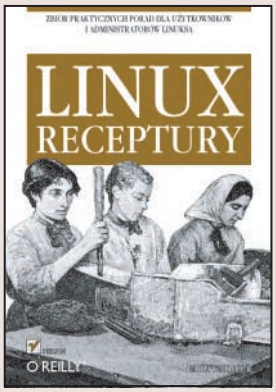
Linux. Receptury



Autor: Carla Schroder
Tłumaczenie: Adam Jarczyk
ISBN: 83-7361-879-1
Tytuł oryginału: Linux Cookbook
Format: B5, stron: 560

Zbiór praktycznych porad dla użytkowników i administratorów Linuksa

O popularności Linuksa i jej powodach napisano już bardzo dużo. Dostępny nieodpłatnie system operacyjny, stabilny, bezpieczny i możliwy do zastosowania zarówno w serwerach, jak i stacjach roboczych – brzmi to niemal jak utopia, a jednak jest prawdą. Wielu użytkowników komputerów, zachęconych opiniami na temat Linuksa, zdecydowało się na jego instalację. Pierwszy kontakt z systemem w większości sytuacji przebiega bez problemów dzięki dopracowanemu modułowi instalacyjnemu i graficznemu interfejsowi użytkownika. Problemy rozpoczynają się w momencie przejścia do bardziej zaawansowanych zagadnień. Tryb tekstowy, polecenia z dziesiątkami opcji i parametrów, pliki konfiguracyjne liczące dziesiątki linijek i trudna do zlokalizowania dokumentacja dość skutecznie odstraszają od prób poznawania tajników Linuksa.

„Linux. Receptury" to książka, dzięki której Linux wyda się mniej przerażający. Zawiera rozwiązania większości problemów mogących pojawić się podczas pracy z Linuksem. Opisuje sposoby konfigurowania systemu, administrowania nim, znajdowania informacji w dokumentacji oraz korzystania z poleceń i narzędzi systemowych. Wszystko, co może sprawiać kłopoty użytkownikowi lub administratorowi Linuksa, zostało tu wyjaśnione w jasny i czytelny sposób. Każda receptura, poza omówieniem problemu i przedstawieniem gotowego rozwiązania, zawiera także analizę, która jest bardzo pomocna przy dostosowywaniu sposobu postępowania do własnych potrzeb.

- Przeszukiwanie dokumentacji systemowej
- Instalowanie oprogramowania z pakietów RPM i z kodu źródłowego
- Wykrywanie nowego sprzętu
- Uruchamianie i zatrzymywanie systemu
- Administracja kontami użytkowników i systemem plików
- Aktualizowanie jądra systemu
- Nagrywanie płyt CD i DVD
- Konfiguracja wielosystemowa
- Tworzenie kopii zapasowych i odtwarzanie systemu
- Konfiguracja usług sieciowych
- Serwer WWW i poczty elektronicznej

Stosowanie gotowych rozwiązań najczęściej występujących problemów to oszczędność nie tylko pracy, ale i czasu.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 4.

Instalacja programów z kodu źródłowego

4.1. Wprowadzenie

Mimo wszystkich dostępnych menedżerów pakietów i narzędzi do rozwiązywania zależności bywa, że preferowana jest kompilacja z kodu źródłowego. Na przykład potrzebny nam program może być niedostępny w postaci pakietu, możemy potrzebować ścisłej kontroli nad tym, które opcje i funkcje mają być wbudowane, lub chcemy zoptymalizować program dla konkretnej architektury. Wielu doświadczonych administratorów zaleca kompilację z kodu źródłowego programów krytycznych dla bezpieczeństwa (np. ssh i wszystkie serwery).

Przy kompilacji z kodu źródłowego zaleca się przeczytanie wszystkich instrukcji. Wprawdzie procedura złożona z poleceń `configure` — `make` — `make install` jest w miarę standardowa, lecz zdarza się wiele wyjątków spowodowanych dziwactwami autorów. Często dostępnych jest wiele opcji konfiguracji, o których mówi tylko dokumentacja programu.

Jeśli ktoś woli korzystać z pakietów, to wszystkie narzędzia do budowy własnych pakietów RPM i *.deb* są dostępne za darmo. Jednakże krzywa uczenia się używania RPM lub budowy własnych *.deb* jest dość stroma. Jest jeszcze trzecia opcja — CheckInstall. Program ten jest doskonałym narzędziem pozwalającym z łatwością budować z kodu źródłowego własne pakiety RPM, Debiana lub Slackware.

4.2. Przygotowanie systemu do kompilacji programów ze źródeł

Problem

Wiemy, że do budowania programów ze źródeł potrzebny jest kompilator i, być może, jeszcze jakieś inne narzędzia, lecz nie wiemy dokładnie jakie.

Rozwiązanie

Potrzebne będą programy z dwóch kategorii:

- Niezbędne narzędzia programistyczne wspólne dla wszystkich systemów linuksowych.
- Konkretne biblioteki lub narzędzia zależne od kompilowanego programu.

Oto lista typowych narzędzi programistycznych dla Linuksa:

GNU coreutils
: Duży zbiór niezbędnych narzędzi systemowych: *shellutils* (narzędzia powłoki), *fileutils* (narzędzia plikowe) i *textutils* (narzędzia tekstowe). Pełna lista znajduje się pod adresem *http://www.gnu.org/software/coreutils/*; patrz też *info coreutils*.

GNU binutils
: Narzędzia do pracy na plikach binarnych (*http://www.gnu.org/software/binutils/*).

gcc
: GNU Compiler Collection — zbiór kompilatorów obejmujący języki C, C++, Objective-C, Fortran, Java i Ada oraz biblioteki dla nich.

GNU tar
: Narzędzie archiwizacyjne do obsługi archiwów tar kodu źródłowego (z rozszerzeniem *.tar*).

gunzip
: Narzędzie do kompresji często używane w połączeniu z tar (rozszerzenie *.tar.gz*).

bunzip2
: Format superkompresji do pakowania i rozpakowywania archiwów tar (rozszerzenie *.bz2*).

make
: To polecenie odczytuje opcje konfiguracji i inicjuje kompilację plików programu.

Dokumentacja dla kompilowanej aplikacji powinna powiedzieć o wszystkim, co jest niezbędne do pomyślnej kompilacji.

Analiza

W większości dystrybucji Linuksa dostępna jest opcja instalacji „Core Development Tools" lub podobna, więc nie musimy polować na narzędzia i instalować ich indywidualnie.

Trzeba będzie przeczytać dokumentację kompilowanej aplikacji, aby wyszukać wszelkie wymogi dla danego programu. Archiwum tar kodu źródłowego powinno zawierać pliki *README*, *INSTALL* lub inną dokumentację. Radzę przeczytać wszystko. Skrypt konfiguracyjny po uruchomieniu sprawdza w systemie, czy wszystkie potrzebne elementy są obecne. Jeśli czegokolwiek brakuje, skrypt kończy pracę i zwraca komunikat o błędzie, określając, co jeszcze jest potrzebne.

Zobacz również

- Rozdział 14. książki *LPI Linux Certification in a Nutshell* Jeffa Deana (O'Reilly)

4.3. Generowanie listy plików przy instalacji ze źródeł w celu ułatwienia deinstalacji

Problem

Chcemy wiedzieć, które pliki są instalowane w systemie przy instalacji programu z kodu źródłowego, co pozwoli znaleźć i usunąć je wszystkie, gdy zdecydujemy, że program jest już niepotrzebny. Niektórzy autorzy zapobiegliwie udostępniają opcję „`make uninstall`", co pozwala całkowicie odinstalować program, lecz wielu tak nie robi.

Rozwiązanie

Możemy za pomocą standardowych narzędzi linuksowych wygenerować listę wszystkich plików w systemie przed instalacją, a następnie listę plików po instalacji. Porównanie tych dwóch list poleceniem `diff` pozwoli zbudować listę świeżo zainstalowanych plików. Poniższy przykład opiera się na JOE (ang. *Joe's Own Editor*):

```
# find / | grep -v -e ^/proc/ -e ^/tmp/ -e ^/dev/ > joe-preinstall.list
```

Po skompilowaniu i zainstalowaniu nowego programu wygenerujemy drugą listę:

```
# find / | grep -v -e ^/proc/ -e ^/tmp/ -e ^/dev/ > joe-postinstall.list
```

a następnie za pomocą `diff` listę plików zainstalowanych przez Joe:

```
# diff joe-preinstall.list joe-postinstall.list > joe-installed.list
```

Analiza

Używając `find` w połączeniu z `grep`, możemy z łatwością wykluczyć katalogi, które nie mają znaczenia dla ostatecznej listy. Opcja `-v` polecenia `grep` odwraca wzorzec wyszukiwania, tak że są wyszukiwane wiersze niezgodne z podanym wzorcem. `-e` definiuje wzorzec wyszukiwania, natomiast ^ oznacza początek wiersza. Razem mówi to: „Ignoruj wiersze zaczynające się od…"

Nie musimy zawracać sobie głowy katalogami */proc* i */tmp*, ponieważ ich zawartość nieustannie się zmienia. Plikami */dev* zarządza system, więc również możemy je zignorować. Jest to też ważny środek bezpieczeństwa — gdy usuwamy program ręcznie, używając wygenerowanej listy różnic, */proc*, */tmp* i */dev* są katalogami, których nie powinniśmy w żadnym przypadku ruszać.

Zobacz również

- *grep(1)*, *find(1)*, *diff(1)*

4.4. Instalacja programów z kodu źródłowego

Problem

Chcemy zainstalować program z kodu źródłowego, lecz trudno jest poruszać się po gęstwinie archiwów tar, plików makefile itp.

Rozwiązanie

Najpierw należy rozpakować archiwum tar, a następnie skonfigurować (`configure`), skompilować (`make`) i zainstalować (`install`) program.

Zaczniemy pracę w katalogu mieszczącym archiwum tar i drzewo katalogów źródeł. Poniższy przykład opiera się na JOE (Joe's Own Editor):

```
# cd /usr/src/downloads
# tar zxvf joe-2.9.8.tar.gz
# cd joe-2.9.8
# ls
# less README
# less INFO
# ./configure --help
# ./configure <opcje, jeśli są potrzebne>
# make
# make install | tee joe-makeinstall
```

Ostatnie polecenie zapisuje wyjście procedury instalacji do pliku tekstowego *joe-makeinstall*.

Niektóre programy są archiwizowane za pomocą narzędzia *bzip2* zamiast bardziej tradycyjnego *gzip*. Archiwa *.bz2* rozpakowuje się następująco:

```
# tar jxvf joe-2.9.8.tar.bz2
```

Do odinstalowania programu skompilowanego z kodu źródłowego posłuży polecenie:

```
# make uninstall
```

Zadziała to tylko wtedy, jeśli autor programu przewidział opcję `make uninstall`. Przekierowanie wyjścia polecenia `make install` do pliku tekstowego pozwala zapamiętać informacje, które mogą się przydać, gdy trzeba będzie usunąć wszystkie pliki ręcznie. Listę możemy też wygenerować tak jak w recepturze 4.3.

Analiza

Kroki opisane w niniejszej recepturze są standardowym procesem instalacji programów z kodu źródłowego. Jednakże nie wszyscy autorzy programów posługują się tą samą procedurą. Przed rozpoczęciem pracy powinniśmy przejrzeć dokumentację programu.

Najważniejsze jest zapoznanie się z opcjami *configure*. Niektóre programy, np. Apache, mają dziesiątki opcji kompilacji. Z punktu widzenia bezpieczeństwa rozsądnie jest wkompilować obsługę tylko tego, co naprawdę będzie potrzebne. Jest to szczególnie ważne w przypadku serwerów mających kontakt z niezaufanymi sieciami, na przykład serwerów WWW i poczty.

Kompilacja programów z kodu źródłowego ma następujące zalety:

- Możemy skonfigurować dokładnie takie opcje, jakich nam potrzeba.
- Możemy zoptymalizować program dla konkretnej architektury systemu.
- Mamy absolutną kontrolę nad tym, co ma być zainstalowane.

Wady też się znajdą:

- Uaktualnienia i usuwanie programów może być nieprzyjemne.
- Od piekła zależności dzieli tylko jeden krok.
- Kompilacja dużego programu może trwać godzinami.

Pewne serwery powinny być kompilowane z kodu źródłowego. Na przykład serwer WWW Apache naprawdę wymaga kompilacji z kodu źródłowego, aby pozwolić na pełne dostosowanie do konkretnych potrzeb i optymalizację.

W przypadku systemów biurkowych możemy sobie to darować. Są zbyt duże i złożone. Dla nich optymalne będą wygodne dystrybucje Linuksa oparte na pakietach.

Zobacz również

- *info tar*, *make(1)*, *bzip2(1)*

4.5. Tworzenie pakietów z kodu źródłowego za pomocą CheckInstall

Problem

Chcemy stworzyć z kodu źródłowego pakiet dla systemu Slackware, Red Hat lub Debian, ponieważ dana aplikacja nie jest dostępna w postaci potrzebnego nam pakietu. Z lektury dotyczącej budowania pakietów wynika, że jest to bardzo skomplikowane zadanie. Czy jest jakiś łatwiejszy sposób?

Rozwiązanie

Posłużymy się CheckInstall. Ponownie za przykład weźmiemy Joe's Own Editor. Dla systemu Debian procedura wygląda tak:

```
# mkdir /doc-pak
# tar zxvf joe-2.9.8.tar.gz
# cd joe-2.9.8
# ./configure
# make
# checkinstall -D
```

Narzędzie CheckInstall zastępuje polecenie *make install*, musi więc zostać uruchomione w katalogu głównym drzewa katalogów kodu źródłowego. Dalej wystarczy postępować zgodnie ze wskazówkami. CheckInstall zbuduje i zainstaluje pakiet *.deb*, co możemy sprawdzić:

```
$ dpkg -l | grep joe
ii  joe          2.9.8-1     joe's own editor
```

I to wszystko. Program został zainstalowany i jest gotowy do pracy. Kopia pakietu pozostała w katalogu źródłowym.

Do zbudowania pakietu Slackware posłuży polecenie:

```
# checkinstall -S
```

Do zbudowania pakietu RPM:

```
# checkinstall -R
```

Analiza

W katalogu *doc-pak* CheckInstall umieszcza plik *README* i pozostałą dokumentację programu. Jeśli nie utworzymy tego katalogu, CheckInstall zapyta, czy chcemy utworzyć domyślny katalog na dokumentację. Jeżeli odpowiemy, że nie, pakiet nie będzie zawierać dokumentacji.

CheckInstall wykorzystuje macierzyste menedżery pakietów dla instalowanego programu: RPM w systemie Red Hat, *installpkg* w Slackware i *.apt* w Debianie. Do usunięcia pakietu CheckInstall możemy po prostu użyć menedżera pakietów z naszego systemu.

CheckInstall obsługuje wszelkie skrypty instalacyjne, na przykład:

```
# checkinstall -D make install_packages
# checkinstall -R make modules_install
# checkinstall -S install.sh
# checkinstall -D setup
```

Przypominam o przeczytaniu pliku *README* programu i innej dołączonej dokumentacji. Nie wszystkie pakiety źródłowe używają tradycyjnej procedury *configure-make-make install*. Niektóre wykorzystują skrypty instalacyjne, jak powyższe przykłady.

CheckInstall jak na razie nie pozwala na tworzenie pakietu bez automatycznego zainstalowania, lecz to może się zmienić w przyszłych wersjach.

Zobacz również

- Strona macierzysta CheckInstall (*http://asic-linux.com.mx/~izto/checkinstall/news.php*)