2.2 Ze źródła

126

live-build jest opracowana z wykorzystaniem systemu kontroli wersji Git. W systemach opartych na Debianie, jest on dostarczany przez pakiet *git*. Aby sprawdzić najnowszy kod, wykonaj:

128

```
$ git clone git://live-systems.org/git/live-build.git
```

Możesz zbudować i zainstalować własną paczkę Debiana wykonując:

129

130

```
$ cd live-build
$ dpkg-buildpackage -b -uc -us
$ cd ..
```

Teraz zainstaluj którąkolwiek świeżo zbudowaną paczkę *#{.deb}*, wedle wyboru, np.

131

132

```
# dpkg -i live-build_3.0-1_all.deb
```

Możesz również zainstalować *live-build* bezpośrednio w swoim

133

systemie wykonując:

```
# make install
```

i odinstalować go wykonując:

```
# make uninstall
```

3.2.3 Ze zrzutów deweloperskich

Jeśli nie chcesz, kompilować i zainstalować *live-build* ze źródła, możesz użyć zrzutów deweloperskich. Są to automatycznie zbudowane paczki z najnowszej wersji Git i są one dostępne na <http://live-systems.org/debian/>.

3.3 Instalowanie live-boot i live-config

Uwaga: Nie musisz instalować *live-boot* lub *live-config* w systemie do tworzenia niestandardowych systemów żywych. Jednak ten sposób nie zaszkodzi i jest przydatny do celów porównawczych. Jeśli chcesz tylko przejrzeć dokumentację, możesz zainstalować pakiety *live-boot-doc* i *live-config-doc* oddzielnie.

3.3.1 Z repozytorium Debiana

Zarówno *live-boot* oraz *live-config* są dostępne w repozytorium Debiana, tak jak w [<Instalacji live-build>](#).

3.3.2 Ze źródła

Aby używać najnowszych źródeł z repozytorium GIT, użyj poniższego polecenia. Proszę upewnij się, że zapoznałeś się z warunkami wymienionymi w [<Warunkach>](#).

- Sklonuj źródła *live-boot* i *live-config*

```
$ git clone git://live-systems.org/git/live-boot.git
$ git clone git://live-systems.org/git/live-config.git
```

Poradź się podręcznikiem man pakietów *live-boot* i *live-config* aby uzyskać szczegółowe informacje na temat dostosowywania, jeżeli to jest Twój powód do budowania tych pakietów ze źródeł.

- Zbuduj pliki .deb *live-boot* i *live-config*

Musisz budować obraz albo na dystrybucji docelowej lub w środowisku chroot zawierającym platformę docelową: oznacza to, czy celem jest **jessie** to obraz należy budować na **jessie**.

Używaj osobistych konstruktorów takich jak *pbuilder* lub *sbuid*, jeżeli istnieje potrzeba zbudowania *live-boot* na dystrybucji docelowej, która różni się od systemu budowania. Na przykład, dla obrazów **jessie** live, zbuduj *live-boot* w środowisku chroot **jessie**. Jeśli dystrybucja docelowa zgadza się z dystrybucją systemu kompilacji, można wtedy zbudować bezpośrednio na wbudowanym systemie, używając *dpkg-buildpackage* (dostarczanego przez pakiet *dpkg-dev*):

```
$ cd live-boot
$ dpkg-buildpackage -b -uc -us
$ cd ../live-config
$ dpkg-buildpackage -b -uc -us
```

- Użyj mających wygenerowanych plików .deb

Przez to, że *live-boot* i *live-config* są instalowane przez system *live-build*, instalacji pakietów w systemie gospodarza nie jest wystarczająca: należy traktować wygenerowane pliki deb jak inne pakiety niestandardowe.. Ponieważ z reguły celem budowania ze źródła jest testowanie nowe rzeczy w krótkim okresie przed oficjalną premierą, poinstruuuj się **«Instalowanie zmodyfikowanych paczek innych firm»**, aby tymczasowo umieścić odpowiednie pliki w konfiguracji. W szczególności należy zauważyć, że oba pakiety są podzielone na rodzajowe części, część dokumentacji i jeden lub więcej części dodatkowych. Obejmują część rodzajową, tylko jeden back-end (część dodatkowa) dopasowana do konfiguracji i ewentualnie część dokumentacji. Zakładając, że budujesz obraz live w bieżącym katalogu i wszystkie wygenerowane paczki .deb dla pojedynczej wersji obu pakietów znajdują się w katalogu powyżej, te polecenia bash skopiują wszystkie odpowiednie pakiety, w tym domyślne dla nich back-endy:

```
$ cp ../live-boot{_, -initramfs-tools, -doc}*.deb config/packages.chroot/  
$ cp ../live-config{_, -sysvinit, -doc}*.deb config/packages.chroot/
```

3.3.3 Ze zrzutów deweloperskich

Możesz pozwolić *live-build* automatycznie skorzystać z najnowszych zrzutów deweloperskich *live-boot* i *live-live-config* przez skonfigurowanie repozytorium pakietu *live-systems.org* jako repozytorium innych firm w katalogu konfiguracyjnym *live-build*.

Podstawy

4. Podstawy

This chapter contains a brief overview of the build process and instructions for using the three most commonly used image types. The most versatile image type, `iso-hybrid`, may be used on a virtual machine, optical medium or USB portable storage device. In certain special cases, as explained later, the `hdd` type may be more suitable. The chapter includes detailed instructions for building and using a `netboot` type image, which is a bit more involved due to the setup required on the server. This is an slightly advanced topic for anyone who is not already familiar with netbooting, but it is included here because once the setup is done, it is a very convenient way to test and deploy images for booting on the local network without the hassle of dealing with image media.

The section finishes with a quick introduction to `<webbooting>` which is, perhaps, the easiest way of using different images for different purposes, switching from one to the other as needed using the internet as a means.

Throughout the chapter, we will often refer to the default filenames produced by *live-build*. If you are `<downloading a prebuilt image>` instead, the actual filenames may vary.

4.1 Co to jest system live?

A live system usually means an operating system booted on a computer from a removable medium, such as a CD-ROM or USB stick, or from a network, ready to use without any installation on the usual drive(s), with auto-configuration done at run time (see `<Terms>`).

With live systems, it's an operating system, built for one of the

supported architectures (currently amd64 and i386). It is made from the following parts:

- **Obraz jądra Linuxa** , zazwyczaj nazwany `mlinuz*`
- **Initial RAM disk image (initrd)** : a RAM disk set up for the Linux boot, containing modules possibly needed to mount the System image and some scripts to do it.
- **System image** : The operating system's filesystem image. Usually, a SquashFS compressed filesystem is used to minimize the live system image size. Note that it is read-only. So, during boot the live system will use a RAM disk and 'union' mechanism to enable writing files within the running system. However, all modifications will be lost upon shutdown unless optional persistence is used (see `<Persistence>`).
- **Bootloader** : A small piece of code crafted to boot from the chosen medium, possibly presenting a prompt or menu to allow selection of options/configuration. It loads the Linux kernel and its `initrd` to run with an associated system filesystem. Different solutions can be used, depending on the target medium and format of the filesystem containing the previously mentioned components: `isolinux` to boot from a CD or DVD in ISO9660 format, `syslinux` for HDD or USB drive booting from a VFAT partition, `extlinux` for ext2/3/4 and `btrfs` partitions, `pxelinux` for PXE netboot, GRUB for ext2/3/4 partitions, etc.

You can use *live-build* to build the system image from your specifications, set up a Linux kernel, its `initrd`, and a bootloader to run them, all in one medium-dependant format (ISO9660 image, disk image, etc.).

4.2 Pobieranie prekompilowanych obrazów

While the focus of this manual is developing and building your own

live images, you may simply wish to try one of our prebuilt images, either as an introduction to their use or instead of building your own. These images are built using our [live-images git repository](https://live-images.gitrepository) and official stable releases are published at <https://www.debian.org/CD/live/>. In addition, older and upcoming releases, and unofficial images containing non-free firmware and drivers are available at <http://live-systems.org/cdimage/release/>.

4.3 Using the web live image builder

As a service to the community, we run a web-based live image builder service at <http://live-build.debian.net/>. This site is maintained on a best effort basis. That is, although we strive to keep it up-to-date and operational at all times, and do issue notices for significant operational outages, we cannot guarantee 100% availability or fast image building, and the service may occasionally have issues that take some time to resolve. If you have problems or questions about the service, please [contact us](#), providing us with the link to your build.

4.3.1 Używanie i przestrogi dotyczące Web buildera

The web interface currently makes no provision to prevent the use of invalid combinations of options, and in particular, where changing an option would normally (i.e. using *live-build* directly) change defaults of other options listed in the web form, the web builder does not change these defaults. Most notably, if you change `--architectures` from the default `i386` to `amd64`, you must change the corresponding option `--linux-flavours` from the default `486` to `amd64`. See the `lb_config` man page for the version of *live-build* installed on the web builder for more details. The version number of *live-build* is listed at the bottom of the web builder page.

The time estimate given by the web builder is a crude estimate only and may not reflect how long your build actually takes. Nor is the estimate updated once it is displayed. Please be patient. Do not refresh the page you land on after submitting the build, as this will resubmit a new build with the same parameters. You should [contact us](#) if you don't receive notification of your build only once you are certain you've waited long enough and verified the notification e-mail did not get caught by your own e-mail spam filter.

The web builder is limited in the kinds of images it can build. This keeps it simple and efficient to use and maintain. If you would like to make customizations that are not provided for by the web interface, the rest of this manual explains how to build your own images using *live-build*.

4.4 Pierwsze kroki: budowanie obrazu ISO-hybrydy

Regardless of the image type, you will need to perform the same basic steps to build an image each time. As a first example, create a build directory, change to that directory and then execute the following sequence of *live-build* commands to create a basic ISO hybrid image containing a default live system without X.org. It is suitable for burning to CD or DVD media, and also to copy onto a USB stick.

The name of the working directory is absolutely up to you, but if you take a look at the examples used throughout *live-manual*, it is a good idea to use a name that helps you identify the image you are working with in each directory, especially if you are working or experimenting with different image types. In this case you are going to build a default system so let's call it, for example, `live-default`.

```
$ mkdir live-default && cd live-default
```

Then, run the `lb config` command. This will create a “config/” hierarchy in the current directory for use by other commands:

```
$ lb config
```

No parameters are passed to these commands, so defaults for all of their various options will be used. See <The `lb config` command> for more details.

Now that the “config/” hierarchy exists, build the image with the `lb build` command:

```
# lb build
```

This process can take a while, depending on the speed of your computer and your network connection. When it is complete, there should be a `live-image-i386.hybrid.iso` image file, ready to use, in the current directory.

Note: If you are building on an amd64 system the name of the resulting image will be `live-image-amd64.hybrid.iso`. Keep in mind this naming convention throughout the manual.

4.5 Korzystanie z hybrydowego obrazu ISO live

After either building or downloading an ISO hybrid image, which can be obtained at <<https://www.debian.org/CD/live/>>, the usual next step is to prepare your medium for booting, either CD-R(W) or DVD-R(W) optical media or a USB stick.

4.5.1 Wypalanie obrazu ISO na fizycznym nośniku

Burning an ISO image is easy. Just install *xorriso* and use it from the command-line to burn the image. For instance:

```
# apt-get install xorriso
$ xorriso -as cdrecord -v dev=/dev/sr0 blank=as_needed live-image-i386.↔
  hybrid.iso
```

4.5.2 Kopiowanie obrazu ISO-hybrydy na nośnik USB

ISO images prepared with *xorriso*, can be simply copied to a USB stick with the `cp` program or an equivalent. Plug in a USB stick with a size large enough for your image file and determine which device it is, which we hereafter refer to as `${USBSTICK}`. This is the device file of your key, such as `/dev/sdb`, not a partition, such as `/dev/sdb1`! You can find the right device name by looking in `dmesg`’s output after plugging in the stick, or better yet, `ls -l /dev/disk/by-id`.

Once you are certain you have the correct device name, use the `cp` command to copy the image to the stick. **This will definitely overwrite any previous contents on your stick!**

```
$ cp live-image-i386.hybrid.iso ${USBSTICK}
$ sync
```

Note: The `sync` command is useful to ensure that all the data, which is stored in memory by the kernel while copying the image, is written to the USB stick.

4.5.3 Wykorzystanie przestrzeni pozostałej na nośniku USB

After copying the `live-image-i386.hybrid.iso` to a USB stick, the first partition on the device will be filled up by the live system. To use the remaining free space, use a partitioning tool such as *gparted* or *parted* to create a new partition on the stick.

```
# gparted ${USBSTICK}
```

After the partition is created, where `${PARTITION}` is the name of the partition, such as `/dev/sdb2`, you have to create a filesystem on it. One possible choice would be `ext4`.

```
# mkfs.ext4 ${PARTITION}
```

Note: If you want to use the extra space with Windows, apparently that OS cannot normally access any partitions but the first. Some solutions to this problem have been discussed on our [mailing list](#), but it seems there are no easy answers.

Remember: Every time you install a new `live-image-i386.hybrid.iso` on the stick, all data on the stick will be lost because the partition table is overwritten by the contents of the image, so back up your extra partition first to restore again after updating the live image.

4.5.4 Uruchamianie nośnika live

The first time you boot your live medium, whether CD, DVD, USB key, or PXE boot, some setup in your computer's BIOS

may be needed first. Since BIOSes vary greatly in features and key bindings, we cannot get into the topic in depth here. Some BIOSes provide a key to bring up a menu of boot devices at boot time, which is the easiest way if it is available on your system. Otherwise, you need to enter the BIOS configuration menu and change the boot order to place the boot device for the live system before your normal boot device.

Once you've booted the medium, you are presented with a boot menu. If you just press enter here, the system will boot using the default entry, `Live` and default options. For more information about boot options, see the "help" entry in the menu and also the *live-boot* and *live-config* man pages found within the live system.

Assuming you've selected `Live` and booted a default desktop live image, after the boot messages scroll by, you should be automatically logged into the user account and see a desktop, ready to use. If you have booted a console-only image, such as standard or rescue flavour [prebuilt images](#), you should be automatically logged in on the console to the user account and see a shell prompt, ready to use.

4.6 Używanie wirtualnej maszyny do testowania

It can be a great time-saver for the development of live images to run them in a virtual machine (VM). This is not without its caveats:

- Running a VM requires enough RAM for both the guest OS and the host and a CPU with hardware support for virtualization is recommended.
- There are some inherent limitations to running on a VM, e.g. poor video performance, limited choice of emulated hardware.
- When developing for specific hardware, there is no substitute for

running on the hardware itself.

226

- Occasionally there are bugs that relate only to running in a VM. When in doubt, test your image directly on the hardware.

Provided you can work within these constraints, survey the available VM software and choose one that is suitable for your needs.

4.6.1 Testowanie obrazu ISO z użyciem QEMU

The most versatile VM in Debian is QEMU. If your processor has hardware support for virtualization, use the *qemu-kvm* package; the *qemu-kvm* package description briefly lists the requirements.

First, install *qemu-kvm* if your processor supports it. If not, install *qemu*, in which case the program name is *qemu* instead of *kvm* in the following examples. The *qemu-utils* package is also valuable for creating virtual disk images with *qemu-img*.

```
# apt-get install qemu-kvm qemu-utils
```

Uruchamianie obrazu ISO jest proste:

```
$ kvm -cdrom live-image-i386.hybrid.iso
```

Zobacz podręczniki man, aby uzyskać więcej szczegółów.

4.6.2 Testowanie obrazu ISO z użyciem VirtualBox'a

W celu przetestowania ISO w *VirtualBox* ie:

```
# apt-get install virtualbox virtualbox-qt virtualbox-dkms
$ virtualbox
```

Create a new virtual machine, change the storage settings to use *live-image-i386.hybrid.iso* as the CD/DVD device, and start the machine.

Note: For live systems containing X.org that you want to test with *virtualbox*, you may wish to include the VirtualBox X.org driver package, *virtualbox-guest-dkms* and *virtualbox-guest-x11*, in your *live-build* configuration. Otherwise, the resolution is limited to 800x600.

```
$ echo "virtualbox-guest-dkms virtualbox-guest-x11" >> config/package-lists<↵
/my.list.chroot
```

In order to make the dkms package work, also the kernel headers for the kernel flavour used in your image need to be installed. Instead of manually listing the correct *linux-headers* package in above created package list, the selection of the right package can be done automatically by *live-build*.

```
$ lb config --linux-packages "linux-image linux-headers"
```

4.7 Budowanie i używanie obrazu HDD

Building an HDD image is similar to an ISO hybrid one in all respects except you specify *-b hdd* and the resulting filename

is `live-image-i386.img` which cannot be burnt to optical media. It is suitable for booting from USB sticks, USB hard drives, and various other portable storage devices. Normally, an ISO hybrid image can be used for this purpose instead, but if you have a BIOS which does not handle hybrid images properly, you need an HDD image.

Note: if you created an ISO hybrid image with the previous example, you will need to clean up your working directory with the `lb clean` command (see <The `lb clean` command>):

```
# lb clean --binary
```

Run the `lb config` command as before, except this time specifying the HDD image type:

```
$ lb config -b hdd
```

A teraz zbuduj obraz używając polecenia `lb build`:

```
# lb build
```

When the build finishes, a `live-image-i386.img` file should be present in the current directory.

The generated binary image contains a VFAT partition and the syslinux bootloader, ready to be directly written on a USB device. Once again, using an HDD image is just like using an ISO hybrid one on USB. Follow the instructions in <Using an ISO hybrid live

image>, except use the filename `live-image-i386.img` instead of `live-image-i386.hybrid.iso`.

Likewise, to test an HDD image with Qemu, install *qemu* as described above in <Testing an ISO image with QEMU>. Then run `kvm` or `qemu`, depending on which version your host system needs, specifying `live-image-i386.img` as the first hard drive.

```
$ kvm -hda live-image-i386.img
```

4.8 Budowanie obrazu netboot

The following sequence of commands will create a basic netboot image containing a default live system without X.org. It is suitable for booting over the network.

Note: if you performed any previous examples, you will need to clean up your working directory with the `lb clean` command:

```
# lb clean
```

In this specific case, a `lb clean --binary` would not be enough to clean up the necessary stages. The cause for this is that in netboot setups, a different initramfs configuration needs to be used which *live-build* performs automatically when building netboot images. Since the initramfs creation belongs to the chroot stage, switching to netboot in an existing build directory means to rebuild the chroot stage too. Therefore, `lb clean` (which will remove the chroot stage, too) needs to be used.

Run the `lb config` command as follows to configure your image

for netbooting:

```
$ lb config -b netboot --net-root-path "/srv/debian-live" --net-root-server↔
"192.168.0.2"
```

In contrast with the ISO and HDD images, netbooting does not, itself, serve the filesystem image to the client, so the files must be served via NFS. Different network filesystems can be chosen through `lb config`. The `--net-root-path` and `--net-root-server` options specify the location and server, respectively, of the NFS server where the filesystem image will be located at boot time. Make sure these are set to suitable values for your network and server.

A teraz zbuduj obraz używając polecenia `lb build`:

```
# lb build
```

In a network boot, the client runs a small piece of software which usually resides on the EPROM of the Ethernet card. This program sends a DHCP request to get an IP address and information about what to do next. Typically, the next step is getting a higher level bootloader via the TFTP protocol. That could be `pxelinux`, `GRUB`, or even boot directly to an operating system like Linux.

For example, if you unpack the generated `live-image-i386.netboot.tar` archive in the `/srv/debian-live` directory, you'll find the filesystem image in `live/filesystem.squashfs` and the kernel, `initrd` and `pxelinux` bootloader in `tftpboot/`.

We must now configure three services on the server to enable netbooting: the DHCP server, the TFTP server and the NFS server.

4.8.1 Serwer DHCP

We must configure our network's DHCP server to be sure to give an IP address to the netbooting client system, and to advertise the location of the PXE bootloader.

Here is an example for inspiration, written for the ISC DHCP server `isc-dhcp-server` in the `/etc/dhcp/dhcpd.conf` configuration file:

```
# /etc/dhcp/dhcpd.conf - configuration file for isc-dhcp-server

ddns-update-style none;

option domain-name "example.org";
option domain-name-servers ns1.example.org, ns2.example.org;

default-lease-time 600;
max-lease-time 7200;

log-facility local7;

subnet 192.168.0.0 netmask 255.255.255.0 {
    range 192.168.0.1 192.168.0.254;
    filename "pxelinux.0";
    next-server 192.168.0.2;
    option subnet-mask 255.255.255.0;
    option broadcast-address 192.168.0.255;
    option routers 192.168.0.1;
}
```

4.8.2 Serwer TFTP

This serves the kernel and initial ramdisk to the system at run time.

You should install the `tftpd-hpa` package. It can serve all files contained inside a root directory, usually `/srv/tftp`. To let it serve

files inside `/srv/debian-live/tftpbboot`, run as root the following command:

```
# dpkg-reconfigure -plow tftpd-hpa
```

and fill in the new tftp server directory when being asked about it.

4.8.3 Serwer NFS

Once the guest computer has downloaded and booted a Linux kernel and loaded its initrd, it will try to mount the Live filesystem image through a NFS server.

Musisz zainstalować pakiet *nfs-kernel-server*.

Then, make the filesystem image available through NFS by adding a line like the following to `/etc/exports`:

```
/srv/debian-live *(ro,async,no_root_squash,no_subtree_check)
```

and tell the NFS server about this new export with the following command:

```
# exportfs -rv
```

Setting up these three services can be a little tricky. You might need some patience to get all of them working together. For more information, see the syslinux wiki at <http://www.syslinux.org/wiki/index.php/PXELINUX> or the Debian Installer Manual's TFTP Net Booting

section at <http://d-i.alioth.debian.org/manual/en.i386/ch04s05.html>. They might help, as their processes are very similar.

4.8.4 Netboot testing HowTo

Netboot image creation is made easy with *live-build*, but testing the images on physical machines can be really time consuming.

Aby ułatwić sobie życie możemy użyć wirtualizacji.

4.8.5 Qemu

- Zainstaluj *qemu*, *bridge-utils*, *sudo*.

Edytuj `/etc/qemu-ifup`:

```
#!/bin/sh
sudo -p "Password for $0:" /sbin/ifconfig $1 172.20.0.1
echo "Executing /etc/qemu-ifup"
echo "Bringing up $1 for bridged mode..."
sudo /sbin/ifconfig $1 0.0.0.0 promisc up
echo "Adding $1 to br0..."
sudo /usr/sbin/brctl addif br0 $1
sleep 2
```

Zainstaluj, lub zbuduj grub-floppy-netboot.

Uruchom qemu z `"-net nic,vlan=0 -net tap,vlan=0,ifname=tun0"`

4.9 Webbooting

Webbooting is a convenient way of retrieving and booting live systems using the internet as a means. The requirements for webbooting are very few. On the one hand, you need a medium with a bootloader, an initial ramdisk and a kernel. On the other

hand, a web server to store the squashfs files which contain the filesystem.

4.9.1 Getting the webboot files

As usual, you can build the images yourself or use the prebuilt files, which are available on the project's homepage at <http://live-systems.org/>. Using prebuilt images would be handy for doing initial testing until one can fine tune their own needs. If you have built a live image you will find the files needed for webbooting in the build directory under `binary/live/`. The files are called `vmlinuz`, `initrd.img` and `filesystem.squashfs`.

It is also possible to extract those files from an already existing iso image. In order to achieve that, loopback mount the image as follows:

```
# mount -o loop image.iso /mnt
```

The files are to be found under the `live/` directory. In this specific case, it would be `/mnt/live/`. This method has the disadvantage that you need to be root to be able to mount the image. However, it has the advantage that it is easily scriptable and thus, easily automatized.

But undoubtedly, the easiest way of extracting the files from an iso image and uploading it to the web server at the same time, is using the midnight commander or *mc*. If you have the *genisoimage* package installed, the two-pane file manager allows you to browse the contents of an iso file in one pane and upload the files via ftp in the other pane. Even though this method requires manual work, it does not require root privileges.

4.9.2 Uruchamianie obrazów webboot

While some users will prefer virtualization to test webbooting, we refer to real hardware here to match the following possible use case which should only be considered as an example.

In order to boot a webboot image it is enough to have the components mentioned above, i.e. `vmlinuz` and `initrd.img` in a usb stick inside a directory named `live/` and install `syslinux` as bootloader. Then boot from the usb stick and type `fetch=URL/-PATH/TO/FILE` at the boot options. *live-boot* will retrieve the squashfs file and store it into ram. This way, it is possible to use the downloaded compressed filesystem as a regular live system. For example:

```
append boot=live components fetch=http://192.168.2.50/images/webboot/↔  
filesystem.squashfs
```

Use case: You have a web server in which you have stored two squashfs files, one which contains a full desktop, like for example `gnome`, and a rescue one. If you need a graphical environment for one machine, you can plug your usb stick in and webboot the `gnome` image. If you need the rescue tools included in the second type of image, perhaps for another machine, you can webboot the rescue one.

Przegląd narzędzi

5. Przegląd narzędzi

Ten rozdział zawiera przegląd trzech głównych narzędzi stosowanych w budowie systemów live: *live-build*, *live-boot* i *live-config*.

5.1 Pakiet live-build

live-build to zbiór skryptów do budowania systemów live. Skrypty te są również określane jako “polecenia”.

Pomysłem stojącym za *live-build* jest bycie oprawą, która używa struktury katalogów jako konfiguracji, aby całkowicie zautomatyzować i dostosować wszystkie aspekty budowania obrazu live.

Wiele pojęć jest podobnych do tych używanych do budowania pakietów Debiana z użyciem *debhelper*’a:

- Skrypty posiada centralną lokalizację dla konfiguracji ich działania. Dla *debhelper*’a jest to podkatalog drzewa pakietów *debian/*. Na przykład, *dh_install* będzie szukać, spośród innych, pliku o nazwie *debian/install* do określenia, które pliki powinny zawierać się w określonym pakiecie binarnym. W taki sam sposób, *live-build* przechowuje swoją konfigurację w całości w podkatalogu *config/*.
- Skrypty są niezależne - to znaczy, że zawsze jest bezpieczne uruchomienie poszczególnych poleceń.

W przeciwieństwie do *debhelper*a, *live-build* zapewnia narzędzia do generowania szkieletu katalogów konfiguracyjnych. Może to być uznane za podobne do narzędzi takich jak *dh-make*. Aby uzyskać więcej informacji na temat tych narzędzi, kontynuuj

czytanie, ponieważ pozostała część tego rozdziału omawia cztery najważniejsze polecenia. Należy zauważyć, że głównym wrapperem dla polecenia *live-build* jest *lb*.

- **lb config** : Responsible for initializing a Live system configuration directory. See <[The lb config command](#)> for more information. 306
- **lb build** : Odpowiedzialny za rozpoczęcie kompilacji systemu live. Zobacz <[polecenie lb build](#)> aby uzyskać więcej informacji. 307
- **lb clean** : Odpowiedzialny za czyszczenie kompilacji systemu live. Zobacz <[polecenie lb clean](#)> aby uzyskać więcej informacji. 308

5.1.1 Polecenie lb config 309

Jak omówiono w <[live-build](#)>, skrypty, które składają się na *live-build* czytają swoją konfigurację przy użyciu polecenia *source* z katalogu o nazwie *config/*. Budowanie tego katalogu ręcznie byłoby czasochłonne i podatne na błędy, polecenie *lb config* może być używane do tworzenia początkowej konfiguracji drzewa katalogów.

Issuing *lb config* without any arguments creates the *config/* subdirectory which is populated with some default settings in configuration files, and two skeleton trees named *auto/* and *local/*.

```
$ lb config
[2014-04-25 17:14:34] lb config
P: Updating config tree for a debian/wheezy/i386 system
```

5.2 Wcięcia 313

Normalnie, pewnie będziesz chciał określić niektóre opcje. Na 314

przykład, aby określić, jakiego menadżera pakierów chcesz użyć podczas budowania obrazu:

```
$ lb config --apt aptitude
```

Jest możliwe ustalenie wielu opcji, takich jak:

```
$ lb config --binary-images netboot --bootappend-live "boot=live components↵
hostname=live-host username=live-user" ...
```

Pełna lista opcji dostępna jest w podręczniku man pakietu `lb_config`.

5.2.1 Polecenie `lb build`

Polecenie `lb build` czyta konfigurację z katalogu `config/`. A następnie uruchamia polecenia niższego poziomu potrzebne do budowy Twojego systemu live.

5.2.2 Polecenie `lb clean`

Zadaniem polecenia `lb clean`, jest to aby usunąć różne części kompilacji tak aby można było zacząć od czystego stanu. Domyślnie etapy `chroot`, `binary` and `source` są sprzątane, ale `cache` pozostaje nienaruszone. Ponadto, tylko poszczególne etapy mogą być oczyszczane. Na przykład, jeśli zostały wprowadzone zmiany, które wpływają tylko na etap binarny, należy użyć `lb clean --binary` przed budowaniem nowych plików binarnych. Jeśli zmiany unieważniają proces `bootstrap` i/lub zmieniają `cache` pakietów, np. po zmianie opcji `--mode`, `--architecture`, lub

`--bootstrap`, trzeba użyć `lb clean --purge`. Zobacz podręcznik man pakietu `lb_clean` aby uzyskać listę wszystkich opcji.

5.3 Pakiet `live-boot`

live-boot to zbiór skryptów zapewniających haki do *initramfs-tools*, wykorzystywane do wytwarzania plików `initramfs`, które są w stanie uruchomić system live, takich jak te stworzone przez *live-build*. Obejmuje to obrazy ISO systemów live, archiwa `netboot` i obrazów dysku USB.

W czasie rozruchu będzie szukać nośników tylko do odczytu zawierających katalog `/live/`, gdzie jest przechowywany system plików (często skompresowany obraz systemu plików jak `SquashFS`). Jeśli znajdzie takowy, stworzy zapisywalne środowisko, stosując `aufs`, dla systemów takich jak `Debian`, aby z niego wystartować.

Więcej informacji na temat początkowych plików `ramfs` w `Debianie` można znaleźć w Podręczniku `Debian Linux Kernel` na <http://kernel-handbook.alioth.debian.org/> w rozdziale `initramfs`.

5.4 Pakiet `live-config`

live-config zawiera skrypty, które są uruchamiane przy starcie systemu live po *live-boot*, tak aby automatycznie skonfigurować system live. Obsługuje on takie zadania jak ustawienie nazwy hosta, lokalizacji i strefy czasowej, tworzenie użytkownika live, zatrzymywanie zadań `crona` i autologowanie użytkownika live.

Zarządzanie konfiguracją

6. Zarządzanie konfiguracją

Ten rozdział wyjaśnia, jak zarządzać konfiguracją live od początku jej tworzenia, przez kolejne zmiany i kolejne wersje oprogramowania *live-build* i obrazu live.

6.1 Radzenie sobie ze zmianami konfiguracji

Konfiguracje live rzadko są idealne na przy pierwszej próbie. Powinno się dodać opcje `lb config` z linii poleceń do wykonania pojedynczej kompilacji, ale bardziej typowe jest sprawdzać te opcje i kompilować ponownie aż do uzyskania satysfakcji. Aby poradzić sobie z tymi zmianami, potrzeba automatycznych skryptów, które zapewnią, że konfiguracja przechowywana jest w stanie spójnym.

6.1.1 Czemu używać automatycznych skryptów? Co one robią?

Polecenie `lb config` zapisuje opcje, które są wprowadzane do plików w `config/*#` oraz wielu innym opcją przypisuje wartości domyślne. Jeśli uruchomisz `#{lb config}` ponownie, nie skasuje to żadnych opcji, która została zapisana na podstawie początkowych opcji. Tak więc, na przykład, jeśli by uruchomić `lb config` ponownie z nową wartością dla `--binary-images`, wszelkie opcje zależne, które zostały przypisane jako domyślne dla poprzedniej dystrybucji mogą już nie współgrać z nowym ustawieniem. Pliki te nie są przeznaczone do odczytu lub edycji. Przechowują one wartości dla ponad stu opcji, więc nikt, nie mówiąc już o robieniu tego w pojedynkę, nie będzie mógł zobaczyć, które z tych opcji są faktycznie przypisane. I wreszcie,

po uruchomieniu `lb config`, a następnie uaktualnieniu *live-build* a zdarza się, że zmieniają się nazwy opcji, `config/*` nadal będzie zawierać zmienne nazwane po staremu, które nie są już aktualne.

Z tych wszystkich powodów, skrypty `auto/*` czynią Twoje życie łatwiejszym. Są proste wrappery do poleceń `lb config`, `lb build` i `lb clean`, które są zaprojektowane, aby pomóc w zarządzaniu konfiguracją. Skrypt `auto/config` przechowuje toż samo polecenie `lb config` ze wszystkimi pożądanymi opcjami, skrypt `auto/clean` usuwa pliki zawierające wartości zmiennych konfiguracyjnych a skrypt `auto/build` zachowuje log `build.log` każdej kompilacji. Każdy z tych scenariuszy jest uruchamiany automatycznie przy każdym uruchomieniu odpowiedniego polecenia `lb`. Korzystając z tych skryptów, konfiguracja jest bardziej czytelna i jest przechowywana w sposób wewnętrznie spójny z jednej wersji do następnej. Ponadto, będzie o wiele łatwiej zidentyfikować opcje, które należy zmienić po uaktualnieniu *live-build* i po przeczytaniu dokumentacji aktualizacji.

6.1.2 Użyj przykładowych automatycznych skryptów

Dla Twojej wygody, *live-build* jest dostarczany z przykładowymi skryptami powłoki do automatycznego kopiowania i edycji. Rozpocznij nową, domyślną konfigurację, a następnie skopiuj do niej przykłady:

```
$ mkdir mójlive && cd mójlive && lb config
$ mkdir auto
$ cp /usr/share/doc/live-build/examples/auto/* auto/
```

Edytuj `auto/config`, dodając wszelkie opcje, jakie uważasz. Przykładowo:


```
#!/bin/sh
lb config noauto \
  --architectures i386 \
  --linux-flavours 686-pae \
  --binary-images hdd \
  --mirror-bootstrap http://ftp.ch.debian.org/debian/ \
  --mirror-binary http://ftp.ch.debian.org/debian/ \
  "${@}"
```

Teraz, za każdym razem kiedy korzystasz z `lb config`, `auto/config` skasuje konfigurację w oparciu o te opcje. Gdy chcesz wprowadzić zmiany do nich, należy edytować opcje w tym pliku zamiast przekazywać je do `lb config`. Podczas korzystania z `lb clean`, `auto/clean` oczyści pliki w `config/*` wraz z innymi produktami kompilacji. I wreszcie, kiedy używasz `lb build`, log kompilacji zostanie zapisany przez `auto/build` w `build.log`.

Uwaga: Specjalny parametr `noauto` jest użyty tutaj, aby powstrzymać kolejne zapytania do `auto/config`, zapobiegając w ten sposób nieskończonej rekurencji. Upewnij się, że przypadkowo nie został usunięty podczas dokonywania zmiany. Również należy zadbać o to aby podczas dzielenia polecenia `lb config` na wiele linii dla czytelności, jak pokazano w powyższym przykładzie, nie zapomnisz odwrotnego ukośnika (na końcu każdej linii, która kontynuuje polecenie w następnej linijce).

6.2 Klonowanie konfiguracji opublikowanej przez Git

Użyj opcji `lb config --config` aby sklonować repozytorium Git, który zawiera konfigurację systemu live. Jeśli chcesz oprzeć swoją konfigurację na jednej dostarczanej przez Projekt Systemów Live, odwiedź <http://live-systems.org/gitweb/> i szukaj repozytorium o

nazwie `live-images` (obrazy live) w kategorii Packages (pakiety). To repozytorium zawiera konfiguracje dla **prekompilowanych obrazów** systemów live.

Na przykład, aby zbudować obraz ratunkowy, użyj repozytorium `live-images` w następujący sposób:

```
$ mkdir live-images && cd live-images
$ lb config --config git://live-systems.org/git/live-images.git
$ cd images/rescue
```

Edytuj `auto/config` i wszelkie inne rzeczy, których wymagasz do własnych potrzeb w drzewie katalogów `config`. Na przykład, nie oficjalne prekompilowane obrazy tworzy się dodając po prostu `--archive-areas "main contrib non-free"`.

Opcjonalnie można zdefiniować skrót w konfiguracji Git przez dodanie następujących opcji do `$(HOME)/.gitconfig`:

```
[url "git://live-systems.org/git/"]
  insteadOf = lso:
```

Dzięki temu można skorzystać z `lso:` wszędzie, gdzie trzeba podać adres repozytorium git `live-systems.org`. Jeśli również opuścisz opcjonalny przyrostek `.git`, rozpoczynając nowy obraz przy użyciu tej konfiguracji jest to tak proste:

```
$ lb config --config lso:live-images
```

Klonowanie całego repozytorium `live-images` (obrazów live) sściąga konfigurację używaną dla kilku różnych obrazów. Jeśli

masz ochotę na budowę innego obrazu po zakończeniu pracy z pierwszym, przejdź do innego katalogu i ponownie w miarę potrzeb dokonaj zmian.

354

W każdym przypadku należy pamiętać, że za każdym razem trzeba będzie budować obraz jako super-użytkownik: `lb build`

Dostosowywanie zawartości

7. Opis dostosowywania

Ten rozdział zawiera przegląd różnych sposobów, w jaki można dostosować system live.

7.1 Konfiguracja podczas kompilacji vs. podczas uruchamiania systemu

Opcje konfiguracji systemu live są podzielone na opcje w czasie budowania, które są stosowane w czasie kompilacji i na opcje w czasie rozruchu, które są stosowane podczas uruchamiania systemu. Opcje podczas uruchamiania są podzielone na te występujące wcześniej podczas uruchamiania, zastosowane przez *live-boot*, i na te, występujące później, zastosowane przez *live-config*. Każdy parametr rozruchu może zostać zmodyfikowany przez użytkownika poprzez ustalenie go podczas startu. Obraz może być również zbudowany z domyślnymi parametrami startowymi, dzięki czemu użytkownicy mogą normalnie tylko uruchomić bezpośrednio systemu live bez podawania żadnych parametrów, gdy wszystkie opcje domyślne są odpowiednie. W szczególności argument `lb --bootappend-live` składa się z wszelkich opcji wiersza poleceń domyślnych dla jądra systemu live, takich jak trwałość (ang. persistence), układ klawiatury, lub strefa czasowa. Zobacz [«Dostosowywanie lokalizacji i języka»](#), dla przykładów.

Build-time configuration options are described in the `lb config` man page. Boot-time options are described in the man pages for *live-boot* and *live-config*. Although the *live-boot* and *live-config* packages are installed within the live system you are building, it is recommended that you also install them on your build system for easy reference when you are working on your configuration. It

is safe to do so, as none of the scripts contained within them are executed unless the system is configured as a live system.

7.2 Etapy kompilacji

Proces kompilacji jest podzielony na etapy, w każdym z nich z zastosowanymi w kolejności różnymi dostosowaniami. Pierwszym etapem do uruchomienia jest etap **bootstrap**. Jest to wstępna faza wypełniania katalogu chroot pakietami aby stworzyć kadłub systemu Debian. Następnym etapem jest **chroot**, który kończy budowę katalogu chroot, wypełniania go wszystkimi pakietami wymienionymi w konfiguracji, wraz z innymi materiałami. Najwięcej dostosowywania zawartości odbywa się w tym etapie. Ostatnim etapem przygotowania obrazu live jest etap **binarny** (ang. binary), który tworzy możliwy do uruchomienia obraz, używając zawartości katalogu chroot do budowy głównego systemu plików w systemie live, a tym instalatora i wszelkich innych dodatkowych materiałów na nośniku docelowym poza system plików na systemie live. Po skompilowaniu obrazu live, jeśli włączono, archiwum źródłowe tarball jest budowane podczas etapu **source** (ang. źródło).

W każdym z tych etapów istnieje szczególna sekwencja, w której stosuje się polecenia. Są one usytuowane w taki sposób, aby zapewnić modyfikacjom bycie ułożonym w rozsądny sposób. Na przykład, w etapie **chroot**, prekonfiguracja (ang. preseeding) jest stosowana, zanim zostaną zainstalowane jakiejkolwiek pakiety, pakiety są instalowane zanim jakiejkolwiek lokalnie zawarte pliki zostaną kopiowane, a haki są wprowadzane później, gdy wszystkie materiały są już na miejscu.

7.3 Uzupełnienie lb config plikami

Mimo, że `lb config` tworzy konfigurację katalogów w `config/`,

aby osiągnąć swoje cele, może być konieczne udostępnienie dodatkowych plików w podkatalogach `config/`. W zależności od tego, gdzie pliki są przechowywane w konfiguracji, mogą być skopiowane do systemu plików systemu live lub do binarnego obrazu systemu plików, lub mogą zostać zapewnione konfiguracje w czasie budowy systemu, które byłoby kłopotliwie do przekazania jako opcje wiersza polecenia. Można zawrzeć rzeczy takie jak niestandardowe listy pakietów, niestandardowa grafika lub inny skrypt do uruchomienia zarówno w czasie kompilacji jak i w czasie startu systemu, zwiększając już znaczną elastyczność debian-live swoim własnym kodem.

366

7.4 Zadania dostosowywania

367

Kolejne rozdziały są podzielone na rodzaje zadań dostosowywania, który użytkownicy zazwyczaj wykonują: <Dostosowywanie instalacji pakietu>, <Dostosowywanie zawartości> i <Dostosowywanie ustawień regionalnych i języka> obejmują tylko niektóre z rzeczy, które możesz zrobić.

Dostosowywanie instalacji pakietów

8. Dostosowywanie instalacji pakietów

Perhaps the most basic customization of a live system is the selection of packages to be included in the image. This chapter guides you through the various build-time options to customize *live-build*'s installation of packages. The broadest choices influencing which packages are available to install in the image are the distribution and archive areas. To ensure decent download speeds, you should choose a nearby distribution mirror. You can also add your own repositories for backports, experimental or custom packages, or include packages directly as files. You can define lists of packages, including metapackages which will install many related packages at once, such as packages for a particular desktop or language. Finally, a number of options give some control over *apt*, or if you prefer, *aptitude*, at build time when packages are installed. You may find these handy if you use a proxy, want to disable installation of recommended packages to save space, or need to control which versions of packages are installed via APT pinning, to name a few possibilities.

8.1 Źródła pakietu

8.1.1 Dystrybucja, działy archiwum i tryb

The distribution you choose has the broadest impact on which packages are available to include in your live image. Specify the codename, which defaults to **jessie** for the **jessie** version of *live-build*. Any current distribution carried in the archive may be specified by its codename here. (See [Terms](#) for more details.) The `--distribution` option not only influences the source of packages within the archive, but also instructs *live-build* to behave as needed

to build each supported distribution. For example, to build against the **unstable** release, **sid**, specify:

```
$ lb config --distribution sid
```

Within the distribution archive, archive areas are major divisions of the archive. In Debian, these are **main**, **contrib** and **non-free**. Only **main** contains software that is part of the Debian distribution, hence that is the default. One or more values may be specified, e.g.

```
$ lb config --archive-areas "main contrib non-free"
```

Experimental support is available for some Debian derivatives through a `--mode` option. By default, this option is set to **debian** only if you are building on a Debian or on an unknown system. If `lb config` is invoked on any of the supported derivatives, it will default to create an image of that derivative. If `lb config` is run in e.g. **ubuntu** mode, the distribution names and archive areas for the specified derivative are supported instead of the ones for Debian. The mode also modifies *live-build* behaviour to suit the derivatives.

Note: The projects for whom these modes were added are primarily responsible for supporting users of these options. The Projekt Systemów Live, in turn, provides development support on a best-effort basis only, based on feedback from the derivative projects as we do not develop or support these derivatives ourselves.

8.1.2 Serwery lustrzane dystrybucji

The Debian archive is replicated across a large network of mirrors around the world so that people in each region can choose a nearby mirror for best download speed. Each of the `--mirror-*` options governs which distribution mirror is used at various stages of the build. Recall from *Stages of the build* that the **bootstrap** stage is when the chroot is initially populated by *debootstrap* with a minimal system, and the **chroot** stage is when the chroot used to construct the live system's filesystem is built. Thus, the corresponding mirror switches are used for those stages, and later, in the **binary** stage, the `--mirror-binary` and `--mirror-binary-security` values are used, superseding any mirrors used in an earlier stage.

8.1.3 Serwery lustrzane dystrybucji używane podczas budowania obrazu

To set the distribution mirrors used at build time to point at a local mirror, it is sufficient to set `--mirror-bootstrap`, `--mirror-chroot-security` and `--mirror-chroot-backports` as follows.

```
$ lb config --mirror-bootstrap http://localhost/debian/ \
  --mirror-chroot-security http://localhost/debian-security/ \
  --mirror-chroot-backports http://localhost/debian-backports/
```

Serwer lustrzany chroot, określony przez `--mirror-chroot`, domyślnie do wartości `--mirror-bootstrap`.

8.1.4 Serwery lustrzane dystrybucji użyte podczas uruchomienia

The `--mirror-binary*` options govern the distribution mirrors

placed in the binary image. These may be used to install additional packages while running the live system. The defaults employ `http.debian.net`, a service that chooses a geographically close mirror based, among other things, on the user's IP family and the availability of the mirrors. This is a suitable choice when you cannot predict which mirror will be best for all of your users. Or you may specify your own values as shown in the example below. An image built from this configuration would only be suitable for users on a network where "mirror" is reachable.

```
$ lb config --mirror-binary http://mirror/debian/ \
  --mirror-binary-security http://mirror/debian-security/ \
  --mirror-binary-backports http://mirror/debian-backports/
```

8.1.5 Dodatkowe repozytoria

You may add more repositories, broadening your package choices beyond what is available in your target distribution. These may be, for example, for backports, experimental or custom packages. To configure additional repositories, create `config/archives/your-repository.list.chroot`, and/or `config/archives/your-repository.list.binary` files. As with the `--mirror-*` options, these govern the repositories used in the **chroot** stage when building the image, and in the **binary** stage, i.e. for use when running the live system.

For example, `config/archives/live.list.chroot` allows you to install packages from the debian-live snapshot repository at live system build time.

```
deb http://live-systems.org/ sid-snapshots main contrib non-free
```

If you add the same line to `config/archives/live.list.binary`, the repository will be added to your live system's `/etc/apt/sources.list.d/` directory.

Jeżeli takie pliki istnieją to zostaną wybrane automatycznie.

Należy również umieścić klucz GPG używany do podpisywania repozytorium w `config/archives/twój-klucz-repozytorium.-key{binary, chroot}`.

Should you need custom APT pinning, such APT preferences snippets can be placed in `config/archives/your-repository.-pref.{binary, chroot}` files and will be automatically added to your live system's `/etc/apt/preferences.d/` directory.

8.2 Wybieranie pakietów do instalacji

There are a number of ways to choose which packages *live-build* will install in your image, covering a variety of different needs. You can simply name individual packages to install in a package list. You can also use metapackages in those lists, or select them using package control file fields. And finally, you may place package files in your `config/` tree, which is well suited to testing of new or experimental packages before they are available from a repository.

8.2.1 Lista pakietów

Listy pakietów są skutecznym sposobem wyrażenia, które pakiety powinny zostać zainstalowane. Składnia list obsługuje instrukcje warunkowe, które ułatwiają budowanie list i dostosowywanie ich do wykorzystania w wielu konfiguracjach. Nazwy pakietów mogą być także wstrzykiwane do listy za pomocą pomocników powłoki w czasie kompilacji.

Note: The behaviour of *live-build* when specifying a package that does not exist is determined by your choice of APT utility. See [«Choosing apt or aptitude»](#) for more details.

8.2.2 Używanie metapakietów

Najprostszym sposobem, aby wypełnić swoją listę pakietów jest użycie metapakietu zadań dostarczanych przez dystrybucję. Na przykład:

```
$ lb config
$ echo task-gnome-desktop > config/package-lists/desktop.list.chroot
```

This supercedes the older predefined list method supported in *live-build* 2.x. Unlike predefined lists, task metapackages are not specific to the Live System project. Instead, they are maintained by specialist working groups within the distribution and therefore reflect the consensus of each group about which packages best serve the needs of the intended users. They also cover a much broader range of use cases than the predefined lists they replace.

All task metapackages are prefixed `task-`, so a quick way to determine which are available (though it may contain a handful of false hits that match the name but aren't metapackages) is to match on the package name with:

```
$ apt-cache search --names-only ^task-
```

Oprócz tych, znajdziesz jeszcze inne metapakiety do różnych celów. Niektóre są podzbiorami szerszych pakietów zadań, jak

gnome-core, podczas gdy inne są indywidualne specjalistyczne części czystego Debiana mieszanki, takie jak metapakiety `education-*`. Aby wyświetlić wszystkie metapakiety w archiwum, należy zainstalować pakiet `debtags` i wyświetlić wszystkie pakiety z tagiem `role::metapackage` w następujący sposób:

```
$ debtags search role::metapackage
```

8.2.3 Lokalna lista pakietów

Whether you list metapackages, individual packages, or a combination of both, all local package lists are stored in `config/-package-lists/`. Since more than one list can be used, this lends itself well to modular designs. For example, you may decide to devote one list to a particular choice of desktop, another to a collection of related packages that might as easily be used on top of a different desktop. This allows you to experiment with different combinations of sets of packages with a minimum of fuss, sharing common lists between different live image projects.

Package lists that exist in this directory need to have a `.list` suffix in order to be processed, and then an additional stage suffix, `.chroot` or `.binary` to indicate which stage the list is for.

Note: If you don't specify the stage suffix, the list will be used for both stages. Normally, you want to specify `.list.chroot` so that the packages will only be installed in the live filesystem and not have an extra copy of the `.deb` placed on the medium.

8.2.4 Lokalna lista pakietów binarnych

To make a binary stage list, place a file suffixed with `.list.binary`

in `config/package-lists/`. These packages are not installed in the live filesystem, but are included on the live medium under `pool/`. You would typically use such a list with one of the non-live installer variants. As mentioned above, if you want this list to be the same as your `chroot` stage list, simply use the `.list` suffix by itself.

8.2.5 Wygenerowana lista pakietów

It sometimes happens that the best way to compose a list is to generate it with a script. Any line starting with an exclamation point indicates a command to be executed within the `chroot` when the image is built. For example, one might include the line `! grep-aptavail -n -sPackage -FPriority standard |sort` in a package list to produce a sorted list of available packages with `Priority: standard`.

In fact, selecting packages with the `grep-aptavail` command (from the `dctrl-tools` package) is so useful that `live-build` provides a `Packages` helper script as a convenience. This script takes two arguments: `field` and `pattern`. Thus, you can create a list with the following contents:

```
$ lb config
$ echo '! Packages Priority standard' > config/package-lists/standard.list.<↵
chroot
```

8.2.6 Używanie instrukcji warunkowych w listach pakietów.

Any of the *live-build* configuration variables stored in `config/*` (minus the `LB_` prefix) may be used in conditional statements in package lists. Generally, this means any `lb config` option

uppercased and with dashes changed to underscores. But in practice, it is only the ones that influence package selection that make sense, such as `DISTRIBUTION`, `ARCHITECTURES` or `ARCHIVE_AREAS`.

Na przykład, aby zainstalować `ia32-libs` jeżeli wybrano `--architectures amd64`:

```
#if ARCHITECTURES amd64
ia32-libs
#endif
```

Można przetestować dowolny szereg wartości, na przykład zainstalować `memtest86+`, jeżeli ustalono `--architectures i386` lub `--architectures amd64`:

```
#if ARCHITECTURES i386 amd64
memtest86+
#endif
```

You may also test against variables that may contain more than one value, e.g. to install `vrms` if either `contrib` or `non-free` is specified via `--archive-areas`:

```
#if ARCHIVE_AREAS contrib non-free
vrms
#endif
```

Zagnieżdżanie instrukcji warunkowych jest nie obsługiwane.

8.2.7 Usuwanie pakietu podczas instalacji

You can list packages in files with `.list.chroot_live` and `.list.chroot_install` suffixes inside the `config/package-lists` directory. If both a live and an install list exist, the packages in the `.list.chroot_live` list are removed with a hook after the installation (if the user uses the installer). The packages in the `.list.chroot_install` list are present both in the live system and in the installed system. This is a special tweak for the installer and may be useful if you have `--debian-installer live` set in your config, and wish to remove live system-specific packages at install time.

8.2.8 Pulpit i zadania językowe

Desktop and language tasks are special cases that need some extra planning and configuration. Live images are different from Debian Installer images in this respect. In the Debian Installer, if the medium was prepared for a particular desktop environment flavour, the corresponding task will be automatically installed. Thus, there are internal `gnome-desktop`, `kde-desktop`, `lxde-desktop` and `xfce-desktop` tasks, none of which are offered in `tasksel`'s menu. Likewise, there are no menu entries for tasks for languages, but the user's language choice during the install influences the selection of corresponding language tasks.

When developing a desktop live image, the image typically boots directly to a working desktop, the choices of both desktop and default language having been made at build time, not at run time as in the case of the Debian Installer. That's not to say that a live image couldn't be built to support multiple desktops or multiple languages and offer the user a choice, but that is not *live-build*'s default behaviour.

Because there is no provision made automatically for language tasks, which include such things as language-specific fonts and input-method packages, if you want them, you need to specify them in your configuration. For example, a GNOME desktop image containing support for German might include these task metapackages:

```
$ lb config
$ echo "task-gnome-desktop task-laptop" >> config/package-lists/my.list.chroot
$ echo "task-german task-german-desktop task-german-gnome-desktop" >> config/package-lists/my.list.chroot
```

8.2.9 Rodzaj jądra i wersja

One or more kernel flavours will be included in your image by default, depending on the architecture. You can choose different flavours via the `--linux-flavours` option. Each flavour is suffixed to the default stub `linux-image` to form each metapackage name which in turn depends on an exact kernel package to be included in your image.

Thus by default, an amd64 architecture image will include the `linux-image-amd64` flavour metapackage, and an i386 architecture image will include the `linux-image-486` and `linux-image-686-pae` metapackages. At time of writing, these packages depend on `linux-image-3.2.0-4-amd64`, `linux-image-3.2.0-4-486` and `linux-image-3.2.0-4-686-pae`, respectively.

When more than one kernel package version is available in your configured archives, you can specify a different kernel package name stub with the `--linux-packages` option. For example, supposing you are building an amd64 architecture image and add

the experimental archive for testing purposes so you can install the `linux-image-3.7-trunk-amd64` kernel. You would configure that image as follows:

```
$ lb config --linux-packages linux-image-3.7-trunk
$ echo "deb http://ftp.debian.org/debian/ experimental main" > config/archives/experimental.list.chroot
```

8.2.10 Niestandardowe jądra

You can build and include your own custom kernels, so long as they are integrated within the Debian package management system. The *live-build* system does not support kernels not built as `.deb` packages.

The proper and recommended way to deploy your own kernel packages is to follow the instructions in the `kernel-handbook`. Remember to modify the ABI and flavour suffixes appropriately, then include a complete build of the `linux` and matching `linux-latest` packages in your repository.

If you opt to build the kernel packages without the matching metapackages, you need to specify an appropriate `--linux-packages` stub as discussed in [Kernel flavour and version](#). As we explain in [Installing modified or third-party packages](#), it is best if you include your custom kernel packages in your own repository, though the alternatives discussed in that section work as well.

It is beyond the scope of this document to give advice on how to customize your kernel. However, you must at least ensure your configuration satisfies these minimum requirements:

- Use an initial ramdisk.

- Include the union filesystem module (i.e. usually aufs).
- Include any other filesystem modules required by your configuration (i.e. usually squashfs).

8.3 Instalowanie zmodyfikowanych pakietów lub pakietów innych firm

While it is against the philosophy of a live system, it may sometimes be necessary to build a live system with modified versions of packages that are in the Debian repository. This may be to modify or support additional features, languages and branding, or even to remove elements of existing packages that are undesirable. Similarly, “third-party” packages may be used to add bespoke and/or proprietary functionality.

This section does not cover advice regarding building or maintaining modified packages. Joachim Breitner’s ‘How to fork privately’ method from <http://www.joachim-breitner.de/blog/archives/282-How-to-fork-privately.html> may be of interest, however. The creation of bespoke packages is covered in the Debian New Maintainers’ Guide at <https://www.debian.org/doc/maint-guide/> and elsewhere.

Istnieją dwa sposoby instalacji niestandardowych pakietów:

- `packages.chroot`
- Używanie niestandardowego repozytorium APT

Using `packages.chroot` is simpler to achieve and useful for “one-off” customizations but has a number of drawbacks, while using a custom APT repository is more time-consuming to set up.

8.3.1 Używanie `packages.chroot` do instalacji niestandardowych pakietów

To install a custom package, simply copy it to the `config/-`

`packages.chroot/` directory. Packages that are inside this directory will be automatically installed into the live system during build - you do not need to specify them elsewhere.

Packages **must** be named in the prescribed way. One simple way to do this is to use `dpkg-name`.

Korzystanie z `packages.chroot` do instalacji niestandardowych pakietów ma wady:

- Nie jest możliwe użycie bezpiecznego APT.
- You must install all appropriate packages in the `config/packages.chroot/` directory.
- It does not lend itself to storing live system configurations in revision control.

8.3.2 Używanie repozytorium APT aby zainstalować niestandardowe pakiety

Unlike using `packages.chroot`, when using a custom APT repository you must ensure that you specify the packages elsewhere. See [Choosing packages to install](#) for details.

While it may seem unnecessary effort to create an APT repository to install custom packages, the infrastructure can be easily re-used at a later date to offer updates of the modified packages.

8.3.3 Niestandardowe pakiety i APT

live-build uses APT to install all packages into the live system so will therefore inherit behaviours from this program. One relevant example is that (assuming a default configuration) given a package available in two different repositories with different

version numbers, APT will elect to install the package with the higher version number.

Because of this, you may wish to increment the version number in your custom packages' `debian/changelog` files to ensure that your modified version is installed over one in the official Debian repositories. This may also be achieved by altering the live system's APT pinning preferences - see [APT pinning](#) for more information.

8.4 Konfigurowanie APT podczas kompilacji

You can configure APT through a number of options applied only at build time. (APT configuration used in the running live system may be configured in the normal way for live system contents, that is, by including the appropriate configurations through `config/includes.chroot/`.) For a complete list, look for options starting with `apt` in the `lb_config` man page.

8.4.1 Wybieranie `apt` lub `aptitude`

You can elect to use either *apt* or *aptitude* when installing packages at build time. Which utility is used is governed by the `--apt` argument to `lb config`. Choose the method implementing the preferred behaviour for package installation, the notable difference being how missing packages are handled.

- `apt`: Używając tej metody, jeśli zaznaczono brakujący pakiet, instalacja pakietu zakończy się niepowodzeniem. To jest domyślne ustawienie.
- `aptitude`: Używając tej metody, jeśli zaznaczono brakujący pakiet, instalacja pakietu zakończy się powodzeniem.

8.4.2 Używanie serwera proxy z APT

One commonly required APT configuration is to deal with building an image behind a proxy. You may specify your APT proxy with the `--apt-ftp-proxy` or `--apt-http-proxy` options as needed, e.g.

```
$ lb config --apt-http-proxy http://proxy/
```

8.4.3 Podkręcanie APT celu zaoszczędzenia miejsca

You may find yourself needing to save some space on the image medium, in which case one or the other or both of the following options may be of interest.

Jeśli nie chcesz zawierać indeksów APT w obrazie, można je pominąć z:

```
$ lb config --apt-indices false
```

This will not influence the entries in `/etc/apt/sources.list`, but merely whether `/var/lib/apt` contains the indices files or not. The tradeoff is that APT needs those indices in order to operate in the live system, so before performing `apt-cache search` or `apt-get install`, for instance, the user must `apt-get update` first to create those indices.

If you find the installation of recommended packages bloats your image too much, provided you are prepared to deal with the consequences discussed below, you may disable that default option of APT with:

```
$ lb config --apt-recommends false
```

The most important consequence of turning off recommends is that `live-boot` and `live-config` themselves recommend some packages that provide important functionality used by most Live configurations, such as `user-setup` which `live-config` recommends and is used to create the live user. In all but the most exceptional circumstances you need to add back at least some of these recommends to your package lists or else your image will not work as expected, if at all. Look at the recommended packages for each of the `live-*` packages included in your build and if you are not certain you can omit them, add them back into your package lists.

The more general consequence is that if you don't install recommended packages for any given package, that is, "packages that would be found together with this one in all but unusual installations" (Debian Policy Manual, section 7.2), some packages that users of your Live system actually need may be omitted. Therefore, we suggest you review the difference turning off recommends makes to your packages list (see the `binary.packages` file generated by `lb build`) and re-include in your list any missing packages that you still want installed. Alternatively, if you find you only want a small number of recommended packages left out, leave recommends enabled and set a negative APT pin priority on selected packages to prevent them from being installed, as explained in [<APT pinning>](#).

8.4.4 Przekazywanie opcji do apt lub aptitude

If there is not a `lb config` option to alter APT's behaviour in the way you need, use `--apt-options` or `--aptitude-options`

to pass any options through to your configured APT tool. See the man pages for `apt` and `aptitude` for details. Note that both options have default values that you will need to retain in addition to any overrides you may provide. So, for example, suppose you have included something from `snapshot.debian.org` for testing purposes and want to specify `Acquire::Check-Valid-Until=false` to make APT happy with the stale Release file, you would do so as per the following example, appending the new option after the default value `--yes`:

```
$ lb config --apt-options "--yes -oAcquire::Check-Valid-Until=false"
```

Please check the man pages to fully understand these options and when to use them. This is an example only and should not be construed as advice to configure your image this way. This option would not be appropriate for, say, a final release of a live image.

For more complicated APT configurations involving `apt.conf` options you might want to create a `config/apt/apt.conf` file instead. See also the other `apt-*` options for a few convenient shortcuts for frequently needed options.

8.4.5 Pinning APT

For background, please first read the `apt_preferences(5)` man page. APT pinning can be configured either for build time, or else for run time. For the former, create `config/archives/*.pref`, `config/archives/*.pref.chroot`, and `config/apt/preferences`. For the latter, create `config/includes.chroot/etc/apt/preferences`.

Let's say you are building a **jessie** live system but need all the

live packages that end up in the binary image to be installed from **sid** at build time. You need to add **sid** to your APT sources and pin the live packages from it higher, but all other packages from it lower, than the default priority. Thus, only the packages you want are installed from **sid** at build time and all others are taken from the target system distribution, **jessie**. The following will accomplish this:

494

```
$ echo "deb http://mirror/debian/ sid main" > config/archives/sid.list.<↵
chroot
$ cat >> config/archives/sid.pref.chroot << EOF
Package: live-*
Pin: release n=sid
Pin-Priority: 600

Package: *
Pin: release n=sid
Pin-Priority: 1
EOF
```

495

Negative pin priorities will prevent a package from being installed, as in the case where you do not want a package that is recommended by another package. Suppose you are building an LXDE image using `task-lxde-desktop` in `config/package-lists/desktop.list.chroot`, but don't want the user prompted to store wifi passwords in the keyring. This metapackage depends on *lxde-core*, which recommends *gksu*, which in turn recommends *gnome-keyring*. So you want to omit the recommended *gnome-keyring* package. This can be done by adding the following stanza to `config/apt/preferences`:

496

```
Package: gnome-keyring
Pin: version *
Pin-Priority: -1
```

Dostosowywanie zawartości

9. Dostosowywanie zawartości

Ten rozdział omawia dokładną regulację dostosowywania zawartości systemu poza samym wybieraniem pakietów do wyboru, które będą zawarte. Uwzględnianie pozwala dodać lub wymienić dowolne pliki w obrazie systemu live, haki pozwalają na wykonanie dowolnego polecenia na różnych etapach produkcji oraz w czasie rozruchu a opcji preseeding pozwala skonfigurować pakiety, gdy są zainstalowane poprzez dostarczenie odpowiedzi na pytania debconfa.

9.1 Uwzględnianie

Chociaż najlepiej byłoby gdyby system live zawierał się wyłącznie z plików dostarczonych przez całkowicie niemodyfikowane pakiety, to czasami wygodniejszym jest, dostarczenie lub zmodyfikowanie pewnych treści za pomocą plików. Korzystając z uwzględniania, można dodać (lub wymienić) dowolne pliki w systemie obrazu live. *live-build* dostarcza dwa takie mechanizmy wykorzystania:

- **Lokalne uwzględnianie chroot:** Pozwoli Ci dodać lub zastąpić pliki w systemie plików chroot/live. Zobacz **«Lokalne uwzględnianie chroot/live»**, aby uzyskać więcej informacji.
- **Lokalne uwzględnianie danych binarnych:** Pozwoli Ci dodać lub zastąpić pliki w obrazie binarnym. Zobacz **«Lokalnych uwzględnianie danych binarnych»**, aby uzyskać więcej informacji.

Proszę zobaczyć **«Definicje»** aby uzyskać więcej informacji na temat różnicy między obrazami “binarnymi” a obrazami “live”.

9.1.1 Lokalnie uwzględniane w chroot/live

Lokalnie uwzględnione w chroot mogą być używane, aby dodać lub zastąpić pliki w systemie plików chroot/live, tak aby mogły być one użyte w systemie live. Typowym zastosowaniem jest dostarczenie szkieletowego katalogu użytkownika (`/etc/skel`) używanego przez system live do tworzenia katalogu domowego użytkownika live. Innym jest dostarczanie plików konfiguracyjnych, które mogą być po prostu dodane lub podmienione w obrazie bez przetwarzania; zobacz **«Lokalne haki chroot/live»**, czy potrzebne jest przetwarzanie.

Aby dołączyć pliki, po prostu dodaj je do katalogu `config/includes.chroot`. Ten katalog odnosi się do katalogu root / systemu live. Na przykład, aby dodać plik `/var/www/index.html` do systemu live, użyj:

```
$ mkdir -p config/includes.chroot/var/www
$ cp /path/to/my/index.html config/includes.chroot/var/www
```

Konfiguracja będzie mieć następujący układ:

```
- config
  [...]
  |-- includes.chroot
  |   |-- var
  |       |-- www
  |           |-- index.html
  [...]
```

Lokalnie uwzględnione w chroot są instalowane po instalacji pakietów, tak że pliki zainstalowane przez pakiety są zastępowane.

9.1.2 Lokalnie uwzględniane dane binarne

Aby zawierać materiały takie jak filmy lub dokumentacja na systemie plików nośnika tak, aby były one dostępne od razu po włożeniu nośnika bez uruchamiania systemu live, możesz użyć lokalnego uwzględniania danych binarnych. Działa to w podobny sposób do lokalnego uwzględniania w chroot/live. Załóżmy na przykład, że pliki `~/video_demo.*` to filmy demonstracyjne systemu live opisanego przez i dowiązane przez stronę indeksu HTML. Wystarczy skopiować materiał z `config/includes.binary/` w następujący sposób:

```
$ cp ~/video_demo.* config/includes.binary/
```

Pliki te pojawią się teraz w katalogu głównym nośnika live.

9.2 Haki

Haki pozwalają poleceniom aby były wykonywane w etapach chroot i binarnym kompilacji w celu dostosowania obrazu.

9.2.1 Lokalne haki chroot/live

Aby uruchomić komendy na etapie chroot, należy utworzyć skrypt hak z przyrostkiem `.hook.chroot` zawierającym polecenia w katalogu `config/hooks/`. Hak będzie działać w chroot po reszcie konfiguracji chroot zastosowano, więc pamiętaj, aby zapewnić, że konfiguracja zawiera wszystkie pakiety i pliki do Twoich potrzeb haka w celu uruchomienia. Zobacz przykładowe skrypty hak chroot dla różnych zadań dostosowywania wspólnej chroot przewidzianych w `# {/usr/share/doc/live-build/przykłady/`

haki} #, który można skopiować lub linku do korzystania z nich w swoim własnym konfiguracji.

9.2.2 Haki podczas uruchamiania

Aby wykonać polecenia przy starcie systemu, możesz dostarczyć haki *live-config*, jak wyjaśniono w jego podręczniku man w sekcji “Customization” (ang. dostosowywanie). Przeanalizuj własne haki *live-config* dostarczone w `/lib/live/config/`, zwracając uwagę na numery sekwencji. Następnie dodaj swój własne hak z odpowiednim prefiksem numeru sekwencji, albo jako lokalnie uwzględnione w chroot w `config/includes.chroot/lib/live/-config/`, lub w postaci niestandardowego pakietu omówione w [<Instalowanie zmodyfikowanych pakietów lub pakietów innych firm>](#).

9.2.3 Lokalne haki binarne

Aby uruchomić komendy na etapie binarnym, należy utworzyć skrypt hak z przyrostkiem `.hook.binary` zawierającym polecenia w katalogu `config/hooks/`. Hak będzie uruchomiony po wszystkich innych binarnych poleceniach, ale przed `binary_checksums`; ostatnim poleceniem. Polecenia haka nie działają w środowisku chroot, więc należy zadbać, aby nie modyfikować żadnych plików poza drzewem kompilacji albo może to uszkodzić system kompilacji! Zobacz przykładowe skrypty binarne dla różnych typowych binarnych zadań dostosowywania przewidzianych w `/usr/share/doc/live-build/examples/hooks`, które można skopiować lub utworzyć dowiązanie symboliczne aby skorzystać z nich w swojej własnej konfiguracji.

9.3 Wstępne ustawienie pytań Debconfa (Preseeding)

524

- 525 Pliki w katalogu `config/preseed/` z rozszerzeniem `.cfg` w następującym etapie (`.chroot` lub `.binary`) są brane pod uwagę jako pliki preseed debconfa i są instalowane za pomocą *live-build* przez `debconf-set-selections` podczas odpowiedniego etapu.
- 526 Aby uzyskać więcej informacji o debconf, proszę przeczytaj `debconf(7)` z pakietu *debconf*.

Dostosowywanie zdarzeń podczas uruchamiania systemu

10. Dostosowywanie zdarzeń podczas uruchamiania systemu

Cała konfiguracja, która odbywa się w czasie pracy systemu jest wykonywana przez *live-config*. Oto niektóre z najbardziej popularnych opcji *live-config*, którymi mogą być zainteresowani użytkownicy. Pełną listę wszystkich możliwości można znaleźć w podręczniku man pakietu *live-config*.

10.1 Personalizacja użytkownika live

Jednym ważnym czynnikiem jest to, że użytkownik jest tworzony przez *live-boot* w czasie startu systemu, a nie *live-build* w czasie kompilacji. To wpływa nie tylko, na to gdzie materiały dotyczące użytkownika live są wprowadzone w kompilacji, jak to opisano w [«Uwzględnianie lokalne Live/chroot»](#), ale również na wszelkie grupy i uprawnienia związane z użytkownikiem live.

Można określić dodatkowe grupy, do których użytkownik live będzie należeć korzystając z jednej z możliwości, aby skonfigurować *live-config*. Na przykład, aby dodać użytkownika live do grupy fuse, można dodać następujący plik w `config/includes.chroot/etc/live/config/user-setup.conf`:

```
LIVE_USER_DEFAULT_GROUPS="audio cdrom dip floppy video plugdev netdev ↔
powerdev scanner bluetooth fuse"
```

lub użyj `live-config.user-default-groups=audio,cdrom,dip,floppy,video,plugdev,netdev,powerdev,scanner,bluetooth,fuse` jako parametru startowego.

Możliwe jest również, aby zmienić domyślną nazwę użytkownika "user" i domyślne hasło "live". Jeśli chcesz to zrobić, z jakiegokolwiek powodu, można to łatwo osiągnąć w następujący sposób:

Aby zmienić domyślną nazwę użytkownika należy po prostu określić ją w konfiguracji:

```
$ lb config --bootappend-live "boot=live components username=live-user"
```

Jednym z możliwych sposobów zmiany domyślnego hasła jest użycie odpowiedniego haka, jak opisano w [«Haki podczas uruchamiania systemu»](#). W tym celu można użyć haka "passwd" z `/usr/share/doc/live-config/examples/hooks`, przedrostkiem jest odpowiednio (np. 2000-passwd), należy go dodać do `config/includes.chroot/lib/live/config/`

10.2 Ustawianie lokalizacji i języka

Podczas uruchamiania systemu live, język jest definiowany przez dwa etapy:

- generowanie plików lokalizacji
- ustawienie konfiguracji klawiatury

Domyślne ustawieniem lokalnym podczas budowania systemu live jest `locales=en_US.UTF-8`. Aby określić ustawienia regionalne, które powinny być wygenerowane, użyj parametru `locales` w opcji `--bootappend-live` polecenia `lb config`, np.

```
$ lb config --bootappend-live "boot=live components locales=de_CH.UTF-8"
```

Wiele lokalizacji może być określone w postaci listy rozdzielonej przecinkami.

Parametr ten, jak również parametr konfiguracyjny klawiatury jak wskazano poniżej, może być również używany w linii poleceń jądra. Można określić ustawienia regionalne poprzez `language_country` (w tym przypadku używane jest kodowanie domyślne) lub pełnej nazwy z kodowaniem `language_country.encoding`. Lista obsługiwanych lokalizacji i kodowań można znaleźć w `/usr/share/i18n/SUPPORTED`.

Zarówno konfiguracja konsoli i klawiatury X wykonywana jest przez `live-config` przy pomocy pakietu `console-setup`. Aby je skonfigurować, ustaw parametry startowe `keyboard-layouts`, `keyboard-variants`, `keyboard-options` i `keyboard-model` przez opcję `--bootappend-live`. Prawidłowe wartości opcje można znaleźć w `/usr/share/X11/xkb/rules/base.lst`. Aby znaleźć układy i warianty klawiatury dla danego języka, spróbuj wyszukać nazwę w języku angielskim i/lub kraj, gdzie mówi się danym językiem, np.:

```
$ egrep -i '(!|german.*switzerland)' /usr/share/X11/xkb/rules/base.lst
! model
! layout
  ch                German (Switzerland)
! variant
  legacy            ch: German (Switzerland, legacy)
  de_nondeadkeys    ch: German (Switzerland, eliminate dead keys)
  de_sundeadkeys    ch: German (Switzerland, Sun dead keys)
  de_mac            ch: German (Switzerland, Macintosh)
! option
```

Należy pamiętać, że każdy wariant wymienia układ, którego dotyczy w opisie.

Często tylko układ klawiatury musi być skonfigurowany. Na

przykład, aby uzyskać listę plików lokalizacyjnych dla niemieckiego i szwajcarskiego niemieckiego układu klawiatury w systemie X użyj:

```
$ lb config --bootappend-live "boot=live components locales=de_CH.UTF-8 ↔
keyboard-layouts=ch"
```

Jednak dla bardzo konkretnych przypadków użycia, można dodać inne parametry. Na przykład, aby ustawić francusko języczny system z układem klawiatury French-Dvorak (zwany Bepo) na klawiaturze typu USB TypeMatrix EZ-Reach 2030, użyj:

```
$ lb config --bootappend-live \
  "boot=live components locales=fr_FR.UTF-8 keyboard-layouts=fr keyboard↔
  -variants=bepo keyboard-model=tm2030usb"
```

Wiele wartości oddzielonych przecinkami może być przypisane do każdego z parametrów `keyboard-*`, z wyjątkiem `keyboard-model`, który przyjmuje tylko jedną wartość. Proszę przejrzeć podręcznik `man keyboard(5)` aby uzyskać więcej szczegółów i przykładów zmiennych `XKBMODEL`, `XKBLAYOUT`, `XKBVARIANT` i `XKBOPTIONS`. Jeśli podano wiele `keyboard-variants` (ang warianty klawiatury), będą one dopasowane jeden do drugiego przez wartość `keyboard-layouts` (patrz opcja `setxkbmap(1) -variant`). Puste wartości są dozwolone; na przykład aby zdefiniować dwa układy, domyślny US QWERTY oraz drugi US Dvorak, zastosuj:

```
$ lb config --bootappend-live \
  "boot=live components keyboard-layouts=us,us keyboard-variants=,dvorak↔
  "
```

10.3 Persistence

Odmianą Live CD jest preinstalowany system, który uruchamia się z nośników tylko do odczytu, takich jak cdrom, gdzie operacje zapisu i modyfikacje nie przetrwają restartów sprzętowych hosta, na którym jest uruchomiony.

System live jest uogólnieniem tego paradygmatu, a tym samym wspiera media inne prócz płyt CD; ale dalej jako jego domyślne zachowanie, należy uważać operacje tylko do odczytu a wszelkie zmiany, w czasie działania są tracone podczas zamykania systemu.

'Persistence' (ang. trwałość) to wspólna nazwa dla różnych rodzajów rozwiązań dla zapisania niektórych lub wszystkich danych między restartami podczas używania i wprowadzania zmian do systemu. Aby zrozumieć, jak to działa to przydatna była by wiedza, że nawet wtedy, gdy system jest uruchamiany i działa z nośnika tylko do odczytu, to modyfikacje plików i katalogów są zapisywane na zapisywalnych nośnikach, typowo dysk RAM (tmpfs) a dane w pamięci RAM nie przetrwają restartu.

Dane przechowywane na tym ramdysku powinny być przechowywane na zapisywalnym trwałym nośniku takim jak lokalny dysk, lokalny udział sieciowy lub nawet na sesji wielosesyjnego dysku CD-RW/DVD-RW. Wszystkie te nośniki są obsługiwane w systemach live na różne sposoby i wszystkie, oprócz ostatniej wymagają specjalnego parametru startowego określonego w czasie startu systemu: persistence.

Jeśli ustawiony jest parametr startowy persistence (a nopersistence nie jest ustawiony), lokalne nośniki (np. dyski twarde, napędy USB) będą przeszukane w celu znalezienia woluminów trwałości podczas startu systemu. Możliwe jest ograniczenie, jakiego typu woluminy trwałości będą wykorzystane przez określenie pewnych parametrów startowych opisanych

w podręczniku man *live-boot(7)*. Wolumin trwałości może być każdym z wymienionych:

- partycja, identyfikowana po nazwie GPT.
- system plików, identyfikowany po etykiecie.
- plik obrazu zlokalizowany na każdym obsługiwanym systemie plików (nawet na partycji NTFS innego systemu), identyfikowany po nazwie pliku.

Etykietą dla woluminu musi być persistence, ale będzie ignorowane, dopóki nie będzie zawarty w katalogu głównym plik o nazwie persistence.conf, który jest używany by pełni dostosować wolumin persistence, to znaczy, że wskazuje się w nim katalogi, w których chcesz zapisać na woluminie zmiany przy restarcie. Zobacz <Plik persistence.conf>, aby uzyskać więcej szczegółów.

Oto kilka przykładów, jak przygotować wolumin, aby mógł być on użyty z opcją persistence. Może to być, na przykład, partycja ext4 na dysku twardym lub na nośniku wymiennym stworzona przez, np.:

```
# mkfs.ext4 -L persistence /dev/sdb1
```

Zobacz również <Wykorzystanie przestrzeni pozostałej na nośniku USB>.

Jeśli masz już partycję na urządzeniu, można po prostu zmienić jego etykietę używając następującego polecenia:

```
# tune2fs -L persistence /dev/sdb1 # for ext2,3,4 filesystems
```

Oto przykład, jak stworzyć plik obrazu opartego na ext4 do zastosowania z opcją persistence: 571

```
$ dd if=/dev/null of=persistence bs=1 count=0 seek=1G # for a 1GB sized ↔
  image file
$ /sbin/mkfs.ext4 -F persistence
```

Po utworzeniu pliku obrazu, na przykład, aby sprawić by katalog /usr był prwały, ale tylko zapisywał zmiany wprowadzone w tym katalogu, a nie całą zawartość /usr, można użyć opcji "union". Jeśli plik obrazu znajduje się w katalogu domowym, należy skopiować go do katalogu głównego systemu plików na dysku twardym i zamontować go w /mnt w następujący sposób:

```
# cp persistence /
# mount -t ext4 /persistence /mnt
```

Następnie utwórz plik persistence.conf dodając zawartość i odmontowując plik obrazu.

```
# echo "/usr union" >> /mnt/persistence.conf
# umount /mnt
```

Teraz uruchom ponownie i wybierz nośnik live, a następnie uruchom dodając parametr startowy "persistence".

10.3.1 Plik persistence.conf

Partycję z etykietą persistence należy skonfigurować za pomocą pliku persistence.conf, aby dowolne katalogi stały się trwałe. Ten

plik, znajdujący się w głównym katalogu systemu plików partycji, kontroluje które katalogi są trwałe i w jaki sposób.

To jak niestandardowe wierzchnie zamontowania są skonfigurowane jest opisane w szczegółach w podręczniku man persistence.conf(5), ale ten prosty przykład powinien być wystarczający dla większości zastosowań. Powiedzmy, że chcemy, aby nasz katalog domowy i cache APT było trwałe w pamięci podręcznej systemu plików ext4 na partycji /dev/sdb1:

```
# mkfs.ext4 -L persistence /dev/sdb1
# mount -t ext4 /dev/sdb1 /mnt
# echo "/home" >> /mnt/persistence.conf
# echo "/var/cache/apt" >> /mnt/persistence.conf
# umount /mnt
```

Następnie uruchamiamy ponownie komputer. Podczas pierwszego uruchomienia zawartość /home i /var/cache/apt zostanie skopiowana do woluminu trwałości i od tej pory wszystkie zmiany w tych katalogach będą przechowywane na tym woluminie. Należy pamiętać, że wszelkie ścieżki wymienione w pliku persistence.conf nie mogą zawierać spacji lub specjalnych komponentów ścieżki: . i . . . Ponadto, ani /lib, /lib/live (lub którykolwiek z jego podkatalogów) ani / nie może zostać utrwalony za pomocą własnych punktów montowania. Jako obejście tego ograniczenia można dodać / union do pliku persistence.conf w celu osiągnięcia pełnej trwałości.

10.3.2 Używanie więcej niż jednego magazynu persistence

Istnieją różne sposoby korzystania z wielu magazynów trwałości (ang. persistence) dla różnych zastosowań. Na przykład, przy

używanie kilku magazynów w tym samym czasie lub wybranie tylko jednego, spośród różnych, do bardzo specyficznych zastosowań.

585 Kilka różnych niestandardowych woluminów-nakładek (z własnymi plikami `persistence.conf`) może być używane w tym samym czasie, ale jeżeli kilka woluminów tworzy ten sam katalog trwałym, będzie używany tylko jeden z nich. Jeśli jakieś dwa punkty montowania są "zagnieżdżone" (np. jeden jest podkatalogiem drugiego) to katalog parent (ang. rodzic) zostanie zamontowany przed katalogiem child (ang. dziecko) tak że jeden punkt montowania nie będzie ukryty przed innym. Zagnieżdżone niestandardowe woluminy są problematyczne, jeżeli są wymienione w tym samym pliku `persistence.conf`. Zobacz podręcznik `man persistence.conf(5)`, jak radzić sobie z tym przypadkiem, jeśli naprawdę potrzebujesz (Wskazówka: zazwyczaj nie potrzebujesz).

586 Jednym z możliwych przypadków użycia: Jeśli chcesz przechowywać dane użytkownika np. `/home` i dane superużytkownika tj. `/root` na różnych partycjach, utwórz dwie partycje z etykietą `persistence` i dodaj plik `persistence.conf` na każdą z nich, tak `## echo "/home" > persistence.conf` na pierwszej partycji, przez co będą zapisywane pliki użytkownika i `# echo "/root" > persistence.conf` na drugiej partycji, która będzie przechowywać pliki superużytkownika. Wreszcie, należy użyć parametru startowego `persistence`.

587 Jeśli użytkownik będzie potrzebował wiele magazynów trwałości tego samego typu dla różnych miejsc lub dla celów testowych, takich jak magazyny `private` i `#{work}#` parametr startowy `persistence-label` użyty w połączeniu z parametrem `persistence` pozwoli na wiele unikatowych magazynów trwałości. Przykładem może być, jeśli użytkownik chciałby użyć partycji trwałości oznaczonej `private` dla prywatnych danych, takich jak

zakładki w przeglądarce lub innych typów danych, to mógłby użyć parametrów startowych: `#{persistence}# persistence-label=private`. A do przechowywania danych związanych z pracą, takich jak dokumenty, projekty badawcze lub inne rodzaje, mógłby skorzystać z parametrów startowych: `persistence persistence-label=work`.

Ważne jest, aby pamiętać, że każda z tych partycji, `private` i `#{work}#`, także potrzebuje pliku `persistence.conf`. Podręcznik `man` pakietu `live-boot` zawiera więcej informacji o tym, jak korzystać z tych etykiet z zapisanymi nazwami.

10.4 Używanie opcji `persistence` z szyfrowaniem

Korzystanie z funkcji trwałości (ang. `persistence`) oznacza, że niektóre poufne dane mogą zostać narażone na ryzyko. Zwłaszcza jeżeli trwałe dane są przechowywane na urządzeniu przenośnym, takim jak pamięci USB lub zewnętrzne dyski twarde. To jest miejsce, gdzie przydatne staje się szyfrowanie. Nawet jeśli cała procedura może wydawać się skomplikowana, ze względu na liczbę kroków, które należy podjąć, to jest bardzo łatwo obsługiwać szyfrowane partycje z `live-boot`. Aby móc korzystać z **luks**, który jest obsługiwanym typem szyfrowania, musisz zainstalować `cryptsetup` zarówno na maszynie tworzenia zaszyfrowanych partycji, a także w systemie `live`, który będzie używał szyfrowanej trwałej partycji.

Aby zainstalować `cryptsetup` na twoim komputerze:

```
# apt-get install cryptsetup
```

Aby zainstalować `cryptsetup` na twoim systemie `live` dodaj go do listy pakietów:

```
$ lb config
$ echo "cryptsetup" > config/package-lists/encryption.list.chroot
```

Gdy Twój system live posiada *cryptsetup*, to w zasadzie wystarczy, już tylko utworzyć nową partycję, zaszyfrować ją i uruchomić z parametrami *persistence* i *persistence-encryption=luks*. Mogliśmy już przewidzieć ten krok i dodać parametry startowe w czasie kompilacji przestrzegając następującej procedury:

```
$ lb config --bootappend-live "boot=live components persistence persistence-<->
-encryption=luks"
```

przejdźmy do szczegółów dla tych wszystkich, którzy nie są zaznajomieni z szyfrowaniem. W poniższym przykładzie mamy zamiar użyć partycji na dysku USB, która odpowiada */dev/sdc2*. Należy zaznaczyć, że należy ustalić, która partycja jest jedną tą, którą masz zamiar używać w tym konkretnym przypadku.

Pierwszym krokiem jest podłączenie dysku USB i określenie, którym jest urządzeniem. Zalecaną metodą tworzenia listy urządzeń w *live-manual* jest `ls -l /dev/disk/by-id`. Następnie utworzymy nową partycję, a następnie zaszyfrujemy ją hasłem w następujący sposób:

```
# cryptsetup --verify-passphrase luksFormat /dev/sdc2
```

Następnie otwieramy partycję LUKS w wirtualnym elemencie odwzorowującym urządzenia ***/dev/mapper***. Można tu użyć dowolnej nazwy. Używamy ***live*** jako przykład:

```
# cryptsetup luksOpen /dev/sdc2 live
```

Następnym krokiem jest wypełnienie urządzenia zerami przed utworzeniem systemu plików:

```
# dd if=/dev/zero of=/dev/mapper/live
```

Teraz jesteśmy gotowi do stworzenia systemu plików. Warto zauważyć, że dodajemy etykietę *persistence* tak, aby urządzenie zostało zamontowane jako *persistence store* (magazyn *persistence*) w czasie startu systemu.

```
# mkfs.ext4 -L persistence /dev/mapper/live
```

Aby kontynuować naszą konfigurację, musimy zamontować urządzenie, na przykład w */mnt*.

```
# mount /dev/mapper/live /mnt
```

I stwórz plik *persistence.conf* w katalogu głównym partycji. To jest, jak wyjaśniono wyżej, absolutnie konieczne. Zobacz **<Plik persistence.conf>**.

```
# echo "/" union" > /mnt/persistence.conf
```

Potem odmontuj punkt montowania:

```
# umount /mnt
```

612 I opcjonalnie, choć może to być dobry sposób na zabezpieczenie
danych, które właśnie dodaliśmy do partycji, możemy zamknąć
urządzenie:

613

```
# cryptsetup luksClose live
```

614 Podsumujmy proces. Do tej pory stworzyliśmy system live z
możliwością szyfrowania, który można skopiować na nośnik usb,
jak wyjaśniono w [«kopiowaniu hybrydowego obrazu ISO na nośnik
pamięci USB»](#). Stworzyliśmy również zaszyfrowaną partycję, która
może znajdować się na tym samym nośniku usb, aby można było
go nosić ze sobą wszędzie i mamy skonfigurowaną zaszyfrowaną
partycję, stosowaną jako magazyn persistence. Więc teraz,
musimy tylko uruchomić system live. W czasie startu systemu, na
live-boot poprosi nas o wpisanie hasła i zamontuje zaszyfrowaną
partycję używaną przez opcję persistence.

Dostosowywanie obrazu binarnego

11. Dostosowywanie obrazu binarnego

11.1 Programy ładujące (ang. Bootloadery)

live-build używa *syslinux* i niektórych jego pochodnych (w zależności od typu obrazu) w domyślnym programie ładującym (ang. bootloader). Można je łatwo dostosować do własnych potrzeb.

W celu wykorzystania pełnego motywu, skopiuj `/usr/share/live/build/bootloaders` do `config/bootloaders` i edytuj tam te pliki. Jeśli nie chcesz się martwić modyfikacją wszystkich obsługiwanych konfiguracji programu ładującego (ang. bootloader), tylko zapewnienie lokalnego zmodyfikowaną kopię jednego z typu programów, np. `* {isolinux} * w # {config / programy ładujące / isolinux} #` wystarczy też, w zależności od przypadku użycia.

Podczas modyfikacji jednego z domyślnych motywów, jeśli chcesz korzystać z indywidualnego obrazu tła, który będzie wyświetlany razem z menu startowym, dodaj obraz `splash.png` 640x480 pikseli. Następnie usuń plik `splash.svg`.

Istnieje wiele możliwości, jeśli chodzi o wprowadzanie zmian. Na przykład, pochodne *syslinux* mają domyślnie skonfigurowany limit czasowy (ang. timeout) na 0 (zero), co oznacza, że wstrzymują się one na czas nieokreślony na w ich ekranie powitalnym aż do naciśnięcia klawisza.

Aby zmienić limit czasowy podczas rozruchu w domyślnym obrazie *iso-hybrid* wystarczy zmienić domyślny plik **`isolinux.cfg`** określając limit czasu (ang. timeout) w jednostkach 1/10 sekundy. Zmodyfikowany **`isolinux.cfg`** uruchamiający rozruch po pięciu sekundach byłby podobny do tego:

```
include menu.cfg
default vesamenu.c32
prompt 0
timeout 50
```

11.2 Metadane ISO

Podczas tworzenia binarnego obrazu ISO9660, można korzystać z następujących opcji, aby dodać różne metadane tekstowe do obrazu. To może pomóc w identyfikacji wersji lub konfiguracji obrazu bez uruchamiania go.

- **LB_ISO_APPLICATION/--iso-application NAZWA:** Pole to powinno opisywać zastosowanie obrazu. Maksymalna długość tego pola wynosi 128 znaków.
- **LB_ISO_PREPARER/--iso-preparer NAME:** Pole to powinno opisywać przygotowującego obraz, zwykle z kilkoma danymi kontaktowymi. Domyślną wartością tej opcji jest używana wersja *live-build*, które może później pomóc przy debugowaniu. Maksymalna długość tego pola wynosi 128 znaków.
- **LB_ISO_PUBLISHER/--iso-publisher NAME:** Pole to powinno opisywać wydawcę obrazu, zwykle z kilkoma danymi kontaktowymi. Maksymalna długość tego pola wynosi 128 znaków.
- **LB_ISO_VOLUME/--iso-volume NAME:** Pole to powinno opisywać ID woluminu obrazu. Jest używane jako etykieta widoczna dla użytkownika na niektórych platformach, takich jak Windows i Apple Mac OS. Długość maksymalna dla tego pola to 32 znaków.

Dostosowywanie Instalatora Debiana

12. Dostosowywanie Instalatora Debiana

Obrazy systemów live mogą być zintegrowane z Instalatorem Debiana. Istnieje wiele różnych typów instalacji, różniących się tym, co jest załączone w obrazie i jak działa instalator.

Proszę zwrócić uwagę na uważne wykorzystanie wielkich liter, podczas odnoszenia się do "Debian Installer" (ang. Instalatora Debiana) w tej sekcji - kiedy jest stosowany tak, że odnosi się wyraźnie do oficjalnego instalatora dla systemu Debian, a nie cokolwiek innego, to jest często postrzegana w skrócie jako "d-i".

12.1 Typy Instalatora Debiana

Trzy główne rodzaje instalatora to:

Instalator Debiana "Normal" : To jest normalny obraz systemu live z oddzielnym jądrem i initrd, który (gdy wybrany z odpowiedniego menu programu ładującego) uruchamia do standardowej instancji Instalatora Debiana, tak jakby pobrano obraz płyty Debiana i uruchomiło się go. Obrazy zawierające system typu live i inny niezależny instalator są często określane jako "połączone obrazy" (ang. combined images).

Na takich obrazach, Debian jest instalowany przez pobieranie i instalowanie pakietów .deb za pomocą *debootstrap*, z lokalnych mediów lub jakiegoś opartego na sieci, w wyniku czego domyślny system Debian jest instalowany na dysku twardym.

Cały proces może być zapisany w pliku preseed i dostosowany na wiele różnych sposobów, przeczytaj odpowiednie strony w instrukcji Instalatora Debiana, aby uzyskać więcej informacji. Kiedy

utworzysz działający plik preseed, *live-build* może automatycznie umieścić go w Twoim obrazie i włączyć go dla Ciebie.

Instalator Debiana "Live" : To jest obraz systemu live z oddzielnym jądrem i initrd, który (gdy wybrany z odpowiedniego menu programu ładującego) uruchamia do instancji Instalatora Debiana.

Instalacja będzie przebiegać w identyczny sposób do instalacji "normalnej" opisanej powyżej, ale na tym samym etapie instalacji, zamiast używać *debootstrap* aby pobrać i zainstalować pakiety, systemu plików obrazu live zostanie skopiowany do celu. Jest to możliwe dzięki specjalnym pakiecie udeb zwanym *live-installer*.

Po tym etapie, Instalator Debiana kontynuuje normalnie, instalując i konfigurując elementy, takie jak programy ładujące i lokalnych użytkowników, itp.

Uwaga: aby zapewnić wsparcie dla wpisów zarówno normalnych i instalatora live w bootloaderze (ang. programie ładującym) na tym samym nośniku live, należy wyłączyć instalatora live przez wpis w pliku preseed `live-installer/enable=false`.

Instalator Debiana "Desktop" : Niezależnie od wybranego rodzaju Instalatora Debiana, di może zostać uruchomiony z pulpitu, klikając na jego ikonę. Jest bardziej przyjazny użytkownikowi w niektórych sytuacjach. W celu skorzystania z tej opcji, pakiet *debian-installer-launcher* musi zostać uwzględniony.

Należy pamiętać, że domyślnie, *live-build* nie obejmuje obrazów Instalatora Debiana w obrazach, musi to być sprecyzowane przez `lb config`. Również należy również pamiętać, że aby działał "Desktop Installer" (ang. instalator uruchamiany z poziomu pulpitu) to jądro systemu live musi być zgodne z jądrem wykorzystywanym przez `d-i` dla określonej architektury. Na przykład:

mogą być zawarte w initrd instalatora, w sposób podobny do **Uwzględnianie lokalne live/chroot**, poprzez umieszczenie materiałów w `config/includes.installer/`.

```
$ lb config --architectures i386 --linux-flavours 486 \
  --debian-installer live
$ echo debian-installer-launcher >> config/package-lists/my.list.chroot
```

12.2 Dostosowywanie Instalatora Debiana przez preseeding

Jak opisano w Podręczniku Instalatora Debiana, Załącznik B na <https://www.debian.org/releases/stable/i386/apb.html>, “Plik preseed dostarcza sposób, aby ustawić odpowiedzi na pytania zadawane w trakcie procesu instalacji, bez konieczności ręcznego wprowadzania odpowiedzi, podczas instalacji. Pozwala to w pełni zautomatyzować większość rodzajów instalacji, a nawet oferuje kilka funkcji niedostępnych w zwykłych instalacjach.” Takie dostosowanie jest najlepiej osiągnięte używając *live-build* poprzez umieszczenie konfiguracji w pliku `preseed.cfg` zawartym w `config/includes.installer/`. Na przykład, aby ustawić lokalizację na `en_US`:

```
$ echo "d-i debian-installer/locale string en_US" \
  >> config/includes.installer/preseed.cfg
```

12.3 Dostosowywanie zawartości Instalatora Debiana

Do celów doświadczalnych lub debugowania, możesz zechcieć dołączyć lokalnie zbudowany element `di` w paczce `udeb`. Należy umieścić je w `config/packages.binary/`, aby włączyć je do obrazu. Również dodatkowe lub zamienne pliki i katalogi

Projekt

Wnoszenie wkładu do tego projektu

13. Wnoszenie wkładu do tego projektu

Wnosząc swój wkład należy jasno określić posiadacza jego praw autorskich i zawrzeć wszelkie stosowane oświadczenie licencjonowania. Należy pamiętać, że aby zmiany były zaakceptowane, wkład musi być licencjonowany na tej samej licencji co reszta dokumentów, a mianowicie, GPL w wersji 3 lub nowszej.

Wkład do projektu, taki jak tłumaczenia i poprawki, są bardzo mile widziane. Każdy może bezpośrednio zaangażować się w repozytoriach, jednak prosimy wysłać większe zmiany do listy mailingowej, aby omówić je w pierwszej kolejności. Patrz rozdział [kontakt](#) aby uzyskać więcej informacji.

`{projekt}` używa Git jako systemu kontroli wersji i zarządzania kodem źródłowym. Jak wyjaśniono w [repozytorium Git](#) są dwie główne gałęzie rozwoju: **debian** i **debian-next**. Każdy może wprowadzić zmiany do gałęzi **debian-next** w repozytoriach *live-boot*, *live-build*, *live-config*, *live-images*, *live-manual* i *live-tools*.

Jednakże istnieją pewne restrykcje. Serwer odrzuci:

- Non fast-forward pushes.
- Zmiany wymagające scalenia.
- Dodawanie lub usuwanie tagów lub gałęzi rozwojowych.

Nawet jeżeli wszystkie zmiany mogą być później zweryfikowane, prosimy abyś używał zdrowego rozsądku i tworzył dobre zmiany opisane dobrym komentarzem.

- Pisz komentarze do zmian, które składają się z pełnych, sensownych zdań w języku angielskim, począwszy od dużej

litery, a kończących się kropką. Zwykle te komentarze zaczynają się od "Fixing/Adding/Removing/Correcting/Translating/...". ("Naprawianie/Dodawanie/Usuwanie/Korygowanie/Tłumaczenie/ itd. ...".)

- Pisz dobre komentarze do zmian. Pierwsza linia musi być dokładne podsumowanie treści popełnić, która zostanie uwzględniona w liście zmian (ang. changelog). Jeśli musisz zrobić kilka wyjaśnień, napisz je niżej pozostawiając pusty wiersz po pierwszej linii, a następnie kolejny pusty wiersz po każdym akapicie. Linie każdego akapitu nie powinny przekraczać 80 znaków.

* Wysyłaj zmiany osobno. To znaczy; nie mieszaj niepowiązanych ze sobą rzeczy w tej samej zmianie. Dodać osobną zmianę dla każdej rzeczy, którą zmieniasz.

13.1 Wprowadzanie zmian

W celu wysłania zmian do repozytoriów, należy wykonać następującą procedurę. Tutaj używamy *live-manual* jako przykładu, więc zastąp go nazwą repozytorium, z którym chcesz pracować. Aby uzyskać szczegółowe informacje na temat edycji podręcznika *live-manual* zobacz: [Współtworzenie tego dokumentu](#).

- Pobierz publiczny klucz do wprowadzania zmian:

```
$ mkdir -p ~/.ssh/keys
$ wget http://live-systems.org/other/keys/git@live-systems.org -O ~/.ssh/keys/git@live-systems.org
$ wget http://live-systems.org/other/keys/git@live-systems.org.pub -O ~/.ssh/keys/git@live-systems.org.pub
$ chmod 0600 ~/.ssh/keys/git@live-systems.org*
```

- Dodaj następującą sekcję do twojej konfiguracji klienta openssh:

```
$ cat >> ~/.ssh/config << EOF
Host live-systems.org
  Hostname live-systems.org
  User git
  IdentitiesOnly yes
  IdentityFile ~/.ssh/keys/git@live-systems.org
EOF
```

- Wyślij poprawki na serwer:

```
$ git push
```

- Sprawdź i sklonuj kopię *live-manual* przez ssh:

```
$ git clone git@live-systems.org:/live-manual.git
$ cd live-manual && git checkout debian-next
```

- Upewnij się, że masz ustawionego autora i adres email w konfiguracji Git:

```
$ git config user.name "John Doe"
$ git config user.email john@example.org
```

Ważne: Pamiętaj, że powinno się wprowadzać wszelkie zmiany wyłącznie w gałęzi **debian-next**.

- Wprowadź zmiany. W tym przykładzie będzie najpierw napisać nowy dział dotyczący stosowania plastrów, a następnie przygotować się do popełnienia dodawanie plików i pisanie popełnienia wiadomość tak:

```
$ git commit -a -m "Adding a section on applying patches."
```

Zgłaszanie błędów

14. Zgłaszanie błędów

Systemy live są dalekie od doskonałości, ale chcemy, uczynić je jak najbliższe perfekcji - z Waszą pomocą. Nie wahaj się zgłosić błąd. Lepiej jest wypełnić raport dwa razy niż wcale. Ten rozdział zawiera zalecenia dotyczące sposobu składania raportów o błędach.

Dla niecierpliwych:

- Zawsze należy sprawdzić najpierw aktualizacje statusu obrazu na naszej stronie internetowej <http://live-systems.org/>, w poszukiwaniu znanych problemów.
- Przed wysłaniem raportu o błędzie zawsze spróbuj odtworzyć problem z **najnowszą wersją** gałęzi *live-build*, *live-boot*, *live-config* i *live-tools*, których używasz (jak najnowsza wersja 4.x *live-build*, jeśli używasz *live-build* 4).
- Spróbuj podać **jak najwięcej specyficznych informacji, jak to tylko możliwe** o błędzie. Obejmuje to (co najmniej) wersję używanych *live-build*, *live-boot*, *live-config* i *live-tools* oraz dystrybucję systemu live którą kompilujesz.

14.1 Znane problemy

Ponieważ Debian **testing** * i Debian ***{unstable}** dystrybucjami ruchomymi, w przypadku określenia jednej z nich jako dystrybucji systemu docelowego, kompilowanie nie zawsze zakończy się sukcesem.

Jeśli budowanie systemu opartego na **testing** lub **unstable** powoduje u Ciebie zbyt wiele trudności, raczej wykorzystaj wydanie **stable**. *live-build* zawsze ustawia domyślnie wydanie **stable**.

Obecnie znane problemy wymienione są w ramach sekcji "status" na naszej stronie internetowej w <http://live-systems.org/>.

To jest poza zakresem tej instrukcji, aby szkolić się poprawnie identyfikować i naprawiać problemy pakietów z dystrybucji rozwojowych, jednak istnieją dwie rzeczy, których zawsze można spróbować: Jeśli kompilowanie nie powiedzie się, gdy docelową dystrybucją jest **testing**, to spróbuj z **unstable**. Jeśli **unstable** również nie działa, to przywróć do dystrybucji **testing** i przypnij nowszą wersję wadliwego pakietu z wersji **unstable** (patrz **<Przypinanie APT>** aby uzyskać szczegóły).

14.2 Przebuduj od zera

Aby upewnić się, że dany błąd nie jest spowodowany nie w pełni wyczyszczonym budowanym systemem, proszę zawsze odbudować cały system live od podstaw, aby sprawdzić, czy błąd jest powtarzalny.

14.3 Używaj aktualnych pakietów

Używanie przestarzałych pakietów może powodować poważne problemy podczas próby odtworzenia (i ostatecznego rozwiązania) problemu. Upewnij się, że system na którym kompilujesz jest aktualny i wszelkie pakiety zawarte w obrazie również.

14.4 Zbierz potrzebne informacje

Proszę dostarczyć wystarczającą ilość informacji z raportem. Obejmujące co najmniej, dokładną wersję *live-build*, gdzie napotkano błąd i kroki, jak go odtworzyć. Proszę użyć zdrowego rozsądku i przedstawić wszelkie inne istotne informacje, jeśli uważasz, że może to pomóc w rozwiązaniu problemu.

Aby w pełni wykorzystać Twój raport o błędzie, będziemy potrzebować przynajmniej następujących informacji:

- Architektura systemu hosta
- Dystrybucja systemu hosta
- Wersja *live-build* na systemie hosta
- Wersja Pythona na systemie hosta
- Wersja *debootstrap* i/lub *cdebootstrap* na systemie hosta
- Architektura systemu live
- Dystrybucja systemu live
- Wersja *live-boot* na systemie live
- Wersja *live-config* na systemie live
- Wersja *live-tools* na systemie live

Można wygenerować log procesu kompilacji przy użyciu polecenia `tee`. Polecamy robić to automatycznie przy użyciu skryptu `auto/build` (patrz <Zarządzanie konfiguracją> w celu uzyskania szczegółów).

```
# lb build 2>&1 | tee build.log
```

W czasie uruchamiania *live-boot* i *live-config* przetrzymują swoje logi w `/var/log/live/`. Sprawdź je w poszukiwaniu błędów.

Dodatkowo, aby wykluczyć inne błędy, zawsze dobrym pomysłem jest, aby spakować do archiwum tar swój katalog `config/` i przesłać go gdzieś (**nie** należy wysłać go jako załącznik do listy), tak aby można spróbować odtworzyć napotkane błędy. Jeśli sprawia to trudność (np. ze względu na rozmiar) można użyć wyjścia z polecenia `lb config --dump`, które tworzy podsumowanie drzewa

konfiguracyjnego (czyli np. listę plików w podkatalogach `config/`, ale bez załączania ich).

Pamiętaj, aby wysłać wszystkie dzienniki i logi, które zostały wyprodukowane z ustawień regionalnych angielskich, np. uruchom polecenia *live-build* ze zmiennymi `LC_ALL=C` lub `LC_ALL=en_US`.

14.5 Wyizoluj prawdopodobną wadę, jeśli to możliwe

Jeśli to możliwe, należy wyizolować przypadek do najmniejszej zmiany, które generuje błąd. Nie zawsze jest łatwo to zrobić, więc jeśli nie możesz dodać takiej informacji do raportu, nie martw się. Jednakże, jeśli użytkownik dobrze planuje swój cykl rozwoju, przy użyciu małych zestawów zmian na iterację, można być w stanie wyizolować problem poprzez budowę prostszej konfiguracji “bazy”, która blisko pasuje do rzeczywistej konfiguracji oraz tylko uszkodzony zestaw zmian do niej dodany. Jeśli masz trudności z sortowaniem, które z Twoich zmian wygenerowały błąd, może to znaczyć, że uwzględniono za dużo w każdym zestawie zmian i powinno się raczej kształcić w mniejszych krokach.

14.6 Wybierz odpowiedni pakiet dla którego zgłaszasz błąd

Jeśli nie wiesz, który komponent jest odpowiedzialny za błąd, lub jeśli błąd jest ogólnym błędem dotyczącym systemów live, można wypełnić raport błędu dotyczący pseudo-pakietu `debian-live`.

Jednak będziemy wdzięczni, jeśli postarasz się zawęzić pole możliwości, w których może pojawiać się błąd.

14.6.1 W czasie budowania podczas ładowania początkowego (bootstrapping)

720 Początkowo *live-build* ładuje podstawowy system Debiana używając *debootstrap* lub *cdebootstrap*. Może się to nie powieść w zależności od używanego narzędzia do ładowania i dystrybucji Debiana, która jest pobierana. Jeśli błąd pojawi się tutaj, należy sprawdzić, czy błąd jest związany z konkretnym pakietem Debiana (najprawdopodobniej) lub, jeśli jest to związane z samym narzędziem ładowania początkowego.

721 W obu przypadkach nie jest to błąd w systemie live, ale w samym Debianie i prawdopodobnie nie możemy naprawić go bezpośrednio. Proszę zgłosić taki błąd jako dotyczący narzędzia ładowania początkowego (ang. bootstrapping tool) lub uszkodzonego pakietu.

722 14.6.2 W czasie budowania podczas instalacji pakietów

723 *live-build* instaluje dodatkowe pakiety z archiwum Debiana i w zależności od używanej dystrybucji Debian i codziennego stanu archiwum, może się to nie powieść. Jeśli błąd pojawi się w tym miejscu, należy sprawdzić, czy błąd jest powtarzalny w normalnym systemie.

724 Jeśli jest to przypadek, gdzie błąd nie występuje w systemie live, ale w Debianie - zgłoś go jako dotyczący wadliwego pakietu. Uruchomienie *debootstrap* niezależnie od samej kompilacji systemu live lub uruchomienie *lb bootstrap* - debug daje więcej informacji.

725 Ponadto, w przypadku korzystania z lokalnego serwera lustrzanego i/lub jakichkolwiek serwerów proxy i doświadczania problemów prosimy zawsze najpierw spróbować odtworzyć czynności z użyciem oficjalnego serwera lustrzanego.

719

14.6.3 W czasie uruchamiania

726

Jeśli obraz nie uruchamia się, należy zgłosić go do listy wraz z informacjami wymaganymi w **<Zbierz potrzebne informacje>**. Nie zapomnij wspomnieć, jak/kiedy dokładnie obraz nie zadziałał, czy był użyty za pomocą wirtualizacji lub rzeczywistego sprzętu. Jeśli korzystasz z technologii wirtualizacji w jakiegokolwiek formie, proszę zawsze uruchomić go na prawdziwym sprzęcie przed zgłoszeniem błędu. Zapewnienie zrzutu ekranu awarii jest również bardzo pomocne.

727

14.6.4 W czasie gdy system jest już uruchomiony

728

Jeśli pakiet został zainstalowany, ale nie działa gdy system live faktycznie działa, jest to prawdopodobnie błąd w systemie live. Jednakże:

729

14.7 Spróbuj wykonać parę kroków

730

Przed zgłoszeniem błędu, proszę poszukaj w internecie danego komunikatu o błędzie lub objawu jaki otrzymujesz. Ponieważ jest mało prawdopodobne, że jesteś jedyną osobą, która zmaga się ze szczególnym problem. Zawsze jest szansa, że został on już omówiony w innym miejscu i zaproponowano już możliwe rozwiązanie, obejście lub patch.

731

Należy zwrócić szczególną uwagę na listę mailingową systemów live, jak również strona główna, ponieważ zawierają one prawdopodobnie najbardziej aktualne informacje. Jeśli takowa informacja istnieje, zawsze należy zawrzeć odniesienie do niej w swoim raporcie o błędzie.

732

Ponadto, należy sprawdzić aktualne wykazy błędów dla *live-build*,

733

live-boot, *live-config* i *live-tools*, aby zobaczyć, czy coś podobnego nie zostało już zgłoszone.

14.8 Gdzie zgłaszać błędy

Projekt Systemów Live śledzi wszystkie błędy w systemie śledzenia błędów (BTS). Aby uzyskać informacje na temat korzystania z tego systemu, zobacz <<https://bugs.debian.org/>>. Możesz też przesłać błędy używając polecenia `#{reportbug}` z pakietu o tej samej nazwie.

Ogólnie rzecz biorąc, należy zgłaszać błędy podczas kompilacji: jako dotyczące pakietu *live-build*, błędy podczas uruchamiania: *live-boot* oraz błędy w czasie działania systemu live: jako dotyczące *live-config*. Jeśli nie jesteś pewien, który pakiet będzie odpowiedni lub potrzebujesz więcej pomocy, przed złożeniem zgłoszenia błędu, zgłoś raport dotyczący pseudo-pakietu *debian-live*. Zajmiemy się wtedy nim i przypiszemy do odpowiedniego pakietu.

Należy pamiętać, że błędy znalezione w dystrybucji pochodzących od Debiana (takich jak Ubuntu, i inne) **nie** powinny być zgłaszane do BTS Debiana, chyba że błędy te mogą być odtworzone w Debianie przy użyciu oficjalnych pakietów Debiana.

Styl Kodowania

15. Styl Kodowania

Rozdział ten dokumentuje styl kodowania używany w systemach live.

15.1 Kompatybilność

- Nie wolno używać składni lub semantyki, które jest unikalne dla basha. Na przykład, użycie układu konstrukcji.
- Nie używaj podzbiorów POSIX'a - na przykład, używaj `$(foo)` zamiast `'foo'`.
- Możesz sprawdzić swoje skrypty używając `'sh -n'` i `'checkbashisms'`.
- Upewnij się, że cały kod powłoki działa z `'set-e'`.

15.2 Wcięcia

- Zawsze używaj tabulatorów zamiast spacji.

15.3 Zawijanie

- Generalnie linie mają maksymalnie 80 znaków.
- Używaj zakończeń lini "typowych dla Linuxa":

Źle:

```
if foo; then
    bar
fi
```

Dobrze:

```
if foo
then
    bar
fi
```

- To samo dotyczy funkcji:

Źle:

```
Foo () {
    bar
}
```

Dobrze:

```
Foo ()
{
    bar
}
```

15.4 Zmienne

- Zmienne występują zawsze zapisane drukowanymi literami.
- Zmienne wykorzystane w *live-build* zawsze zaczynają się przedrostkiem `LB_`.
- Wewnętrzne tymczasowe zmienne w *live-build* należy rozpocząć od przedrostka `<=underscore>LB_`.

- Lokalne zmienne zaczynają się przedrostkiem *live-build* a 764
`<=underscore><=underscore>LB_`.

- Zmienne dotyczące parametrów startowych *live-config* zaczynają się od `LIVE_`.

- Wszystkie inne zmienne w *live-config* zacznij przedrostkiem `_`.

- Używaj nawiasów wokół zmiennych; na przykład napisz `${FOO}` zamiast `$FOO`.

- Zawsze chroń zmienne znakami cytatu do zachowania potencjalnych białych znaków: napisz `"${FOO}"`, a nie `${FOO}`.

`_` * Dla zachowania spójności, należy zawsze używać znaków cytatu podczas przypisywania wartości do zmiennych:

Źle:

```
F00=bar
```

Dobrze:

```
F00="bar"
```

- Jeśli zastosowane jest wiele zmiennych, przytocz całe wyrażenie:

Źle:

```
if [ -f "${F00}/foo/${BAR}/bar ]
then
    foobar
fi
```

Dobrze:

```
if [ -f "${F00}/foo/${BAR}/bar" ]
then
    foobar
fi
```

15.5 Różne

- Używaj `|` (bez otaczających wyrażenie znaków cytatu) jako rozdzielacz w zapytaniu do `sed`'a, np. `sed -e `s|'|`` (bez `""`).

- Nie używaj komendy `test` dla porównań i testów, użyj `[` `]` (bez `""`); np. `"if [-x /bin/foo]; ..."` a nie `"if test -x /bin/foo; ..."`.

`_` * Użyj `case` gdzie to jest możliwe zamiast `test`, jest to łatwiejsze do odczytania i szybsze w wykonaniu.

- Użyj nazw funkcji pisanych wielkimi literami, aby ograniczyć niepożądane działanie w środowisku użytkownika.

784 Procedury

785 16. Procedury

786 Rozdział ten dokumentuje procedury dla Projekt Systemów Live dla różnych zadań, które wymagają współpracy z innymi zespołami w Debianie.

787 16.1 Główne wydanie

788 Wydawanie nowej stabilnej wersji głównej Debiana zawiera wiele różnych zespołów pracujących razem, aby tak się stało. W pewnym momencie, na zespół live dochodzi do pewnego momentu i buduje obrazy systemów live. Wymagania, aby to zrobić to:

- 789 • Serwer lustrzany zawierający wydane wersje dla Debiana i archiwum debian-bezpieczeństwa, które mogą uzyskać dostęp buildd debian-live.
- 790 • Nazwy obrazu muszą być znane (np. debian-live-WERSJA-ARCH-FLAVOUR.iso).
- 791 • Dane z debian-cd muszą zostać zsynchronizowane (listy wykluczające udeb).
- 792 • Obrazy są budowane i przechowywane na cimage.debian.org.

793 16.2 Wydanie Docelowe

- 794 • Ponownie potrzebujemy zaktualizowanych serwerów lustrzanych Debiana i debian-security.
- 795 • Obrazy są budowane i przechowywane na cimage.debian.org.
- 796 • Wyślij wiadomość z obwieszczeniem.

797 16.2.1 Ostatnie Wydanie Docelowe Debiana

Pamiętaj, aby dostosować zarówno chroot i serwery lustrzane z pakietami binarnymi przy budowie ostatniego zestawu obrazów dla wydania Debiana po to został przeniesione z ftp.debian.org do archive.debian.org. W ten sposób stare prekompilowane obrazy live będą wciąż użyteczne bez modyfikacji dokonanych przez użytkownika.

799 16.2.2 Szablon obwieszczenia dla wydania docelowego

Email z obwieszczeniem dla wydania docelowego może być wygenerowany przy użyciu poniższego szablonu i następującego polecenia:

```
sed \
-e 's|@MAJOR@|7.0|g' \
-e 's|@MINOR@|7.0.1|g' \
-e 's|@CODENAME@|wheezy|g' \
-e 's|@ANNOUNCE@|2013/msgXXXXX.html|g'
```

Proszę dokładnie sprawdzić wiadomość przed wysłaniem i przekazać ją innym, aby dokonali korekt.

```
Updated Live @MAJOR@: @MINOR@ released
```

```
The Live Systems Project is pleased to announce the @MINOR@ update of the
Live images for the stable distribution Debian @MAJOR@ (codename "<←
@CODENAME@").
```

```
The images are available for download at:
```

```
<http://live-systems.org/cimage/release/current/>
```

and later at:

<<http://cdimage.debian.org/cdimage/release/current-live/>>

This update includes the changes of the Debian @MINOR@ release:

<<https://lists.debian.org/debian-announce/@ANNOUNCE@>>

Additionally it includes the following Live-specific changes:

- * [WPISZ TUTAJ ASPECYFICZN DLA LIVE ĘZMIAN]
- * [WPISZ TUTAJ ASPECYFICZN DLA LIVE ĘZMIAN]
- * [ŻPOWANIEJSZE PROBLEMY ĄMOG ĆWYMAGA SWOJEJ OSOBNEJ SEKCJI]

About Live Systems

The Live Systems Project produces the tools used to build official live systems and the official live images themselves for Debian.

About Debian

The Debian Project is an association of Free Software developers who volunteer their time and effort in order to produce the completely free operating system Debian.

Contact Information

For further information, please visit the Live Systems web pages at <<http://live-systems.org/>>, or contact the Live Systems team at <debian-live@lists.debian.org>.

Repozytorium Git

17. Repozytorium Git

Lista wszystkich dostępnych repozytoriów dla Projekt Systemów Live można znaleźć na stronie <http://live-systems.org/gitweb/>. Adres URL projektu git ma postać: `protocol://live-systems.org/-git/repository`. Tak więc, w celu sklonowania *live-manual*, uruchom:

```
$ git clone git://live-systems.org/git/live-manual.git
```

Lub

```
$ git clone https://live-systems.org/git/live-manual.git
```

Lub

```
$ git clone http://live-systems.org/git/live-manual.git
```

Adres do sklonowania z uprawnieniami zapisu ma postać: `git@live-systems.org:/repository`.

A zatem jeszcze raz, aby sklonować *live-manual* po ssh wpisz:

```
$ git clone git@live-systems.org:live-manual.git
```

Drzewo git składa się z wielu różnych gałęzi. Gałęzie, które szczególnie wymagają poświęcenia uwagi to **debian** i **debian-next**, ponieważ zawierają one rzeczywistą pracę, które ostatecznie będzie znajdować się w każdej nowej wersji.

Po sklonowaniu każdego z istniejących repozytoriów, będziesz w gałęzi **debian**. To jest właściwe, aby móc przyjrzeć się stanowi najnowszej wersji projektu, ale przed rozpoczęciem pracy ważne jest, aby przejść do gałęzi **debian-next**. Aby to zrobić:

```
$ git checkout debian-next
```

Gałąź **debian-next**, która nie zawsze porusza się do przodu, gdzie wszystkie zmiany są najpierw wprowadzane przed połączeniem w gałęzi **debian**. Aby dokonać analogii, to jest jak poligon doświadczalny. Jeśli pracujesz w tej branży i potrzebujesz wykonać polecenie pull (wyciągnąć), będzie trzeba użyć `git pull --rebase`, tak aby lokalne modyfikacje zostały zachowane przy wyciąganiu z serwera, a następnie Twoje zmiany zostaną wprowadzone na szczycie wszystkich innych.

17.1 Obsługa wielu repozytoriów

Jeśli masz zamiar sklonować kilka repozytoriów systemów live i chcesz przejść do gałęzi **debian-next** od razu aby sprawdzić najnowszy kod, napisać poprawkę lub przyczynić się z tłumaczeniem powinieneś wiedzieć, że serwer git zapewnia `mrconfig`. Plik, który ułatwia obsługę wielu repozytoriów. Aby z niego korzystać musisz zainstalować pakiet *mr* a po tym, uruchomić:

```
$ mr bootstrap http://live-systems.org/other/mr/mrconfig
```

822

Ta komenda automatycznie sklonuje i sprawdzi do gałęzi **debian-next** repozytorium rozwojowego pakietów Debiana wytworzonych w ramach projektu. Należą do nich, między innymi, repozytorium *live-images*, który zawiera konfiguracje, używane do gotowych obrazów, które projekt publikuje do ogólnego użytku. Aby uzyskać więcej informacji na temat korzystania z tego repozytorium, zobacz [Klonowanie konfiguracji opublikowanej przez Git](#)

Przykłady

Przykłady

18. Przykłady

W tym rozdziale omówiono przykłady budowania dla konkretnych przypadków użycia z systemów live. Jeśli jesteś nowy w budowaniu własnych obrazów systemów live, zaleca się najpierw zapoznanie z trzema kolejnymi samouczkami, a każdy z nich nauczy Cię nowych technik, które pomogą Ci używać i rozumieć pozostałe przykłady.

18.1 Używanie przykładów

Aby skorzystać z tych przykładów potrzebujesz systemu, który spełnia wymagania wymienione w [wymaganiach](#) i ma zainstalowane *live-build*, jak opisano w [instalacji live-build](#).

Należy zauważyć, że, ze względu na zwięzłość, w tych przykładach nie określono lokalnego serwera używanego do kompilacji. Można znacznie przyspieszyć pobranie przypadku korzystania z lokalnego serwera lustrzanego. Można określić te opcje podczas korzystania z *lb config*, jak opisano w [Serwery lustrzane dystrybucji używane w czasie kompilacji](#) lub dla większej wygody, ustawić domyślną opcję dla systemu kompilacji w `/etc/live/build.conf`. Wystarczy utworzyć ten plik, a w nim, ustawić odpowiednią zmienną `LB_MIRROR_*` dla preferowanego serwera lustrzanego. Wszystkie inne serwery lustrzane stosowane podczas kompilacji, będą domyślnie ustawione od tych wartości. Na przykład:

```
LB_MIRROR_BOOTSTRAP="http://mirror/debian/"
LB_MIRROR_CHROOT_SECURITY="http://mirror/debian-security/"
LB_MIRROR_CHROOT_BACKPORTS="http://mirror/debian-backports/"
```

18.2 Samouczek 1: Domyślny obraz

Przykład użycia: Stwórz pierwszy prosty obraz aby nauczyć się podstaw *live-build*.

W tym samouczku, będziemy budować domyślny obraz live ISO-hybrid zawierający tylko pakiety podstawowe (bez Xorg'a) i kilka pakietów systemu live, jako pierwsze ćwiczenie w użyciu *live-build*.

Nie można tego zrobić łatwiej niż tak:

```
$ mkdir samouczek1 ; cd samouczek1 ; lb config
```

Zbadaj zawartość katalogu `config/`, jeśli chcesz. Zobaczysz tam konfiguracje przechowywane w szkieletowych katalogach, gotowe do dostosowywania lub, w tym przypadku, użyte natychmiast, aby zbudować domyślny obraz.

A teraz jako super-użytkownik, zbuduj obraz zapisując przy tym log podczas budowania używając *tee*.

```
# lb build 2>&1 | tee build.log
```

Zakładając, że wszystko poszło dobrze, po jakimś czasie, bieżący katalog będzie zawierał `live-image-i386.hybrid.iso`. Ten hybrydowy obraz ISO można uruchomić bezpośrednio na maszynie wirtualnej, jak opisano w [Testowanie obrazu ISO z Qemu](#) i [Testowanie obrazu ISO z VirtualBox](#), lub jeszcze odpowiednio nagrany obraz na nośnikach optycznych lub urządzenia flash USB, w sposób opisany w [Nagrywanie obrazu ISO na nośniku fizycznym](#) i [Kopiowanie hybrydowego obrazu ISO na nośnik USB](#).

18.3 Samouczek 2: Narzędzie przeglądarka

Przykład użycia: Stwórz obraz z przeglądarką internetową, ucząc się jak wprowadzać modyfikacje.

W tym samouczku, stworzymy obraz odpowiedni do wykorzystania jako narzędzie przeglądarki internetowej, służący jako wstęp do dostosowywania obrazów systemu live.

```
$ mkdir samouczek2
$ cd samouczek2
$ lb config
$ echo "task-lxde-desktop iceweasel" >> config/package-lists/my.list.chroot
$ lb config
```

Nasz wybór LXDE na tym przykładzie odzwierciedla nasze pragnienie, aby zapewnić minimalne środowisko pulpitu, ponieważ celem obrazu jest jednorazowy użytek, który mamy na myśli, czyli przeglądarka internetowa. Możemy pójść dalej i zapewnić domyślną konfigurację dla przeglądarki w config/includes.chroot/etc/iceweasel/profile/, lub dodatkowe pakiety wsparcia dla wyświetlania różnego rodzaju treści internetowych, ale pozostawiamy to jako ćwiczenie dla czytelnika.

Zbuduj ponownie obraz jako super-użytkownik, zachowując log jak to opisano w <Samouczku 1>:

```
# lb build 2>&1 | tee build.log
```

Jeszcze raz, zweryfikuj czy obraz jest OK i przetestuj go jak to opisano w <Samouczku 1>.

18.4 Samouczek 3: Spersonalizowany obraz

Przykład użycia: Stwórz projekt spersonalizowanego obrazu zawierającego twoje ulubione oprogramowanie tak abyś mógł go zabrać ze sobą gdziekolwiek pójdziesz i zapisujący sukcesywnie zmiany, kiedy tego potrzebujesz oraz zmiany w konfiguracji.

Ponieważ będziemy zmieniać nasz indywidualny obraz wprowadzając wiele zmian, chcemy, śledzić te zmiany, próbując rzeczy eksperymentalnych i ewentualnie przywracając je, jeśli coś nie wyjdzie, będziemy trzymać naszą konfigurację w popularnym systemie kontroli wersji git. Będziemy również wykorzystywać najlepsze praktyki autokonfiguracji poprzez skrypty auto jak opisano w <Zarządzanie konfiguracją>.

18.4.1 Pierwsza zmiana

```
$ mkdir -p samouczek3/auto
$ cp /usr/share/doc/live-build/examples/auto/* samouczek3/auto/
$ cd samouczek3
```

Edtuj auto/config tak, aby zawierał:

```
#!/bin/sh

lb config noauto \
  --architectures i386 \
  --linux-flavours 686-pae \
  "${@}"
```

Wykonaj lb config, aby wygenerować drzewo konfiguracyjne, używając właśnie utworzonego skryptu w auto/config:

```
$ lb config
```

Teraz uzupełnij swoją lokalną listę pakietów:

```
$ echo "task-lxde-desktop iceweasel xchat" >> config/package-lists/my.list.chroot
```

Po pierwsze, `--architectures i386` zapewnia, że w naszym systemie kompilacji amd64, możemy zbudować 32-bitową wersję odpowiednią do stosowania na większości maszyn. Po drugie, możemy użyć `--linux-flavours 686-pae` bo nie przewidujemy używania tego obrazu na dużo starszych systemach. Po trzecie, wybraliśmy metapakiet zadania *lxde*, który daje nam minimalny pulpit. I w końcu, dodaliśmy dwa wstępne ulubione pakiety: *iceweasel* i *xchat*.

A teraz, zbuduj obraz:

```
# lb build
```

Należy zauważyć, że w przeciwieństwie do dwóch pierwszych samouczków, nie musimy już wpisywać `>> | tee build.log` bo jest to obecnie zawarte w `auto/build`.

Po tym jak przetestowaliśmy obraz (jak to jest w [Samouczku 1](#)) i jesteśmy zadowoleni, że działa, to jest to czas, aby zainicjować nasze repozytorium `git`, dodając tylko automatyczne skrypty przed chwilą stworzone, a następnie dokonać pierwszych zmian:

856

```
$ git init
$ cp /usr/share/doc/live-build/examples/gitignore .gitignore
$ git add .
$ git commit -m "Initial import."
```

18.4.2 Druga zmiana

865

W tej zmianie, będziemy sprzątać nasz pierwszy zbudowany obrazu, dodawać pakiet *vlc* do naszej konfiguracji, budować ponownie, testować i potwierdzać zmiany.

866

Polecenie `lb clean` oczyści wszystkie wygenerowane pliki z poprzedniej kompilacji z wyjątkiem pamięci podręcznej (cache), co oszczędza konieczności ponownego pobierania pakietów. To gwarantuje, że kolejne polecenie `lb build` ponownie uruchomić wszystkie etapy regeneracji pliki z naszej nowej konfiguracji.

867

868

```
# lb clean
```

Teraz dołączpakiet *vlc* do naszej lokalnej listy pakietów w `config/package-list/my.list.chroot`:

869

870

```
$ echo vlc >> config/package-lists/my.list.chroot
```

Zbuduj ponownie:

871

872

```
# lb build
```

Przetestuj i jeżeli jesteś usatysfakcjonowany wprowadź następną

873

zmianę:

```
$ git commit -a -m "Adding vlc media player."
```

Oczywiście, możliwe są bardziej skomplikowane zmiany w konfiguracji, prawdopodobnie dodające pliki w podkatalogach config/. Kiedy wprowadzasz nowe zmiany, tylko uważaj, aby nie edytować ręcznie lub zmieniać plików najwyższego poziomu w config zawierających zmienną LB_*, ponieważ są to także efekty budowania i są zawsze sprzątane przez lb clean i tworzone ponownie przez lb config przez odpowiednie automatyczne skrypty.

Doszliśmy do końca naszej serii samouczka. Chociaż możliwe jest o wiele więcej rodzajów personalizacji, nawet tylko za pomocą kilku funkcji pokazanych w tych prostych przykładach, może być stworzona niemal nieskończona różnorodność obrazów. Pozostałe przykłady w tym rozdziale obejmuje kilka innych przypadków użycia zaczerpnięte z zebranych doświadczeń użytkowników systemów live.

18.5 Kiosk-klient serwera VNC

Przykład użycia: Stwórz obraz za pomocą *live-build*, który podczas uruchamiania, łączy się automatycznie z serwerem VNC.

Stwórz katalog kompilacji i stwórz wewnątrz szkielet folderów konfiguracji, wyłączając zalecane, aby utworzyć minimalny system. A następnie utwórz dwie początkowe listy pakietów: pierwszą wygenerowaną ze skryptu dostarczonego przez *live-build* o nazwie Pakiety (patrz <Wygenerowane listy pakietów>), a drugą uwzględniającą pakiety *xorg*, *gdm3*, *metacity* i *xvnc4viewer*.

```
$ mkdir vnc-kiosk-client
$ cd vnc-kiosk-client
$ lb config -a i386 -k 686-pae --apt-recommends false
$ echo '! Packages Priority standard' > config/package-lists/standard.list.chroot
$ echo "xorg gdm3 metacity xvnc4viewer" > config/package-lists/my.list.chroot
```

Jak wyjaśniono w <Podkreślanie APT, w celu zaoszczędzenia miejsca> może trzeba ponownie dodać niektóre polecane pakiety do prawidłowej pracy obrazu.

Najprostszym sposobem na wypisane listy rekomendowanych pakietów jest użycie *apt-cache*. Na przykład:

```
$ apt-cache depends live-config live-boot
```

W tym przykładzie okazało się, że musimy ponownie objąć kilka pakietów zalecanych przez *live-config* i *live-boot*: *user-setup* do funkcji autologowania i *sudo* jako istotnego przy zamykaniu systemu programu. Poza tym, może być przydatne, również dodanie *live-tools*, aby móc skopiować obraz systemu do pamięci RAM i *eject*, aby ewentualnie wysunąć nośnik live. Tak więc:

```
$ echo "live-tools user-setup sudo eject" > config/package-lists/recommends.list.chroot
```

Po tym, stwórz katalog */etc/skel* w *config/includes.chroot* i umieść tam własny *.xsession* dla domyślnego użytkownika, który

będzie uruchamiał *metacity* i *xvncviewer*, podłączając się do portu 5901 na serwerze w 192.168.1.2:

```
$ mkdir -p config/includes.chroot/etc/skel
$ cat > config/includes.chroot/etc/skel/.xsession << EOF
#!/bin/sh

/usr/bin/metacity &
/usr/bin/xvncviewer 192.168.1.2:1

exit
EOF
```

Zbuduj obraz:

```
# lb build
```

Korzystaj.

18.6 Bazowy obraz dla nośnika USB z 128MB pamięci.

Przykład użycia: Stwórz domyślny obraz z usuniętymi niektórymi komponentami tak, aby zmieścił się on na nośniku USB z 128MB pamięci z pozostawieniem niewielkiej przestrzeni do wykorzystania według potrzeb.

Przy optymalizacji obrazu, aby dopasować go do określonego rozmiaru nośnika, musisz dokonać pewnych kompromisów między rozmiarem a funkcjonalnością. W tym przykładzie, przytniemy tylko tyle, aby zrobić miejsce dla dodatkowego materiału medialnego w rozmiarze 128MB, ale robienia czegokolwiek, co mogłoby zniszczyć integralność zawartych pakietów, np. czyszczenie

danych ustawień regionalnych poprzez pakiet *localepurge*, lub inne tego typu *inwazyjne* optymalizacje. Szczególnie godne uwagi, jest użycie `--debootstrap-options` by stworzyć minimalny system od podstaw.

```
$ lb config -k 486 --apt-indices false --apt-recommends false --debootstrap-  
-options "--variant=minbase" --firmware-chroot false --memtest none
```

Aby uczynić by obraz pracował prawidłowo, musimy ponownie dodać przynajmniej dwa pakiety, które zostały pominięte przez opcję `--apt-recommends false`. Zobacz [Podkreślanie APT w celu zaoszczędzenia miejsca](#)

```
$ echo "user-setup sudo" > config/package-lists/recommends.list.chroot
```

Teraz, zbuduj obraz w typowy sposób:

```
# lb build 2>&1 | tee build.log
```

Na systemie autora w momencie pisania tego dokumentu powyższa konfiguracja wyprodukowała 77MB obraz. To korzystnie w porównaniu z obrazem 177MB wyprodukowanym przez domyślną konfigurację w [Samouczku 1](#).

Największą oszczędność miejsca w porównaniu do budowania obrazu domyślnego w systemie o architekturze i386, jest wybranie typu jądra 486 zamiast domyślnego `-k "486 686-pae"`. Pominięcie indeksów APT `--apt-indices false` również oszczędza sporo miejsca, w zamian trzeba wykonać `apt-get update` przed

używaniem *apt* w systemie live. Pominięcie zalecanych pakietów `-apt-recommends false` oszczędza trochę dodatkowego miejsca, kosztem pominięcia niektórych pakietów, które mogłyby się spodziewać, że powinny się znaleźć w obrazie. Opcja `--debootstrap-options "--variant=minbase"` ładuje minimalny system na początku. Nie automatycznie zawarcie pakietów firmware z `--firmware-chroot false` również zaoszczędzą sporo miejsca. I w końcu, `--memtest none` zapobiega instalacji testera pamięci.

Uwaga: System minimalny można również osiągnąć za pomocą haków, takich jak na przykład `stripped.hook.chroot`, znajdujący się w `/usr/share/doc/live-build/examples/hooks`. To może zaoszczędzić dodatkowo małe ilości przestrzeni i stworzyć 62MB obraz. Jednak robi to przez usunięcie dokumentacji i innych plików z pakietów zainstalowanych w systemie. To narusza integralność tych pakietów, a które, jak komentarz nagłówka ostrzega, mogą prowadzić do nieprzewidzianych konsekwencji. Dlatego właśnie użycie minimalnej opcji *debootstrap* a jest zalecanym sposobem na osiągnięcie tego celu.

18.7 Pulpit GNOME w lokalnym języku oraz instalator

Przykład użycia: Stwórz obraz z pulpitem GNOME i lokalizacją dla Szwajcarii wraz z instalatorem.

Chcemy stworzyć obraz ISO-hybrydy dla architektury i386 z naszym preferowanym pulpitem, w tym przypadku GNOME, zawierającą wszystkie pakiety, które byłyby zainstalowane przez standardowy instalator Debiana dla GNOME.

Naszym początkowym problem jest odkrycie nazw odpowiednich zadań językowych. Obecnie, *live-build* nie może nam w tym pomóc. Chociaż możemy mieć szczęście i znaleźć to metodą prób i błędów,

to jest narzędzie, `grep-dctrl`, które może być użyte do ustalenia to z opisów zadań w `taskel-data` tak więc, aby przygotować się upewnij się, że masz obie te rzeczy:

```
# apt-get install dctrl-tools taskel-data
```

Teraz możemy rozpocząć wyszukiwanie odpowiedniego zadania. najpierw:

```
$ grep-dctrl -FTest-lang de /usr/share/taskel/descs/debian-tasks.desc -<->
sTask
Task: german
```

Dzięki temu poleceniu dowiadujemy się, że zadanie nazywa się po prostu niemiecki (ang. `german`). Teraz, aby znaleźć podobne zadania:

```
$ grep-dctrl -FEnhances german /usr/share/taskel/descs/debian-tasks.desc -<->
sTask
Task: german-desktop
Task: german-kde-desktop
```

W czasie startu systemu będziemy generować lokalizację **de_CH.UTF-8** i wybierać układ klawiatury **ch**. Teraz poskładajmy kawałki razem. Przypominamy sobie **«Korzystanie z metapakietów»** że metapakiety są poprzedzone przedrostkiem `task-`, po prostu określimy te parametry rozruchowe dotyczące języka, a następnie dodamy standardowe pakiety priorytetowe i wszystkie wykryte metapakiety zadań do naszej listy pakietów w następujący sposób:

```
$ mkdir live-gnome-ch
$ cd live-gnome-ch
$ lb config \
  -a i386 \
  -k 486 \
  --bootappend-live "boot=live components locales=de_CH.UTF-8 keyboard-↵
    layouts=ch" \
  --debian-installer live
$ echo '! Packages Priority standard' > config/package-lists/standard.list.↵
  chroot
$ echo task-gnome-desktop task-german task-german-desktop >> config/package↵
  -lists/desktop.list.chroot
$ echo debian-installer-launcher >> config/package-lists/installer.list.↵
  chroot
```

913

Należy pamiętać, że mamy załączony pakiet *debian-installer-launcher*, aby uruchomić instalator z pulpitu live, a także wybraną architekturę 486 jądra kernela, jak to jest obecnie konieczne, aby instalator i jądro systemu live zgadzały się aby instalator działał prawidłowo.

915 **Przewodnik redakcyjny**

916 **19. Przewodnik redakcyjny**

917 **19.1 Wytyczne dla autorów**

918 Ten rozdział zajmuje się pewnymi ogólnymi rozważaniami, które należy wziąć pod uwagę podczas pisania dokumentacji technicznej dla live-instrukcji. Są one podzielone na funkcje językowe oraz zalecane procedury.

919 **Uwaga:** Autorzy powinni najpierw przeczytać <Współtworzenie tego dokumentu>

920 **19.1.1 Funkcje językowe**

921 • *Użyj zwykłego angielskiego*

922 Należy pamiętać, że wysoki procent czytelników nie są rodzimymi użytkownikami języka angielskiego. Więc jako generalną zasadę spróbuj używać krótkich, sensownych zdań, zakończonych kropką.

923 To nie znaczy, że trzeba korzystać z uproszczonego, naiwnego stylu. Proponuje się unikać, na ile to możliwe, złożonych zdań podrzędnych, które czynią tekst trudny do zrozumienia dla nie rodzimych użytkowników języka angielskiego.

924 • *Różnorodność języka angielskiego*

925 The most widely spread varieties of English are British and American so it is very likely that most authors will use either one or the other. In a collaborative environment, the ideal variety would be “International English” but it is very difficult, not to say impossible, to decide on which variety among all the existing ones, is the best to use.

Spodziewamy się, że różne odmiany języka mogą mieszać się bez tworzenia nieporozumień, ale ogólnie należy starać się być spójnym i przed podjęciem decyzji o użyciu brytyjskiego, amerykańskiego lub innego dialektu języka angielskiego wedle uznania, proszę przyjrzeć się, jak inni ludzie piszą i postarać się ich naśladować.

• *Bądź zbalansowany*

Nie być stronniczy. Unikaj w tym odniesienia do ideologii zupełnie niepowiązanych z *live-manual*. Pismo techniczne powinny być możliwie jak najbardziej neutralne. Jak to jest w naturze piśmiennictwa naukowego.

• *Bądź poprawny politycznie*

Staraj się unikać seksistowskiego języka, jak to jest tylko możliwe. Jeśli potrzebujesz, odwołania do trzeciej osoby liczby pojedynczej najlepiej użyć jakiejś formy bezosobowej lub “wy” zamiast “on” , “ona” lub niewygodnych wynalazków, takie jak “mógłbyś/mogłabyś”, “byłem/łam” i podobnych.

• *Bądź zwięzły*

Go straight to the point and do not wander around aimlessly. Give as much information as necessary but do not give more information than necessary, this is to say, do not explain unnecessary details. Your readers are intelligent. Presume some previous knowledge on their part.

• *Oszczędź pracy tłumaczącym*

Należy pamiętać, że cokolwiek napiszesz będzie musiało zostać przetłumaczone na kilka innych języków. Oznacza to, że sporo osób, będzie musiał wykonać dodatkową pracę jeśli dodasz bezużyteczne lub nadmiarowe informacje.

• *Bądź zgodny*

As suggested before, it is almost impossible to standardize a collaborative document into a perfectly unified whole. However, every effort on your side to write in a coherent way with the rest of the authors will be appreciated.

• *Be spójny*

Use as many text-forming devices as necessary to make your text cohesive and unambiguous. (Text-forming devices are linguistic markers such as connectors).

• *Bądź opisowy*

It is preferable to describe the point in one or several paragraphs than merely using a number of sentences in a typical "changelog" style. Describe it! Your readers will appreciate it.

• *Słownik*

Look up the meaning of words in a dictionary or encyclopedia if you do not know how to express certain concepts in English. But keep in mind that a dictionary can either be your best friend or can turn into your worst enemy if you do not know how to use it correctly.

English has the largest vocabulary that exists (with over one million words). Many of these words are borrowings from other languages. When looking up the meaning of words in a bilingual dictionary the tendency of a non-native speaker of English is to choose the one that sounds more similar in their mother tongue. This often turns into an excessively formal discourse which does not sound quite natural in English.

Zgodnie z ogólną zasadą, jeśli koncepcja może być wyrażony za pomocą różnych synonimów to dobrą radą będzie wybieranie pierwszego słowa zaproponowanego przez słownik. W razie wątpliwości, często słusznym jest wybieranie słowa pochodzenia germańskiego (zwykle jednosylabowe słowa). Ostrzegamy, że te dwie techniki, mogą produkować raczej mowę nieformalną,

ale przynajmniej wybór słów będzie o szerokim zastosowaniu i ogólnie przyjęty.

Korzystanie ze słownika kolokacji jest zalecane. Są one bardzo pomocne, kiedy przychodzi do znajomości słów najczęściej występujących razem.

Ponownie; dobrą praktyką jest, aby uczyć się z pracy innych. Korzystanie z wyszukiwarki, aby sprawdzić, jak inni autorzy mogą korzystać z niektórych wyrażen może bardzo pomóc.

• *Fałszywi przyjaciele, idiomy i inne wyrażenia idiomatyczne*

Uważaj na fałszywych przyjaciół. Bez względu na to, jak biegły jesteś w obcym języku nie można pomóc wpadając od czasu do czasu w pułapkę tzw. "fałszywych przyjaciół", słowa, które wyglądają podobnie w dwóch językach, ale których znaczenie lub zastosowanie może być zupełnie inne.

Staraj się unikać idiomów w jak największym stopniu. "Idiomy" to wyrażenia, które mogą mieć znaczenie zupełnie odmienne od tego, co ich poszczególne słowa wydają się oznaczać. Czasami, idiomy, mogą być trudne do zrozumienia nawet dla rodzimych użytkowników języka angielskiego!

• *Unikaj slangu, skrótów, mowy potocznej...*

Nawet jeśli popierasz korzystanie ze zwykłego, codziennego języka angielskiego, pisanie techniczne należy do formalnej formy języka.

Staraj się unikać slangu, niepopularnych skrótów, które są trudne do zrozumienia, a przede wszystkim skrótów, które próbują naśladować język mówiony. Nie wspominając o typowych dla kanałów IRC i przyjaznych dla zamkniętych gron wyrażen.

19.1.2 Procedury

954 • *Przetestuj przed zapisaniem*

955 It is important that authors test their examples before adding them to *live-manual* to ensure that everything works as described. Testing on a clean chroot or VM can be a good starting point. Besides, it would be ideal if the tests were then carried out on different machines with different hardware to spot possible problems that may arise.

956 • *Przykłady*

957 W przypadku dostarczania przykładu spróbuj być tak dokładny jak tylko możesz. Przykład jest, mimo wszystko, tylko przykładem.

958 It is often better to use a line that only applies to a specific case than using abstractions that may confuse your readers. In this case you can provide a brief explanation of the effects of the proposed example.

959 There may be some exceptions when the example suggests using some potentially dangerous commands that, if misused, may cause data loss or other similar undesirable effects. In this case you should provide a thorough explanation of the possible side effects.

960 • *Linki zewnętrzne*

961 Links to external sites should only be used when the information on those sites is crucial when it comes to understanding a special point. Even so, try to use links to external sites as sparsely as possible. Internet links are likely to change from time to time resulting in broken links and leaving your arguments in an incomplete state.

962 Poza tym, ludzie, którzy czytają instrukcję w trybie offline nie będą

953 mogli śledzić tych linków.

- 963 • *Unikaj nadawania marki i rzeczy, które naruszają licencję zgodnie z którą podręcznik ten został opublikowany*

964 Try to avoid branding as much as possible. Keep in mind that other downstream projects might make use of the documentation you write. So you are complicating things for them if you add certain specific material.

965 *live-manual* jest oparty na licencji GNU GPL. Ma to wiele skutków, które odnoszą się do redystrybucji materiału (dowolnego rodzaju, w tym grafiki chronionej prawami autorskimi lub logo), który jest opublikowany wraz nim.

- 966 • *Napisz pierwszy szkic, przeglądaj go, edytuj, popraw i cofnij zmiany jeżeli wymaga tego sytuacja*

967 - Burza mózgów!. Najpierw musisz zorganizować swoje pomysły w logicznej kolejności zdarzeń.

968 - Kiedy już w jakiś sposób masz zorganizowane te koncepcje w głowie napisz pierwszy szkic.

969 - Dokonaj przeglądu gramatyki, składni i pisowni. Należy pamiętać, że właściwe nazwy wydań, takich jak **jessie** lub **sid** nie powinny być kapitalizowane, gdy odnosi się do nich jako nazw kodowych. Aby sprawdzić pisownię można uruchomić cel "spell". tj. polecenie `make spell`

970 - Udoskonalaj swoje wyrażenia, a jeśli to konieczne cofnij i przerób każdą część.

- 971 • *Rozdziały*

972 Use the conventional numbering system for chapters and subtitles. e.g. 1, 1.1, 1.1.1, 1.1.2 ... 1.2, 1.2.1, 1.2.2 ... 2, 2.1 ... and so on. See markup below.

973 If you have to enumerate a series of steps or stages in your

description, you can also use ordinal numbers: First, second, third ... or First, Then, After that, Finally ... Alternatively you can use bulleted items.

- *Znaczники*

And last but not least, *live-manual* uses `<SiSU>` to process the text files and produce a multiple format output. It is recommended to take a look at `<SiSU's manual>` to get familiar with its markup, or else type:

```
$ sisu --help markup
```

To są przykłady znaczników, które mogą okazać się użyteczne:

- Dla pogrubienia użyj:

```
*{foo}* lub !{foo}!
```

powoduje: **foo** lub **foo** . Użyj tego by wyszczególnić określone słowa kluczowe.

- Dla kursywy użyj:

```
/ {foo} /
```

powoduje: *foo*. Użyj tego dla np. nazw paczek Debiana.

- Dla czcionki o stałej szerokości użyj:

```
# {foo} #
```

powoduje: `foo`. Użyj tego np. dla nazw poleceń. A także aby uwidocznić poszczególne słowa kluczowe jak ścieżki dostępowe.

- Dla bloków z kodem użyj:

```
code{
    $ foo
    # bar
}code
```

powoduje:

```
$ foo
# bar
```

Użyj `code{` do otwarcia i `#}code#` do zamknięcia tagu. Ważne jest, aby pamiętać, by zostawić miejsce na początku każdej linii kodu.

19.2 Wytyczne dla tłumaczy

Ta sekcja zajmuje się pewnymi ogólnymi rozważaniami, które należy wziąć pod uwagę przy tłumaczeniu zawartości *live-manual*.

Jako ogólne zalecenie, tłumacze powinni przeczytać i zrozumieć zasady tłumaczenia, które mają zastosowanie do ich specyficznych języków. Zazwyczaj, grupy i listy dyskusyjne tłumaczeń dostarczają informacji o tym, jak tworzyć przetłumaczone prace zgodne z normami jakości Debiana.

995 **Uwaga:** Tłumacze powinni również przeczytać <Współtworzenie 1004
tego dokumentu>. W szczególności rozdział <Tłumaczenie>

996 19.2.1 Wskazówki tłumaczenia

997 • *Komentarze*

998 The role of the translator is to convey as faithfully as possible the
meaning of words, sentences, paragraphs and texts as written by
the original authors into their target language.

999 So they should refrain from adding personal comments or extra
bits of information of their own. If they want to add a comment for
other translators working on the same documents, they can leave
it in the space reserved for that. That is, the header of the strings
in the **po** files preceded by a number sign **#** . Most graphical
translation programs can automatically handle those types of
comments.

1000 • *UT, Uwagi Tłumacza*

1001 It is perfectly acceptable however, to include a word or an
expression in brackets in the translated text if, and only if, that
makes the meaning of a difficult word or expression clearer to the
reader. Inside the brackets the translator should make evident that
the addition was theirs using the abbreviation “TN” or “Translator’s
Note”.

1002 • *Wyrażenia w trybie bezosobowym*

1003 Documents written in English make an extensive use of the
impersonal form “you”. In some other languages that do not share
this characteristic, this might give the false impression that the
original texts are directly addressing the reader when they are
actually not doing so. Translators must be aware of that fact and
reflect it in their language as accurately as possible.

• *Fałszywi przyjaciele*

1005 Pułapka “fałszywych przyjaciół” wyjaśnionych wcześniej
szczególnie dotyczy tłumaczy. Dwukrotnie sprawdzić znaczenie
podejrzanych fałszywych przyjaciół w razie wątpliwości.

• *Znaczniki*

1007 Tłumacze pracujący początkowo nad plikami **pot** , a później z
plikami ***{po}** * znajdują wiele znaczników w ciągach. Mogą oni
przetłumaczyć tekst tak, jak to jest tylko możliwe do tłumaczenia,
ale niezwykle ważnym jest, aby używali oni dokładnie takich
samyh znaczników jak w oryginalnej wersji angielskiej.

• *Bloki kodowe*

1008 Mimo, że bloki z kodem są zazwyczaj nieprzetłumaczalne,
1009 zawarcie ich w tłumaczeniach jest jedynym sposobem, aby
zdobyć 100% kompletne tłumaczenie. I mimo, że oznacza to
więcej pracy na początku, bo to może wymagać interwencji
od tłumaczy jeśli kod się zmieni, to jest to najlepszy sposób,
w dłuższej perspektywie czasu na określenie, co już zostało
przetłumaczone, a co nie podczas sprawdzania integralności
plików .po.

• *Nowe linijki*

1010 Przetłumaczone teksty muszą mieć te same znaki nowej linii jak
1011 teksty oryginalne. Należy uważać, aby nacisnąć klawisz “Enter” lub
wpisać jeśli pojawiają się one w oryginalnych plikach. Znaki te często
pojawiają się, na przykład, w blokach z kodem.

1012 Nie popełniaj błędów, to nie znaczy, że przetłumaczony tekst musi
mieć taką samą długość jak w wersji angielskiej. Jest to prawie
niemożliwe.

• *Nieprzetłumaczalne ciągi znaków*

1014 Tłumacze nigdy nie powinni tłumaczyć:

- 1015 - Nazw kodowych wydań (które powinny być pisane małymi literami)
- 1016 - Nazw programów
- 1017 - Komend podanych jako przykład
- 1018 - Metadanych (zazwyczaj pomiędzy dwukropkami :**metadata**:)
- 1019 - Linków
- 1020 - Ścieżek dostępnych

SiSU Metadata, document information

Title: Podręcznik Systemów Live

Creator: Projekt Systemów Live<debian-live@lists.debian.org>

Rights: Copyright: Copyright (C) 2006-2014 Projekt Systemów Live

License: Ten program jest wolnym oprogramowaniem: możesz go rozprowadzać dalej i / lub modyfikować zgodnie z warunkami Powszechnej Licencji Publicznej GNU opublikowanej przez Free Software Foundation, według wersji 3 tej Licencji lub (według Twojego wyboru) dowolnej późniejszej wersji

Ten program jest rozpowszechniany w nadziei, że będzie użyteczny, ale BEZ JAKIEJKOLWIEK GWARANCJI; bez nawet domyślnej gwarancji PRZYDATNOŚCI HANDLOWEJ albo PRZYDATNOŚCI DO OKREŚLONYCH ZASTOSOWAŃ. Patrz GNU General Public License aby uzyskać więcej szczegółów.

Powinieneś otrzymać kopię Licencji GNU General Public License wraz z tym programem. Jeśli nie, odwiedź <<http://www.gnu.org/licenses/>>.

Pełny tekst licencji GNU General Public można znaleźć w pliku /usr/share/common-licenses/GPL-3.

Publisher: Projekt Systemów Live<debian-live@lists.debian.org>

Date: 2014-10-25

Version Information

Sourcefile: live-manual.ssm.sst

Filetype: SiSU text 2.0, UTF-8 Unicode text, with very long lines

Source Digest: SHA256(live-manual.ssm.sst)=1446ca2ac019d33bcbc8f576-c28fd7245fd2dd8bd0643d2d33183553ec84148d

Generated

Document (ao) last generated: 2014-10-25 13:18:04 +0000

Generated by: SiSU 5.7.1 of 2014w41/7 (2014-10-19)

Ruby version: ruby 2.1.3p242 (2014-09-19) [x86_64-linux-gnu]