http://git.mdcc.cx/ad1810-doc.git
published at cpe, 28. нов 2012. 14:09:49 CET
--

Debian packaging met Subversion en debhelper

door

Joost van Baal-Ilić
<joostvb@debian.org>

lente, zomer 2012

--
La quatrième planète était celle du businessman. Cet homme était si occupé qu'il ne leva même pas la tête à l'arrivée du petit prince. «Bonjour, lui dit celui-ci. Votre cigarette est éteinte. - Trois et deux font cinq. Cinq et sept douze. [...] Pas le temps de la rallumer. Vingt-six et cinq trente et un. [...] cinq cent un millions [...] - Cinq cents millions de quoi? - Hein? Tu es toujours là? Cinq cent un million de... je ne sais plus... j'ai tellement de travail! [...] - Cinq cents millions de quoi?» répéta le petit prince --Antoine de Saint-Exupéry - Le Petit Prince, XIII, 1946

De vierde planeet was van een zakenman. Die meneer had het zo druk, dat hij zelfs niet opkeek bij de komst van de kleine prins. - Goede morgen, zei deze, uw sigaret is uitgegaan. - Drie en twee is vijf. Vijf en zeven is twaalf. [...] Geen tijd om hem weer aan te steken. Zesentwintig plus vijf is eenendertig. [...] vijfhonderd en een miljoen, zeshonderd tweeëntwintig duizend zevenhonderd eenendertig. - Vijfhonderd miljoen waarvan? - Hè? O, ben je er nog? Vijfhonderd en een miljoen... waarvan ook weer, ... ik ben het vergeten... Ik heb het zo druk! - Vijfhonderd en een miljoen waarvan? herhaalde de kleine prins (vertaald uit het Frans door Laetitia de Beaufort-van Hamel, 1951)

Inleiding

Over de auteur

Joost van Baal-Ilić werkt met Debian GNU/Linux sinds 1998, en is Debian Developer sinds 2000. Hij is getrouwd, woont in Eindhoven, is wiskundige, werkt voor de Universiteit van Tilburg als Linux Systeembeheerder en is daarnaast zelfstandig ondernemer met ad 1810 (http://www.ad1810.com/). In zijn vrije tijd doet hij aan hardlopen en verzamelt hij muziek. Zo nu en dan fotografeert hij en doet hij aan yoga. Daarnaast hebben literatuur en moderne kunst zijn warme belangstelling.

Inleiding

Als je werkt met een Debian- of Ubuntu GNU/Linux computer, dan heb je waarschijnlijk ooit "Add/Remove Programs" (PackageKit / gpk-application), of Synaptic, aptitude of apt-get gebruikt om je Linux-systeem bij de tijd te brengen. Deze programma's downloaden en installeren software in de vorm van packages. In dit document wordt uitgelegd hoe je zelf zulke packages kunt maken. Als je software wilt gebruiken waarvoor (nog) geen package is, is het handig ook die software in de vorm van zo'n package te installeren; de software integreert dan beter met je systeem. Als je zelf software schrijft, en je wilt het makkelijk maken voor jezelf en anderen, dan ligt het voor de hand je software in de vorm van een Debian package aan te bieden.

Tarballs en Debian packages

Op Debian-, Ubuntu- en sommige andere Linux-systemen wordt het Debian Package-formaat gebruikt om software te installeren en op te waarderen. (Voor Fedora- en SUSE-systemen wordt RPM gebruikt, een vergelijkbaar packagingsysteem.)

Auteurs van software ("upstream"-auteurs genoemd) leveren hun software vaak aan in een bestand dat een tarball genoemd wordt. We zullen zo'n "upstream"-tarball gebruiken om een package te maken. Toen er nog geen packaging-systemen bestonden, werd software direct vanuit de tarball geïnstalleerd; en als je software wilt gebruiken waarvoor geen package is, kun je dat nog steeds doen. Met een tarball kan software typisch gecompileerd en geïnstalleerd worden op zowel Debian-, Fedora- als bijvoorbeeld FreeBSD-systemen. Het is echter bewerkelijk om software die in zo'n vorm verpakt zit direct te installeren op een systeem. Ook is meestal veel extra software nodig (een compiler bijvoorbeeld) om de software te installeren. We converteren de tarball naar een Debian-pakket om het makkelijker te maken de software te installeren, op te waarderen en eventueel weer te verwijderen van het systeem.

We zullen ons vooral richten op "autoconfiscated" software. Dat zijn tarballs (foobar-5.3.tar.gz) die je na uitpakken installeert door "./configure && make && sudo make install" te typen. Ze zijn te herkennen aan de bestanden "configure" en "Makefile.in". Pakketten zoals bash, PHP, screen, dpkg en heel veel meer worden als autoconfiscated tarball gepubliceerd.

--

```
┌──────────────────┤ tarballs ├──────────────────┐
│                                                 │
│  tar xf caspar-20120322.tar.gz                  │
│                                                 │
│     caspar-20120322/INSTALL                     │
```

```
    caspar-20120322/Makefile.in
    caspar-20120322/configure

    caspar-20120322/COPYING
    caspar-20120322/ChangeLog

  ./configure && make && sudo make install

2/19                          http://mdcc.cx/dpsd
```

--

Voordelen van het gebruiken van een package --vergeleken met een tarball-- zijn:

- Het configureren van de software gebeurt automatisch: na installeren werkt de software out of the box (via zogeheten maintainerscripts die automatisch vóór en na installeren worden uitgevoerd).
- Als je apt-get of een variant daarvan gebruikt, dan wordt je totale systeem helemaal automatisch up to date gehouden.
- Er vindt automatische dependency-tracking plaats: als een gewenst pakket andere software nodig heeft om te kunnen functioneren, dan wordt die andere software automatisch ook geïnstalleerd.
- Van iedere software-library wordt slechts één exemplaar geïnstalleerd (d.m.v. de dependency-tracking).
- De software is cryptografisch ondertekend: er wordt dus alleen software uit vertrouwde bron geïnstalleerd (het Release-bestand is PGP-ondertekend.)
- Van elk bestand op je systeem kun je terugvinden waar het bij hoort, en dus wie ervoor verantwoordelijk is.
- Helemaal verwijderen van het software-pakket is mogelijk, zonder dat er sporen achterblijven.

Er is ook "stow": dat is een soort compromis tussen .tar.gz en .deb/.rpm. Zie [stow].

Zie de noot [softwarepackaging en systeembeheer] voor meer overwegingen over wat softwarepackaging met computersystemen doet.

--

```
                    packages vs tar.gz

automatisch configureren
automatische upgrade van totale systeem (apt-get)
automatische dependency-tracking
slechts één exemplaar van libraries
binary: geen compiler nodig
software uit gegarandeerd vertrouwde bron
duidelijke bron voor ieder bestand
de-installeren mogelijk


3/19                          http://mdcc.cx/dpsd
```

--

Naast een .deb maken we een "sourcepackage": dat wordt gebruikt om automatisch de conversieslag van tarball naar Debian-pakket uit te voeren.

De bedoeling is dat je na bestuderen van dit document in staat bent om van een upstream release (hello-5.3.tar.gz) een Debian package te maken, en om dat package bij de tijd te brengen als er een nieuwe upstream-release is. Je zult dan dus met de packaging bestanden (hello-5.3/debian/*) een ge-update-te Debian .deb (hello_5.3-1_amd64.deb) en ge-update Debian sourcepackage hello_5.3-1.debian.tar.gz maken.

Na deelname zul je als het goed is chocolade kunnen maken van de volgende twee workflows:

```
    debian
   packaging  ->  debian/{changelog,rules,control,copyright}
     VCS

  debian/*         foo_5.3-1.debian.tar.gz foo_5.3-1_amd64.changes
                -> foo_5.3-1.dsc        -> foo_5.3-1_amd64.deb
  foo-5.3.tar.gz   foo_5.3.orig.tar.gz
```

Zie [autotools] voor de workflow van de upstream-auteur, die gebruikt wordt om foo-5.3.tar.gz te maken.

We zullen gebruik maken van het Subversion-revisiecontrolesysteem om het werk te bewaren. Het Debian-pakket zal het Debhelper-systeem gebruiken, een systeem om het compileren/bouwen van zulke pakketten te automatiseren.

--

```
                  source en binary packages

 foo-5.3.tar.gz    debian/{changelog,rules,
                           control,copyright,..}
        ↓                          ↓
  Debian source package: foo_5.3-1.debian.tar.gz
                        ↓
               foo_5.3-1_amd64.deb
                        ↓
  sudo dpkg -i foo_5.3-1_amd64.deb
   sudo apt-get install foo


 4/19                          http://mdcc.cx/dpsd
```

--

Waarschuwing / Disclaimer

WAARSCHUWING: We proberen in dit document zo _snel_ mogelijk een pakket te bouwen. Hier en daar zullen we wat zaken onderbelicht laten. Als je een "net" pakket wilt maken, wat bv. kwalitatief geschikt is om met Debian GNU/Linux meegeleverd te worden, dan zul je nogal wat meer documentatie moeten lezen (en tijd moeten investeren). Een pakket wat je met dit document als handleiding maakt zal echter wel prima geschikt zijn om op je eigen systeem te draaien.

Voorkennis

Er wordt vanuit gegaan dat je bekend met het werken met een Debian- of Ubuntu-systeem bent, en vertrouwd bent met de commandline. Het is handig als je ooit zelf software hebt gecompileerd en weet wat gcc en een Makefile is, en als je ooit "handmatig" een pakket hebt geïnstalleerd ("dpkg -i pakket.deb" hebt uitgevoerd). Verder komt het van pas als je ervaring hebt met het werken met Subversion of een ander revisiecontrolesysteem.

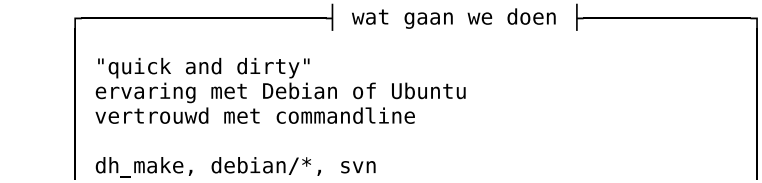
Wat gaan we doen, en wat is nodig

Als eerste zullen we, met behulp van het programma dh_make, de bestanden maken die nodig zijn om een Debian-pakket te bouwen. Daarna zullen we, in deel 2, die bestanden beheren met Subversion. In deel 3 zullen we wat extra achtergronden belichten: we zullen op een zo eenvoudig mogelijke manier een Debian-pakket bouwen, met het programma ar. Deze manier van bouwen wordt alleen uit didactische overwegingen getoond: het is _niet_ de bedoeling dat je dit gaat gebruiken om serieuze pakketten te bouwen. Je kunt dit hoofdstuk zelfs helemaal overslaan als je niet in alle details geïnteresseerd bent. Als laatste zullen we aan de slag gaan met svn-buildpackage, in deel 4: we zullen de gemaakte bestanden met Subversion beheren, en met svn-buildpackage het .deb-pakket genereren.

apt-get, dpkg en dpkg-deb

Er zijn verschillende programma's om .deb's te installeren: dpkg is een programma dat .deb's die op je bestandssysteem staan kan installeren. Het houdt verder bij welke packages geïnstalleerd zijn. apt-get, aptitude en synaptic hebben nog meer features: deze programma's kunnen .deb's zoeken op het internet, in een apt-archief. Nadat de .deb's ge-download zijn, worden ze geïnstalleerd. Een eenvoudiger programma is dpkg-deb: daarmee kun je de verschillende bestanden uit .deb's halen, zonder ze direct te installeren. Het kan echter _nog_ simpeler. In het volgende deel zullen we laten zien hoe je dat doet.

--



--

Noten

[stow] Stow is een manier om software-pakketten die vanuit .tar.gz geïnstalleerd zijn te beheren, zonder .deb of .rpm. De oudste implementatie van dit idee is GNU Stow (http://www.gnu.org/software/stow/). Later kwam XStow (http://xstow.sourceforge.net/); de meest recente implementatie is ghetto-stow (http://non-gnu.uvt.nl/pub/ghetto-stow/ , http://non-gnu.uvt.nl/debian/squeeze/ghetto-stow/).

[softwarepackaging en systeembeheer] Software packaging gaat over de manier waarop software met de rest van het systeem omgaat. Als de software dat "netjes" doet, dan is packaging makkelijk. Als de software dat "smerig" doet, en zich als een slechte gast van het systeem gedraagt, dan kan packaging de ellende beperken. Echter, het is beter om deze problemen in de software _zelf_ op te lossen; dan hoeft iemand die een .rpm maakt, niet dezelfde problemen op een andere manier op te lossen als iemand die een .deb van die software maakt. Meer hierover is te lezen op de UvT Unix Software Wishlist op http://non-gnu.uvt.nl/pub/uvt-unix-doc/software-wishlist/software-wishlist .

[autotools] Om met de GNU autotools een upstream tarball te maken, wordt de volgende workflow gebruikt:

```
             NEWS, foo.c,    sh ./bootstrap &&  ChangeLog
 upstream -> Makefile.am, -> ./configure &&  -> configure, Makefile.in
   VCS       configure.ac    make distcheck     foobar-5.3.tar.gz
```

Voor het maken van een .deb speelt dit geen rol: niet de packager maar de upstream auteur heeft te maken met deze workflow.

Deel 1: Maken van een nieuw Debian-pakket uit sources, met dh_make
==========

We gaan een script schrijven dat we aan kunnen roepen om van een tarball een .deb te maken. Dat script zal debian/rules heten. We zullen dat script in een Subversion-repository bewaren. Daarnaast stoppen we dat script in een bestand dat een "Debian sourcepackage" heet. Zo'n sourcepackage .debian.tar.gz kan gebruikt worden om automatisch van een .tar.gz een .deb te maken. Ook het genereren van een sourcepackage zal via een gestandaardiseerd interface mogelijk zijn.

Benodigde programmatuur

Om een Debian-pakket te bouwen met de instructies uit deze HOWTO, heb

je een Debian- of Ubuntu-systeem nodig met de hieronder genoemde pakketten; voer dus uit:

```
  sudo apt-get install packaging-dev svn-buildpackage
```

(Het pakket packaging-dev installeert o.a. build-essential, debhelper, devscripts en lintian.)

Verder kan het geen kwaad ook eens wat extra documentatie te lezen; die wordt o.a. door de volgende pakketten geleverd. Voer dus eventueel ook uit:

```
  sudo apt-get install debian-policy developers-reference maint-guide
```

en

```
  sudo apt-get install gnu-standards
```

Deze pakketten bevatten de 'Debian Policy Manual', de 'Debian Developer's Reference', de 'Debian New Maintainers' Guide' en, in gnu-standards, de 'GNU Coding Standards' en 'Information for Maintainers of GNU Software'.

En er zijn interessante alternatieven voor de tools die we hier zullen gebruiken, die zeker ook een blik waard zijn:

```
  sudo apt-get install cdbs git-buildpackage
```

Het pakket cdbs is in zekere zin een alternatief voor debhelper. Overigens: je _hoeft_ geen cdbs of debhelper te gebruiken; wanneer de software die je wilt packagen niet al te ingewikkeld is, dan kun je ook "handmatig" de Debian packaging uitvoeren; zie bijvoorbeeld het Debian-pakket "uruk" [uruk-pkg]. Het pakket git-buildpackage is een alternatief voor svn-buildpackage.

--

┤ benodigde programmatuur ├

```
sudo apt-get install \
 build-essential debhelper devscripts dh-make \
 lintian svn-buildpackage debian-policy \
 developers-reference maint-guide gnu-standards \
 cdbs git-buildpackage
```

6/19 http://mdcc.cx/dpsd

--

De "upstream" tarball release (een .tar.gz)

Zorg dat je de "upstream" tarball op je systeem hebt staan, bv. in /usr/local/src/hello/:

```
  cd /usr/local/src/hello/
  wget http://example.org/pad/naar/hello-5.3.tar.gz
```

Zorg verder dat de tarball ook onder de naam met een _ i.p.v. een -, en de kreet ".orig" erbij, bekend is:

```
  ln -s hello-5.3.tar.gz hello_5.3.orig.tar.gz
```

Zie ook deb-version(5).

--

┤ upstream tarball ├

```
cd /usr/local/src/hello/
wget http://example.org/pad/naar/hello-5.3.tar.gz
ln -s hello-5.3.tar.gz hello_5.3.orig.tar.gz
tar xf hello-5.3.tar.gz
cd hello-*
dh_make
```

7/19 http://mdcc.cx/dpsd

--

DEBEMAIL, Maintainer

Voer nu uit

```
  export DEBEMAIL="Jouw Naam <jij@example.org>"
```

Omgevingsvariable DEBEMAIL wordt gebruikt door o.a. de programma's dch en dh-make, die we later zullen gaan gebruiken. Je naam zal o.a. in debian/changelog terecht komen. Voeg dit export-commando toe aan je ~/.bashrc. Zorg dat de waarde overeenstemt met die van het veld Maintainer dat je later in debian/control zult gaan gebruiken.

dh_make en de bestanden debian/*

We gaan nu de bestanden die de Debian-packaging van dit pakket vormen creëren, in een directory hello-5.3/debian/. Pak eerst de tarball uit:

```
  tar xf hello-5.3.tar.gz
```

We gebruiken dh_make (zorg dat je pakket dh-make aan boord hebt), dat tooltje maakt bestanden onder debian/:

```
  cd hello-*
  dh_make

  Type of package: single binary, indep binary, multiple binary, library,
  kernel module, kernel patch or cdbs?  [s/i/m/l/k/n/b]
```

kies "s" als je software gecompileerd wordt (bv. als het C-code

bevat). Als de tarball alleen scripts bevat (PHP of Perl bv.), dan kies je "i". In dat geval zal je pakket architectuur-onafhankelijk zijn: het zelfde pakket zal op zowel amd64 als op bv. arm-computers draaien.

```
  Maintainer name : Joost van Baal-Ilić
  Email-Address   : joostvb@debian.org
  Date            : Mon, 02 Jan 2012 10:19:12 +0100
  Package Name    : hello
  Version         : 5.3
  License         : GPL
  Using dpatch    : no
  Type of Package : Single
  Hit <enter> to confirm:

Kies: <enter>
```

(Als dh_make hier zegt: "License: blank", dan kun je het nu afbreken, rm -r hello-5.3 uitvoeren, en dh_make opnieuw draaien als bv. "dh_make -c gpl".)

De meeste bestanden die nu aangemaakt zijn onder debian/ heb je vrijwel nooit nodig. Gooi die dus weg:

```
  cd debian
  rm emacsen-* init.d.ex menu.ex post* pre*
  rm *.cron.d.ex *.default.ex *.doc-base.EX watch.ex
  rm README.Debian README.source
  rm manpage.sgml.ex manpage.xml.ex
  cd ..
```

Bestanden onder debian/ die je altijd absoluut nodig hebt zijn:

```
  changelog control copyright rules
```

Van harte aanbevolen zijn verder:

```
  compat source/format
```

debian/source/format
- - - - - - - - - - -
Het bestand debian/source/format geeft weer welk Debian sourcepackage format we willen gebruiken. "Ooit" zal dat bestand verplicht worden. Sinds Debian lenny ondersteunt dpkg het "nieuwe" formaat versie 3. Zie http://wiki.debian.org/Projects/DebSrc3.0 . Vóór versie 3 hadden Debian sourcepackages een bestand zoals hello_5.3-1.diff.gz; sinds 3.0 is er een hello_5.3-1.debian.tar.gz.

debian/compat
- - - - - - -
In debian/compat staat de interface-versie van debhelper die we gebruiken.

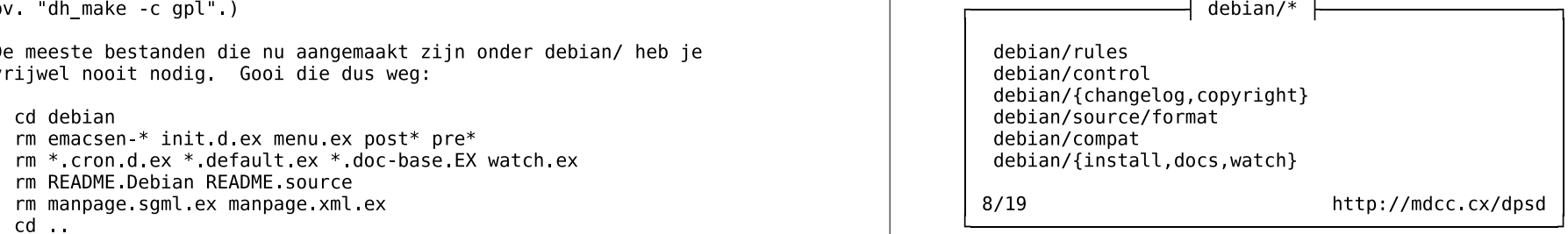
debian/rules
- - - - - -
Het belangrijkste bestand is het Makefile debian/rules. Als het goed is heeft dh-make iets voor je gemaakt dat lijkt op:

```
  #!/usr/bin/make -f
  %:
       dh $@
```

. Pas op: er staat een tab voor "dh", en het bestand debian/rules is executable.

debian/{install,docs,watch}
- - - - - - - - - - - - - - -
Soms heb je daarnaast de bestanden debian/install, debian/docs en debian/watch nodig. Bekijk andere packages voor voorbeelden over wat er dan in die bestanden moet staan. (Met "apt-get source pakketnaam" kun je de Debianized sources van een pakket downloaden.)

--



--

debian/control
- - - - - - - -
Vrijwel zeker zul je debian/copyright, debian/control en debian/changelog moeten bewerken. Laten we beginnen met debian/control.

Een voorbeeld van een (erg minimaal) bestand debian/control is:

```
  Source: ad1810-tachod
  Maintainer: Joost van Baal-Ilić <joostvb@debian.org>
  Build-Depends: debhelper (>= 7.0.50~), autotools-dev

  Package: ad1810-tachod
  Architecture: all
  Depends: ${misc:Depends}
  Description: small X11 tachograph
   ad1810-tachod annoys the user and chases her away from the computer
   when she shouldn't be working.
```

. Zie "5.2. Source package control files -- `debian/control'" in de Debian Policy Manual en "6.2. Best practices for debian/control" in de Debian Developer's Reference.

Pas debian/control dus aan:

```
  vi debian/control
```

En voer daarna uit:

```
  dch -a
```

(dch wordt meegeleverd met het devscripts-pakket.) Op deze manier kun je meteen je wijziging aan het control-file omschrijven in debian/changelog; de regels voor de syntax van debian/changelog zijn nogal strict, dch zorgt ervoor dat je geen fouten maakt bij het bewerken van dat bestand. Omgevingsvariable DEBEMAIL, die je eerder gezet hebt, wordt hier gebruikt voor het genereren van de naam en het e-mailadres in de changelog. Zorg dat de waarde van het veld Maintainer in debian/control gelijk is aan die van DEBEMAIL.

--

┤ debian/control ├
```
vi debian/control

Source: ad1810-tachod
Maintainer: J.E. van Baal-Ilić <joostvb@debian.org>
Build-Depends: debhelper (>= 7.0.50~),
 autotools-dev

Package: ad1810-tachod
Architecture: all
Depends: ${misc:Depends}
Description: small X11 tachograph
 ad1810-tachod chases the user away from computer.

9/19                            http://mdcc.cx/dpsd
```

--

debian/copyright
- - - - - - - -
Voor het formaat van debian/copyright, zie "Machine-readable debian/copyright file" uit de Debian Policy op http://www.debian.org/doc/packaging-manuals/copyright-format/ .

Voer nu uit:

```
  vi debian/copyright
  dch -a
```

--

┤ debian/{changelog,copyright} ├
```

export DEBEMAIL="Jouw Naam <jij@example.org>"
dch -a
vi debian/copyright
dch -a
(debuild -uc -us)
(sudo dpkg -i ../*.deb)

10/19                           http://mdcc.cx/dpsd
```

--

nu bouwen?
- - - - -
We zouden nu een .deb en een .debian.tar.gz kunnen bouwen. Immers, wanneer we nu aanroepen

```
  fakeroot debian/rules binary
```

(zie ook [debian/rules]), of via de wrapper debuild:

```
  debuild -uc -us
```

dan wordt het pakket gebouwd, en dat kunnen we dan installeren. Er zijn overigens ook nóg andere manieren voor; zie [dpkg-buildpackage]. Hoe dan ook: dit gaan we nu nog niet doen; we gaan eerst met Subversion aan de slag.

Noten

[dpkg-buildpackage] Als je in de directory staat waar debian/ een subdirectory van is, dan kun je een .deb bouwen door uit te voeren

```
  fakeroot debian/rules binary
```

. Onder water gebeurt er dan (vereenvoudigd) dit:

```
  fakeroot -> debian/rules -> dh.--> configure, make
                                 |`-> dh_installdocs
                                 |`-> dh_installchangelogs
                                 .`-> dh_install
                                  `-> dh_builddeb -> dpkg-deb
```

Als je ook meteen het bijbehorende Debian source-package wilt bouwen, gebruik je bv.:

```
  dpkg-buildpackage -rfakeroot -uc -us
```

Over het algemeen is het handiger om niet direct debian/rules of dpkg-buildpackage te gebruiken, maar de wrapper debuild:

```
  debuild -uc -us
```

. Dat voert onder water dan het volgende uit:

```
  debuild.--> dpkg-buildpackage.--> fakeroot   --> debian/rules
          `-> lintian           `-> dpkg-source
```

[uruk-pkg] Via "apt-get source uruk" kun je dit voorbeeld downloaden.

Hier een vereenvoudigde versie van zo'n debian/rules die geen gebruik maakt van debhelper of cdbs. Dit voorbeeld laat zien wat er allemaal "onder de motorkap" plaatsvindt om het .deb te maken.

```
 #!/usr/bin/make -f

 package=uruk

 build: build-arch build-indep

 build-arch:

 build-indep:
        ./configure --prefix=/ --datarootdir=/usr/share \
          --sysconfdir=/etc --localstatedir=/var
        $(MAKE)

 clean:
        [ ! -f Makefile ] || $(MAKE) distclean
        rm -f `find . -name "*~"`
        rm -rf debian/$(package) debian/files* debian/substvars

 binary-indep: build
        rm -rf debian/$(package)
        install -d debian/$(package)
        $(MAKE) install prefix=$(CURDIR)/debian/$(package) \
          datarootdir=$(CURDIR)/debian/$(package)/usr/share \
          sysconfdir=$(CURDIR)/debian/$(package)/etc \
          localstatedir=$(CURDIR)/debian/$(package)/var
        cp -a debian/changelog $(docdir)/changelog.Debian
        cp -a debian/copyright $(docdir)
        cp -a ChangeLog $(docdir)/changelog
        cd $(docdir) && gzip -9 changelog changelog.Debian
        gzip -r9 debian/$(package)/usr/share/man
        mkdir debian/$(package)/DEBIAN
        dpkg-gencontrol -isp -Pdebian/$(package)
        cp -a debian/conffiles debian/$(package)/DEBIAN
        for f in postinst prerm postrm; do \
          cp -a debian/$$f debian/$(package)/DEBIAN; \
          chmod a+x debian/$(package)/DEBIAN/$$f; \
        done
        chown -R root.root debian/$(package)
        chmod -R go=rX debian/$(package)
        dpkg-deb --build debian/$(package) ..

 binary-arch: build

 binary: binary-indep binary-arch

 .PHONY: build-arch build-indep binary binary-arch binary-indep clean
```

[debian/rules] De Debian Policy zegt over dit Makefile: "The following targets are required and must be implemented by `debian/rules': `clean', `binary', `binary-arch', `binary-indep', and `build'. These are the targets called by `dpkg-buildpackage'."

Deel 2: Subversion, debcommit, debuild

Een Subversion-repository

Zorg dat je schrijfrechten hebt in een Subversion-repository. Je kunt een guest-account aanvragen op Debian Alioth (http://alioth.debian.org/) en daar een repository gebruiken (zie bv. http://wiki.debian.org/Alioth/PackagingProject), of een eigen repository gebruiken, zie daarvoor [locale subversion-repository].

Zorg nu dat je een lege SVN working directory in ~/svn/packages/hello-pkg/trunk/ hebt staan.

(Zie [debcheckout] voor een tip over hoe om te gaan met packages waarvoor al Debian packaging-code in SVN zit.)

Zorg er verder voor dat ook een uitgepakte upstream tarball in je working directory staat. Registreer deze bron-bestanden echter níet in Subversion! (In Deel 4 zullen we svn-buildpackage introduceren; als je daarmee werkt hoef je je werk-directory niet te "vervuilen" met upstream bestanden.)

--

```
┌──────────────────┤ Subversion repository ├──────────┐
│                                                      │
│  cd ~; svnadmin create svnrepo                       │
│  mkdir svn; cd svn                                   │
│  svn co file:///home/jij/svnrepo; cd svnrepo         │
│  svn mkdir --parents \                               │
│             packages/hello-pkg/trunk/debian          │
│  cd packages/hello-pkg                               │
│                                                      │
│  cp /usr/local/src/hello/hello-5.3.tar.gz .          │
│  ln -s hello-5.3.tar.gz hello_5.3.orig.tar.gz        │
│  tar xf hello-5.3.tar.gz; mv hello-5.3/* trunk/      │
│                                                      │
│ 11/19                          http://mdcc.cx/dpsd   │
└──────────────────────────────────────────────────────┘
```

--

Registreren van debian/* in Subversion

Kopieer alles uit debian/ naar ~/svn/packages/hello-pkg/trunk/debian. Bijvoorbeeld:

```
  cd ~/svn/packages/hello-pkg/trunk
  svn mkdir debian
  cp -a /usr/local/src/hello/hello-5.3/debian/* debian/
  svn add debian/*
```

(en eventueel, voor expanderen van $URL:$ en $Id:$:
  svn propset svn:keywords 'Id URL Author Date Rev' debian/*
)

En commit dit:

  svn commit -m 'initial Debian packaging'

--

┤ debian/* in SVN ├

  cp -a /pad/naar/hello-5.3/debian/* debian/
  svn add debian/*
  cd ~/svn/svnrepo
  svn commit -m 'initial checkin' packages

12/19 http://mdcc.cx/dpsd

--

Goed, alles zit nu in Subversion. We kunnen nu daadwerkelijk met
SVN je bestanden in debian/ gaan beheren en een .deb gaan genereren.

Maar eerst nog iets anders: als je je pakket nu al gemaakt hebt, en
een nieuwere versie wilt packagen, dan kun je, voordat je gaat
bouwen, met uscan de nieuwe tarball downloaden; zie [uscan] en het
stukje over debian/watch, hieronder.

De "wijzig iets"-kringloop

Zorg dat je in directory ~/svn/packages/hello-pkg/trunk staat. We laten
nu zien hoe je met SVN een wijziging in een Debian-pakket aanbrengt.
Als voorbeeld leggen we de bestanden debian/source/format,
debian/watch en debian/docs aan.

debian/source/format
- - - - - - - - - -
In Deel 1 hebben we het bestand debian/source/format geïntroduceerd. Je
legt dat bestand bv. op de volgende manier aan:

  svn mkdir debian/source
  echo '3.0 (quilt)' >debian/source/format
  svn add debian/source/format
  dch 'use 3.0 dpkg source format'
  debcommit --diff && debcommit --confirm

--

┤ kringloop: vi, dch, debcommit ├

  cd ~/svn/svnrepo/packages/hello-pkg/trunk
  svn mkdir debian/source
  echo '3.0 (quilt)' >debian/source/format
  svn add debian/source/format
  dch 'use 3.0 dpkg source format'
  debcommit --diff && debcommit --confirm

13/19 http://mdcc.cx/dpsd

--

debian/watch
- - - - - -
We maken een bestand debian/watch aan; dat wordt gebruikt om
automatisch een nieuwe upstream-versie te vinden, en om die makkelijk
te kunnen downloaden, zie [uscan]. Bijvoorbeeld:

  cat <<EOT >debian/watch
  # See uscan(1) for format
  version=3
  http://mdcc.cx/pub/hello/hello-(.*)\.tar\.gz
  EOT

  svn add debian/watch

debian/docs
- - - - - -
Het bestand debian/docs wordt gebruikt door dh_installdocs: het zorgt
ervoor dat upstream bestanden zoals README en NEWS op de juiste
manier en op de juiste plek geïnstalleerd worden. In debian/docs zet
je bijvoorbeeld:

  NEWS
  README

Het bestand "TODO" kun je hier over het algemeen beter niet noemen.
Over het algemeen is de inhoud van zo'n bestand immers slechts
bruikbaar voor ontwikkelaars van de software; niet voor gebruikers.
(Overigens: ook als debian/docs niet bestaat gedraagt dh_installdocs
zich redelijk. Mogelijk heb je zelf debian/docs dus niet nodig.)

Registreer dit bestand in Subversion:

  svn add debian/docs

maintainer scripts, manpage
- - - - - - - - - - - - - -
Verdere mogelijk interessante bestanden zijn de Debian maintainer
scripts debian/{preinst,postinst,prerm,postrm}, zie daarvoor
[maintainer scripts], en de template manual page manpage.1.ex (zie
[manpage]).

--

┤ debian/*, meer ├

  debian/watch
  debian/docs
  debian/{preinst,postinst,prerm,postrm}
  manpage.1.ex

```
14/19                          http://mdcc.cx/dpsd
```

--

update changelog, commit
- - - - - - - - - - - - -
Bewerk changelog:

```
dch -a 'uscan support (debian/watch), install upstream NEWS and
 README'
```

Je kunt in 1 keer je wijziging in code en in changelog committen,
waarbij je de changelog entry als commit-message gebruikt:

```
debcommit --diff
debcommit --confirm
```

Dit herhaal je totdat je pakket naar smaak is, eventueel gecombineerd
met het genereren en testen van een .deb; zie hieronder.

Genereer een .deb

Compileer de software en genereer bv. hello_5.3-1_amd64.deb:

--

genereer een .deb

```
debuild -us -uc

lintian --pedantic -I -i ../*.changes
debc
tar -vOxf ../*.debian.tar.gz 2>&1 | less

sudo dpkg -i ../*deb

15/19                          http://mdcc.cx/dpsd
```

--

Automagisch wordt nu lintian aangeroepen; dat tooltje is als lint voor
C-code: het checkt tegen de Debian Policy. Mogelijk klaagt lintian,
bv.:

```
Command 'lintian hello/hello_5.3-1_amd64.changes' failed in
 'hello/build-area', how to continue now? [Qri?]:
```

Omdat we nu haast hebben kiezen we "i" (ignore). Later kun je, met

```
lintian --pedantic --display-info --info ../hello_5.3-1_amd64.changes
```

uitzoeken hoe het geconstateerde probleem op te lossen is. Zie ook
[debian-policy].

Dit commando herhaal je totdat je .deb naar smaak is. Met

```
debc
```

kun je controleren of de juiste bestanden door je .deb geïnstalleerd
worden. Met

```
tar -vOxf ../*.debian.tar.gz 2>&1 | less
```

kun je de inhoud van je debian-directory bekijken. Je kunt de
gegenereerde .deb ter test natuurlijk ook installeren en weer
verwijderen van je systeem.

Naast hello_5.3-1_amd64.deb worden door het bouwen ook aangemaakt:

- hello_5.3-1.debian.tar.gz (de inhoud van de debian/
  directory) en
- hello_5.3-1.dsc (een beschrijving van het "sourcepackage",
  i.e. de .debian.tar.gz + .orig.tar.gz), en
- hello_5.3-1_amd64.changes (dat wordt binnen Debian en Ubuntu
  gebruikt voor een upload naar het package-archief), en
- ../build-area/hello_5.3-1_amd64.build: een logfile van de
  build output.

De bestanden .dsc, .debian.tar.gz en .orig.tar.gz vormen het Debian
source-package, ze kunnen gebruikt worden om later eenvoudig weer
een .deb te maken, zie [dpkg-source] voor hoe dat in zijn werk gaat.
Binnen het Debian-project gebruiken autobuilders voor de verschillende
architecturen dit om automatisch een binary .deb te bouwen voor hun
architectuur. Ook kunnen andere ontwikkelaars dat gebruiken om je
pakket te bekijken, of om bugs te fixen. Verder kan het
sourcepackage gebruikt worden om je software op eenvoudige wijze
te bouwen voor een oudere Debian release ("backporten").

--

.deb en source package

```
hello_5.3-1_amd64.deb

hello_5.3.orig.tar.gz
hello_5.3-1.debian.tar.gz
hello_5.3-1.dsc

hello_5.3-1_amd64.changes
hello_5.3-1_amd64.build

16/19                          http://mdcc.cx/dpsd
```

--

Wanneer het Debian-pakket klaar is voor "release" (d.w.z.: de huidige
versie is succesvol getest en helemaal klaar), wijzig dan in het
bestand debian/changelog de term "UNRELEASED" in "unstable", bv.:

```
  hello (5.3-1) unstable; urgency=low

en commit deze wijziging:

  svn commit -m 'ship it!'

.  Nu kun je je ../*.deb installeren met

  sudo dpkg -i hello*deb.

.

Noten
-----
[debcheckout] Als je aan het werk bent met een pakket wat al in
Debian zit, en waar al Debian packaging-bestanden voor bestaan in een
Subversion-repository (of git of wat dan ook), dan kun je die
uitchecken met

  debcheckout pakketnaam

.

[uscan] Op de volgende manier kun je, als je een bestand debian/watch
hebt gemaakt, een nieuwe upstream-tarball downloaden:

  cd ~/svn/packages/hello/trunk
  uscan --destdir=../tarballs --download-current-version --verbose

(uscan wordt meegeleverd met het devscripts-pakket.)

Kijk of de nieuwe tarball gedownload wordt.  Als dat niet zo is,
bekijk dan de output om de naam van de laatste tarball te vinden.
Haal daar het versienummer uit en voorzie debian/changelog van een
nieuwe entry met die versie.  Daarna draai je uscan opnieuw, op
dezelfde manier.

[maintainer scripts] De werking van Debian maintainer scripts
(debian/{preinst,postinst,prerm,postrm}) staat beschreven op
http://wiki.debian.org/MaintainerScripts, door Margarita Manterola
(http://www.marga.com.ar/).

[manpage] Mogelijk is manpage.1.ex nuttig voor je; dat is immers een
template voor een manpage voor je applicatie, hernoem die naar
hello.1 (of hoe je binary ook heet) en schrijf die manpage dan.  Zie
bv.  "Writing Manual Pages" door Lars Wirzenius op
http://liw.fi/manpages/ voor een korte introductie.

[locale subversion-repository]  Je kunt zelf op je eigen systeem
een Subversion-repository maken en gebruiken door

  svnadmin create ~/svnrepo
  mkdir svn; cd svn
  svn co file:///home/jij/svnrepo

te doen.

En nog deze tip: svn propset:

  cd svn/packages/hello/trunk
  svn propset svn:ignore 'ChangeLog
  hello-*.tar.gz' .

(ja, er zit een newline tussen ChangeLog en hello-*.tar.gz.)

[debian-policy]  Mogelijk klaagt lintian over een probleem met
Standards-Version in je bestand debian/control.  Als in
debian/control bijvoorbeeld staat:

 Standards-Version: 3.9.2

dan betekent dat dat je claimt dat jouw pakket zich gedraagt volgens
Debian policy versie 3.9.2.  De Debian-standaarden worden meerdere
malen per jaar gewijzigd.  Het is aan te raden geregeld je pakket up
to date te brengen met de laatste policy.  Hierbij kun je de
"Upgrading Checklist" in
/usr/share/doc/debian-policy/upgrading-checklist* gebruiken.  De
Debian policy zelf is vastgelegd in
/usr/share/doc/debian-policy/policy* .

[dpkg-source]  Met een Debian source-package kan een .deb gebouwd
worden, bv. voor een andere architectuur, of gelinkt tegen andere
libraries.  Bijvoorbeeld zo:

  dget http://example.org/pool/hello_5.3-1.dsc
  dpkg-source -x hello_5.3-1.dsc
  cd hello-5.3
  debuild -uc -us


Deel 3: Handmatig bouwen van een .deb, met ar
=============================================

In dit deel wordt achtergrondinformatie gegeven over de opbouw van
een .deb-bestand, aan de hand van een voorbeeld met low-level
gereedschap.  Als je snel aan de slag wilt met Debian packaging, en
niet in _alle_ achtergronden geïnteresseerd bent, dan kun je dit
deel beter overslaan, en verder gaan met het stuk over
svn-buildpackage.

Om met .deb's om te gaan hoef je geen dpkg of dpkg-deb te gebruiken.
Je hebt zelfs geen Debian- of Ubuntu-systeem nodig.  We zullen laten
zien hoe je van "niks" een .deb kunt bouwen.  We zullen tar en ar
gebruiken.  ar wordt meegeleverd met de GNU binutils, het wordt
oorspronkelijk gebruikt om ELF-libraries samen te stellen.  Het
ar-archief-formaat wordt echter ook door dpkg gebruikt.
```

We zullen van een tarball een .deb maken. We gebruiken hier caspar-20120322.tar.gz als voorbeeld. Vrijwel iedere andere tarball werkt trouwens ook gewoon.

Download de upstream tarball en pak hem uit:

```
$ wget http://mdcc.cx/pub/caspar/caspar-20120322.tar.gz
$ tar xf caspar-20120322.tar.gz
$ cd caspar-20120322
```

Maak een directory aan waar we de input voor het .deb-je gaan neerzetten:

```
$ mkdir -p debian/tmp/DEBIAN
```

Zorg nu dat alle bestanden geïnstalleerd worden onder tmp/DEBIAN, bv. met

```
$ ./configure --prefix=/usr
$ make
$ make install DESTDIR=`pwd`/debian/tmp/DEBIAN
```

Zie [autotools] voor uitleg van wat hier gebeurt.

debian/control
- - - - - - -
Verder maak je een bestand DEBIAN/control aan:

```
$ cd debian/tmp/DEBIAN
$ head control
Package: caspar
Version: 20120322-1
Architecture: all
Maintainer: Joost van Baal-Ilić <joostvb@debian.org>
Depends: make
Description: Makefile snippets for centralized configuration management
 Caspar offers Makefile snippets for tasks like installing files you
```

(Zie Debian Policy Manual "5.3. Binary package control files -- `DEBIAN/control'" voor een beschrijving van dit bestand.) Nodig is verder het bestand DEBIAN/debian-binary:

```
$ cat debian-binary
2.0
$ cd ..
```

Nu pak je alles in, in 2 tar-archieven:

```
$ cd DEBIAN
$ tar caf ../control.tar.gz control
$ tar caf ../data.tar.gz usr
$ cd ..
```

Deze 2 tar-archieven pak je, samen met het bestandje debian-binary, in in een ar-archief:

```
$ ar vqc ../caspar_20120322-1_all.deb control.tar.gz data.tar.gz debian-binary
a - control.tar.gz
a - data.tar.gz
a - debian-binary
```

Nu heb je een Debian-package caspar_20120322-1_all.deb; dat kun je dus bekijken met dpkg-deb:

```
$ dpkg-deb --field ../caspar_20120322-1_all.deb
```

en

```
$ dpkg-deb --contents ../caspar_20120322-1_all.deb
```

. Nog interessanter is het natuurlijk om het met dpkg(1) te installeren:

```
$ sudo dpkg -i ../caspar_20120322-1_all.deb
```

Merk op dat je dus ook _zonder_ dpkg .deb's kunt installeren: bewandel de weg zoals die hier geschetst is dan in de andere richting. Zie daarvoor [no-dpkg-install].

Echter, op deze manier bouwen van een .deb kan wel, maar is over het algemeen niet de beste manier om het te doen: op deze manier maak je wel een .deb, maar krijg je geen Debian sourcepackage (.debian.tar.gz).

Zie verder de manpages deb(5) en deb-control(5) voor meer informatie.

[autotools] autoconf is een programma dat het script configure genereert. automake is een programma dat het bestand Makefile.in genereert. De programma's autoconf en automake maken deel uit van de autotools-suite. Deze suite wordt gebruikt om broncode (C, bijvoorbeeld) eenvoudig te kunnen compileren op verschillende systemen. Het gegenereerde script configure gebruik je als je vanuit een tarball de programmatuur wilt bouwen. Dit script ondersteunt veel verschillende opties, --prefix is er één van. Met die optie geef je mee onder welke directory je de verschillende gecompileerde bestanden wilt installeren. Ook het door configure gegenereerde Makefile is weer op veel verschillende manieren te gebruiken; het ondersteunt o.a. de DESTDIR variabele: DESTDIR wordt gebruikt in plaats van de root / van het bestandssysteem.

[no-dpkg-install] Installeren van een .deb-je zonder dpkg of dpkg-deb te gebruiken. Do not try this at home!

```
$ apt-get --print-uris --reinstall install caspar
$ wget http://ftp.debian.nl/debian/pool/main/c/caspar/caspar_20120530-1_all.deb
$ mkdir ~/caspar_20120322-1_all ; cd ~/caspar_20120322-1_all
```

```
  $ ar vx ../caspar_20120322-1_all.deb
  x - debian-binary
  x - control.tar.gz
  x - data.tar.gz

  $ cd /opt/caspar_20120322-1
  $ sudo tar xf ~/caspar_20120322-1_all/data.tar.gz
```

Echter, dat "wil je niet". Op deze manier wordt niet op je systeem geregistreerd dat je deze versie van dit pakket geïnstalleerd hebt. Er wordt geen rekening gehouden met afhankelijkheden. Je zult eventuele "maintainerscripts" (o.a. "postinst") moeten uitvoeren.

Trucs zoals deze zijn handig voor debugging-doeleinden, en als laatste redmiddel bij disaster recovery. En ze zijn natuurlijk leuk qua hackvalue. :)

Deel 4: Gebruik van svn-buildpackage

Het programma svn-buildpackage automatiseert sommige van de taken die gebruikt worden bij het werken aan packages. Het maakt het bijvoorbeeld makkelijk om je repository van tags te voorzien die wijzen naar eerder ge-release-de versies van je package. Verder voert het sanity checks uit. Al met al suggereert svn-buildpackage een manier van werken volgens de huidige gevolgde praktijk onder Debian-ontwikkelaars, en het beschermt je tegen vergissingen.

Om je repository geschikt te maken voor het gebruik van svn-buildpackage voer je uit:

```
  cd ~/svn/packages/hello-pkg/trunk
  svn propset mergeWithUpstream 1 debian
  cd ~/svn/packages/hello-pkg
  mkdir tarballs
  svn mkdir tags
  svn commit -m 'needed for svn-buildpackage' tags trunk
```

--

```
┌──────────────────┤ svn-buildpackage ├──────────────┐
│                                                     │
│  svn propset mergeWithUpstream 1 debian; cd ..      │
│  svn mkdir tags                                     │
│  svn commit -m 'needs svn-buildpackage' tags trunk  │
│                                                     │
│ 17/19                          http://mdcc.cx/dpsd  │
└─────────────────────────────────────────────────────┘
```

--

Voor meer documentatie over svn-buildpackage, zie /usr/share/doc/svn-buildpackage/HOWTO.html/index.html .

--

```
┌──────────────────┤ reset, tarball ├──────────────┐
│                                                   │
│  rm -r ~/svn                                      │
│  mkdir svn; cd svn                                │
│  svn co file:///home/jij/svnrepo                  │
│  cd svnrepo/packages/hello-pkg/trunk              │
│                                                   │
│  mkdir ../tarballs                                │
│  uscan --destdir=../tarballs \                    │
│   --download-current-version --verbose            │
│                                                   │
│ 18/19                          http://mdcc.cx/dpsd │
└─────────────────────────────────────────────────────┘
```

--

Je kunt je pakket nu bouwen door uit te voeren:

```
  DH_VERBOSE=1 svn-buildpackage -uc -us -rfakeroot --svn-ignore \
   --svn-reuse --svn-move --svn-lintian --svn-builder=debuild
```

--

```
┌──────────────────┤ genereer een .deb ├──────────────┐
│                                                   │
│  DH_VERBOSE=1 svn-buildpackage -uc -us \          │
│   -rfakeroot --svn-ignore --svn-reuse --svn-move \ │
│   --svn-lintian --svn-builder=debuild             │
│                                                   │
│  debc                                             │
│  ../build-area/hello_5.3-1_amd64.build            │
│                                                   │
│ 19/19                          http://mdcc.cx/dpsd │
└─────────────────────────────────────────────────────┘
```

--

Als je pakket klaar is voor release, dan "tag" je het in SVN (en bouw je het); daarvoor voer je uit:

```
  svn-buildpackage -uc -us --svn-tag -rfakeroot --svn-reuse \
   --svn-move --svn-builder=debuild
```

(Als je je release wilt ondertekenen met je PGP-sleutel:

```
  svn-buildpackage --svn-tag -rfakeroot --svn-reuse --svn-move \
   --svn-builder=debuild
```

(hier, bij wijze van toegift, een voorbeeld waarin we meer geavanceerd gebruik laten zien: en als je PGP-key 0B86B067 wilt gebruiken, en per se ook de .orig.tar.gz wilt uploaden, en alle changes vanaf > 0.2-4 in je .changes wilt laten zien, dan doe je:

```
  svn-buildpackage -k0B86B067 -sa -v0.2-4 \
   --svn-tag -rfakeroot --svn-reuse --svn-move \
   --svn-builder=debuild
```

```
) )

Het programma svn-buildpackage heeft nu, na de ge-release-de build, een
wijziging in je packaging-bestanden geschreven die nog niet in SVN
gecommit is.  Bekijk die wijziging:

  svn diff

en commit die:

  debcommit --confirm

.

Nadat je je .deb gecontroleerd hebt (debc, tar), kun je je ../*.deb
installeren met

  sudo dpkg -i ../hello*deb.

Als je je pakket niet direct via dpkg, maar met apt-get vanuit een
apt-archief wilt kunnen installeren, dan zul je je .deb (en liefst ook
je .orig.tar.gz en .debian.tar.gz) naar zo'n archief moeten uploaden.
Voor het bij de tijd brengen van de archiefindexen kun je
dpkg-scanpackages gebruiken, zie [dpkg-scanpackages].


Noten
-----
[dpkg-scanpackages]  Om zelf een apt-able repository aan te leggen,
kopiëer je packages/hello_0.2011.01.05* naar bv. /opt/pool/.  Daarna
draai je

  cd /opt
  sudo dpkg-scanpackages pool | gzip >dists/local/main/binary-i386/
Packages.gz

[multistrap]  We laten zien hoe je een nette chroot-omgeving opzet,
die je als build-omgeving kunt gebruiken.

We zullen multistrap gebruiken om een Debian-installatie in een
chroot uit te voeren.  Eerst configureren we multistrap:

  sudo cat <<EOT >/etc/multistrap.conf
[General]
arch=amd64
directory=/opt/multistrap/
noauth=false
aptsources=DebianNL
bootstrap=Debian
[Debian]
packages=apt procps bash
source=http://ftp.nl.debian.org/debian
keyring=debian-archive-keyring
suite=sid
omitdebsrc=true
EOT

Zorg dat je genoeg ruimte onder /opt/multistrap hebt; mount daar
bijvoorbeeld een van te voren aangemaakt leeg bestandssysteem.  Daarna
voeren we de installatie uit:

  sudo -i
  multistrap -f /etc/multistrap.conf
  for f in /proc /sys /dev /dev/pts; \
   do mount -o bind $f /opt/multistrap/$f; done
  chroot /opt/multistrap dpkg --configure -a

We zorgen dat je in je shell-prompt kunt zien of je al dan niet
binnen de chroot zit:

  echo sid >/opt/multistrap/etc/debian_chroot

We zorgen dat ook binnen de chroot je eigen homedirectory bruikbaar
is:

  mkdir /opt/multistrap/home/joostvb
  mount -o bind /home/joostvb /opt/multistrap/home/joostvb

en dat je user account bruikbaar is:

  chroot /opt/multistrap
  apt-get -V install adduser
  addgroup --gid 1000 joostvb
  adduser --no-create-home --uid 1000 --gid 1000 --disabled-password --
gecos 'Joost van Baal-Ilić' joostvb

Zorg dat het netwerk bruikbaar is vanuit de chroot; voer binnen
de chroot uit:

  echo 'nameserver 192.168.26.10' > /etc/resolv.conf

Nu kun je Debian-packages bouwen binnen de chroot, onder je eigen
user account:

  apt-get install build-essential debhelper devscripts lintian svn-
buildpackage
  su - joostvb
  cd ~/svn/debian-science
  svn co --depth immediates svn+ssh://svn.debian.org/svn/debian-science/
packages
  cd frog
  svn update --set-depth infinity
  cd trunk
  svn-buildpackage -uc -us --svn-tag -rfakeroot --svn-reuse --svn-move --
svn-builder=debuild

Op een machine waar je toegang tot je PGP-sleutel hebt:
```

```
  scp virbalas:frog_0.12.15-3\* .
  debsign frog_0.12.15-3_amd64.changes
  dupload --to anonymous-ftp-master frog_0.12.15-3_amd64.changes
```

Meer informatie

Extra tools en features

Er zijn veel zaken onbesproken gelaten. Voor het bouwen van een pakket kun je naast svn-buildpackage ook pbuilder gebruiken: dat bouwt je pakket vanuit een nette chroot-omgeving. Op die manier weet je zeker dat je pakket niet alleen op jouw systeem zal bouwen, maar op ieder schoon systeem. Verder worden op die manier je build-time dependencies gecheckt. Een alternatieve manier hiervoor is [multistrap]. Het programma mk-build-deps helpt je op een eenvoudige manier build dependencies te installeren. Er is een manier om netjes met wijzigingen in configuratiebestanden om te gaan: "ucf". Er is een manier om netjes je pakket door de systeembeheerder te laten configureren: "debconf". Er is een manier om pakket-configuratie-berichten in je pakket aan te bieden in de taal van de gebruiker: zie het pakket po-debconf. Er is een manier om aanmaken van cache's (locales, manpages) slechts één keer uit te voeren per systeemupgrade (en dus niet voor ieder opgewaardeerd pakket opnieuw): dpkg triggers. Al deze zaken kun je ondersteunen in je pakket.

Zie ook

Naast de Debian Policy Manual en andere documenten die we noemden onder het kopje "Benodigde programmatuur", zijn het bekijken waard:

http://ilk.uvt.nl/software-packages/build.txt en
http://video.fosdem.org/2012/crossdistro/
Debian_packaging_for_beginners.webm
en The Debian Administrator's Handbook door Raphaël Hertzog en Roland Mas:
http://static.debian-handbook.info/browse/stable/debian-packaging.html
. En ook: verschillende Debian Packaging Tutorials door Raphaël Hertzog: http://raphaelhertzog.com/debian-packaging/ .

En eventueel ook:

http://lug.project073.nl/wiki/debian-packaging/

Dank en auteursrechten

Dank

Sven van Trijp (CER), Fred Vos (http://fredvos.org), Ko van der Sloot (http://qa.debian.org/developer.php?login=ko.vandersloot), Rutger van Sleen en Sander Bos voor waardevol commentaar. René Paré en Stiching MAD, voor het mogelijk maken van de eerste op dit document gebaseerde workshop, juni 2012 in Eindhoven. Hans de Goede voor het organiseren van de Linux Packaging Workshop, juni 2012 bij RevSpace, Den Haag (https://revspace.nl/Debrpm), en Jelmer Vernooij (http://www.samba.org/~jelmer/) voor zijn hulp daarbij. De Universiteit van Tilburg voor gastheerschap bij de Debian Packaging Workshop in augustus 2012. De Nederlandse Linux Gebruikers Groep (http://www.nllgg.nl/) voor het organiseren van een Debian Packaging Workshop in oktober 2012. Jean-Paul Saman e.a. voor gastheerschap bij de packaging workshop op de T-DOSE conferentie (http://www.t-dose.org/), oktober 2012 in Eindhoven en Thijs Kinkhorst voor zijn hulp daarbij.

Auteursrechten

Copyright © 2011, 2012 Joost van Baal-Ilić
<joostvb-debian-packaging-2353@mdcc.cx>

Dit document is vrij; verspreiding en gebruik, met of zonder wijzigingen, zijn toegestaan mits bovenstaande vermelding van auteursrecht, deze voorwaarde en de volgende vrijwaring van aansprakelijkheid behouden blijven.

Dit werk wordt verspreid in de hoop dat het bruikbaar zal zijn, maar ZONDER ENIGE GARANTIE; zelfs zonder de garantie van GESCHIKTHEID VOOR EEN SPECIFIEK DOEL.

Dit document is beschikbaar op
http://mdcc.cx/debian/debian-packaging-svn-debhelper.txt en wordt tegen kostprijs door de auteur beschikbaar gesteld.

Redistribution and use, with or without modification, are permitted provided that the above copyright notice, this condition and the following disclaimer are retained.

This work is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.