# Versiebeheer en software-packages

## Waarom en hoe

Joost van Baal



# Inhoudsopgave

# 1. Inleiding

## 1.1. Status

Zo nu en dan worden er verbeteringen op dit document aangebracht. Wanneer je kopie oud is, kijk dan op http://non-gnu.uvt.nl/pub/uvt-unix-doc/packaging of er misschien een nieuwere versie is.

## 1.2. Dit document

Dit document geeft voor- (en na-)delen van het gebruik van een versiebeheersysteem en het maken van packages bij het ontwikkelen en opleveren van software voor Unix-achtige systemen. Het is primair bedoeld voor programmeurs die software aanleveren in een enigszins geformaliseerde omgeving (zoals bijvoorbeeld een IT-departement van een Nederlandse Universiteit). Overigens zullen ook systeembeheerders nuttige informatie uit deze tekst kunnen halen. De tekst is rijkelijk voorzien van voorbeelden uit de praktijk binnen het Unix-team van de Universiteit van Tilburg, zodat je zo snel mogelijk aan de slag kunt.

Er wordt vanuit gegaan dat de lezer een beetje bekend is met (GNU) Make (http://www.gnu.org/software/make/), met de Unix Bourne shell, en een beetje de weg weet te vinden in filesystemen zoals die op Unix-achtige machines te vinden zijn. Verder helpt het als de kreten 'tarball', 'commit message', 'bestandspermissies' en 'file ownership' je iets zeggen.

Het document maakt deel uit van een presentatie die de auteur in februari 2007 verzorgde voor software-ontwikkelaars en systeembeheerders (vooral van de dienst Library and IT Services) van de Universiteit van Tilburg en van een presentatie voor een algemeen publiek in Eindhoven, voor Enosig (http://enosig.org/).

Dit document wordt gepubliceerd op http://non-gnu.uvt.nl/pub/uvt-unix-doc/packaging. Updates zullen aldaar te vinden zijn. (Het wordt onderhouden via SVN, in `$URL: https://infix.uvt.nl/its-id/trunk/sources/uvt-unix-doc/packaging/packaging.dbx $`.)

Dank aan Hans van den Dool voor suggesties.

Opmerkingen over dit document kunnen gestuurd worden naar de auteur: <`joostvb+packaging@uvt.nl`>.

## 1.3. Waarom?

Als je een systeem beheert, is het noodzakelijk dat van ieder bestand op het systeem uit te vinden is waar het vandaan komt, en wie ervoor verantwoordelijk is. Verder helpt het heel erg als van ieder bestand iets van de geschiedenis te achterhalen is. Zonder deze informatie is het onmogelijk systeembeheer te doen: problemen met performance zijn dan nauwelijks op te lossen en de veiligheid en beschikbaarheid van het systeem is niet te garanderen over de langere termijn.

Verder is het handig wanneer software die op verschillende systemen geïnstalleerd moet worden, gegarandeerd exact hetzelfde is. Ook het feit dat alle benodigde programma's en documentatie in één tarball verpakt zijn is handig: op deze manier is de software eenvoudig op een tweede systeem te installeren, bijvoorbeeld om tests uit te voeren. Packaging van software leidt tot een meer gestandaardiseerde manier van werken, en een meer gestandaardiseerde manier om de software te bouwen en installeren. En dat maakt het weer makkelijker om het beheer van een package over te dragen aan een andere persoon. (Of aan jezelf, als je twee jaar later je werk weer terug ziet.)

De meeste bestanden die op een Unix computersysteem te vinden zijn worden door de leverancier van het besturingssysteem aangeleverd, en worden via een met het besturingssysteem meegeleverd package-management-systeem up to date gehouden. Maar ook voor programmatuur en configuratie van locaal ontwikkelde software zal het beheer duidelijk moeten zijn. Verder zullen er directories moeten zijn waar software die niet via het package-management-systeem beheerd wordt, geïnstalleerd kan worden zonder dat die dat package-management-systeem in de weg zit.

# 2. De omgeving: mensen en systemen

## 2.1. Packages

Met packages bedoelen we hier een groep bestanden met instructies over de manier waarop die bestanden op een systeem neer te zetten (te installeren) is. Een package definieert een interface tussen programmeurs (leveranciers van de software) en systeembeheerders (zij die de software installeren).

Typisch is de groep bestanden in een package verpakt in een `.tar.gz` bestand (ook wel 'tarball' genoemd), en is er een `Makefile` of ander script met door machines uit te voeren instructies voor installatie. Maar ook een revisie van een (deel van een) Subversion repository met een `Makefile` daarin kan een package-achtig interface bieden.

Locale software, die niet met het besturingssysteem wordt meegeleverd, wordt over het algemeen onder `/usr/local` en/of `/opt` geïnstalleerd. Software die door de leveranciers van het besturingssysteem wordt aangeleverd, schrijft (bij systemen die zich aan de Filesystem Hierarchy Standard (http://www.pathname.com/fhs/) houden) gegarandeerd niet onder die directories. Het beheer van de locale sofware onder `/usr/local` en/of `/opt` kan dan via versiebeheer en locale packaging gebeuren.

## 2.2. Rollen van mensen en systemen

In het soort omgeving waar we het hier over hebben (op de Universiteit van Tilburg werken minder dan 5 fulltime Unix systeembeheerders die ongeveer 100 systemen beheren; de locale software wordt aangeleverd en onderhouden door ongeveer 10 programmeurs) worden verschillende rollen gespeeld door verschillende mensen en computersystemen. De volgende rollen van mensen treffen we aan:

Programmeur
: schrijft programma's

Technisch Beheerder
: installeert programma's

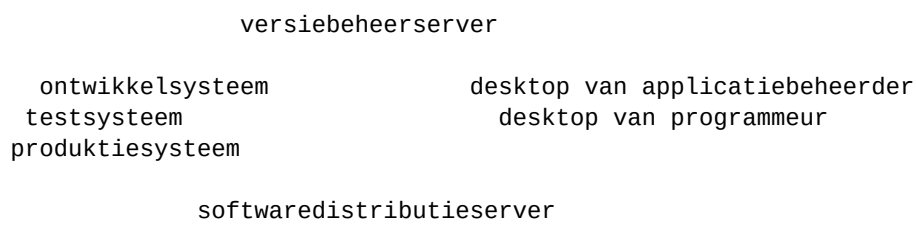
Applicatiebeheerder
: edit configuratie-bestanden (en installeert ze mogelijk)

Gebruiker
: schrijft niet, maar leest

Vaak wordt de rol van Technisch Beheerder vervuld door de Systeembeheerder: die zorgt dat dus voor het installeren en upgraden van zowel lokale programmatuur als van programmatuur die bij het besturingsysteem hoort. Sommigen gebruiken de naam Functioneel Beheerder voor Applicatiebeheerder.

Het is zinvol van een versiebeheersysteem gebruik te maken bij het schrijven en onderhouden van software, en het beheren van configuratiebestanden. Met een versiebeheersysteem bedoelen we iets als Subversion (http://subversion.tigris.org/). Versiebeheersystemen maken het mogelijk met een groep van mensen aan dezelfde bestanden te werken, oudere versies van bestanden te bekijken, en vast te leggen waarom en wanneer welke wijziging door wie op welk bestand uitgevoerd is.

Wanneer ook versiebeheer gebruikt wordt, komt het plaatje van de verschillende computersystemen die een rol spelen bij het process er ongeveer zo uit te zien.




De bedoeling van deze tekst is (o.a.) dat je een beeld krijgt hoe deze verschillende systemen in elkaar grijpen. In het kort komt het hier op neer:

Zowel op de desktop van de programmeur als op de desktop van de applicatiebeheerder zal een werkkopie van een versiecontroleboom staan, die geregeld gesynchroniseerd wordt met de versiecontrolerepository op de versiebeheerserver. De boom waar de programmeur mee werkt zal programmacode bevatten. De programmeur gebruikt het ontwikkelsysteem en het testsysteem om aan de code te werken. Wanneer de code klaar is voor produktie, zal op de desktop van de programmeur typisch make dist wordt uitgevoerd, waarna een 'tarball' naar de softwaredistributieserver wordt gecopieerd. De systeembeheerder copieert die tarball van de softwaredistributieserver naar het produktiesysteem, en installeert die daar.

De boom waar de applicatiebeheerder mee werkt zal configuratiebestanden bevatten. De bestanden worden getest op het testsysteem. Wanneer de configuratiebestanden klaar zijn voor productie, zullen de configuratiebestanden (b.v. met caspar) naar het produktiesysteem gecopieerd worden.

# 3. Versiebeheer

## 3.1. Versiebeheersystemen

Voorbeelden van populaire versiebeheersystemen zijn, naast Subversion, CVS (http://www.nongnu.org/cvs/), Bazaar (http://bazaar-vcs.org/), Git (http://git.or.cz/) en darcs (http://abridgegame.org/darcs/). Zie ook de sectie Version-Control Systems (http://www.catb.org/~esr/writings/taoup/html/ch15s05.html) in 'The Art of Unix Programming'.

Mensen die ervaring hebben met CVS, kunnen over het algemeen snel aan de slag met het modernere Subversion. Zie Paragraaf 8 voor documentatie.

## 3.2. Opbouw van een versiebeheerboom

Wanneer je zowel programmacode als configuratiedata wilt beheren, dan is een mogelijke manier om je versiebeheerboom in te richten:

```
systems/<hostnaam>/
sources/<projectnaam>/
doc/<...>/<document>
```

Van ieder bestand dat in SVN moet (komen te) staan, zou je dan de volgende vragen moeten kunnen beantwoorden; de antwoorden bepalen de plaats in de boom:

- Zijn het systeembestanden of is het documentatie, specifiek voor één host? In dat geval stop je het in `systems/<hostnaam>`. (Bijvoorbeeld onder `systems/<hostnaam>/etc/` of `systems/<hostnaam>/doc/`.)
- Is het programmacode, of documentatie specifiek over programmatuur van één project? In dat geval stop je het in `sources/<projectnaam>/`.
- Is het andere documentatie? In dat geval stop je het in `doc/<...>/<document>`.

Eventueel kun je naast `systems/` ook nog `system-group/` aanleggen. Voor bestanden, specifiek voor één beperkte groep van hosts (die bijvoorbeeld een cluster-achtige omgeving vormen), kun je de bestanden dan onder `system-group/<systeemgroepnaam>` bewaren.

Het is aan te raden zo'n overzichtje van de opbouw van de SVN-boom in een bestand `README` in de stam van iedere boom te stoppen.

Verder is het aan te raden een e-maillijst in te richten waar commit-berichten heen gestuurd worden, eventueel vergezeld van de bijbehorende wijzigingen in diff(1) formaat. Uiteraard dienen de commit-ers op de hoogte te zijn van het bestaan van zo'n lijst.

# 4. Packaging, installeerbaarheid en `Makefile`'s

Wanneer je je software wilt voorzien van machine-leesbare installatie-instructies, dan kun je dat op verschillende manieren doen. Het ligt voor de hand een `Makefile` te schrijven. (Zie hiervoor b.v. ook de secties make: Automating Your Recipes (http://www.catb.org/~esr/writings/taoup/html/ch15s04.html) en Generating Makefiles (http://www.catb.org/~esr/writings/taoup/html/ch15s04.html#id2987644) in 'The Art of Unix Programming'.)

Het simpelste `Makefile` dat je zou kunnen meeleveren ziet er ongeveer zo uit:

```
install:
```

```
install hello.pl /usr/local/bin/hello
```

Nadeel hiervan is dat het onmogelijk is om automatisch een installatie op een niet-standaard plaats te doen. Later, in Paragraaf 5.7, zullen we zien waarom dat soms toch wel erg gewenst is.

In Bijlage A kun je een voorbeeld zien van een `Makefile` dat dat wel mogelijk maakt. Helaas loopt het onderhoud van zo'n `Makefile` al gauw de klauwen uit.

Een andere manier om dit probleem op te lossen is de GNU Autotools te gebruiken. Niet iedereen is even enthousiast over die tools. Er zijn nadelen:

1. **oud.** Misschien dat je zelfs zou kunnen zeggen: de Autotools zijn verouderd.
2. **groot en complex.** Je package zal immers op ieder Unix-achtig systeem te installeren zijn. /bin/sh en Make gedragen zich op iedere Unix weer net even anders.

Er zijn echter ook voordelen:

1. **oud.** Je zelfs kunnen zeggen: 'proven technology'. Er is dus veel kennis over beschikbaar.
2. **portable zonder dependencies.** Je package is te installeren op ieder Unix-achtig systeem (dat wil zeggen: als je de Autotools goed gebruikt, kun je dat voor elkaar krijgen), zonder dat niet-standaard software nodig is.

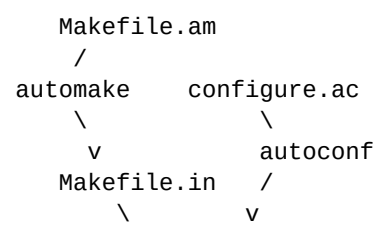
En natuurlijk is het zo dat de Autotools in de eerste plaats voor C-programma's geschreven zijn. Gebruik van Autotools om zaken als shell- of Perl-scripts te packagen zou dus als overkill gezien kunnen worden.

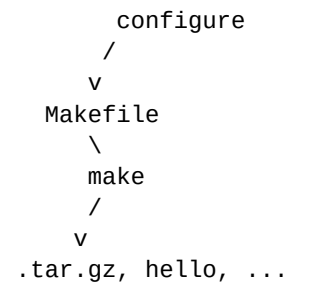
# 5. Autoconfiscation en xstow

## 5.1. Inleiding

We geven hier een voorbeeld: we laten zien hoe je software 'autoconfiscate (http://foldoc.org/?autoconfiscate)'. Autoconfiscaten van software is die software voorzien van extra bestanden zodat **autoconf**, **automake** (en **libtool**) gebruikt kunnen worden om die sofware te compileren. Omdat ook het genereren van een installeerbare tarball makkelijk wordt gemaakt, is autoconfiscation ook handig voor software die niet gecompileerd hoeft te worden. We zullen hier een klein project op een 'quick and dirty' manier onder handen nemen; we zullen niet al te diep in de details gaan duiken. (In Paragraaf 8 staat documentatie genoemd waarin de details worden uitgelegd.)

Voor nu volstaan we met het volgende vereenvoudigde schema waarin in- en uitvoer van **automake** **autoconf**, **configure** en **make** gegegeven worden:






Het voorbeeld gaat over 'uvt-sum': twee heel kleine shellscriptjes. Je kunt het resultaat ook op de uvt-sum homepage (http://non-gnu.uvt.nl/pub/uvt-sum/) bekijken.

## 5.2. Autoconfication: bouwen van een uvt-sum tarball vanuit SVN

We beginnen met de volgende bestanden:

```
uvt-sum/man/sum-recv.pod
uvt-sum/man/sum-send.pod
uvt-sum/script/sum-recv
uvt-sum/script/sum-send
```

`man/sum-recv.pod` is een manpage in Perl's Plain Old Documentation formaat (zie perlpod(1)). We willen dat de scriptjes sum-recv en sum-send in `/usr/local/bin/` geïnstalleerd worden, dat manpages gegenereerd worden, en dat die in `/usr/local/man/man1/` terecht komen zodat man(1) ze kan vinden. Verder willen we natuurlijk dat `--prefix` en `DESTDIR` ondersteund worden, zodat mensen installaties onder niet-standaard plaatsen kunnen uitvoeren, en dat er op eenvoudige wijze binary packages gemaakt kunnen worden.

We beginnen een bestand `configure.ac` aan te maken, voor **autoconf**. Dat doe je zo:

```
joostvb@banach:~/uvt-sum% autoscan
joostvb@banach:~/uvt-sum% mv configure.scan configure.ac
joostvb@banach:~/uvt-sum% rm autoscan.log
joostvb@banach:~/uvt-sum% vi configure.ac
```

Aan het gegenereerde bestand `configure.ac` voeg je `AM_INIT_AUTOMAKE` en `AC_CONFIG_FILES` toe, en je vult de templates in `AC_INIT` in. Je maakt dan dus bijvoorbeeld:

```
AC_PREREQ(2.61)
AC_INIT([uvt-sum], [0.1], [joostvb+sum@uvt.nl])
AM_INIT_AUTOMAKE
AC_CONFIG_FILES([Makefile
                 man/Makefile
                 script/Makefile])
```

```
AC_OUTPUT
```

De macro `AC_OUTPUT` zorgt ervoor dat `Makefile`'s en andere bestanden gemaakt gaan worden door **./configure**. In `AC_CONFIG_FILES` geef je aan welke bestanden dat precies zijn: alle bestanden die je hier noemt, zullen uit hun `.in`-broertje gemaakt worden. Zie de Autoconf Macro Index (http://www.gnu.org/software/autoconf/manual/autoconf.html#Autoconf-Macro-Index) voor een uitputtende lijst van de beschikbare macro's.

Vervolgens gaan we bestanden `Makefile.am` voor **automake** aanmaken. Maak in iedere directory zo'n file: `uvt-sum/Makefile.am` bestaat uit één regel: `SUBDIRS = script man`. Het bestand `man/Makefile.am` is:

```
man_MANS = sum-recv.1 sum-send.1
SUFFIXES = .pod .1

.pod.1:
	pod2man --center='$(PACKAGE)' --release='$(VERSION)' $< $@

EXTRA_DIST = sum-recv.pod sum-send.pod
CLEANFILES= $(man_MANS)
```

en `script/Makefile.am` is:

```
bin_SCRIPTS = sum-recv sum-send
EXTRA_DIST = $(bin_SCRIPTS)
```

In `EXTRA_DIST` zet je bestanden die je mee wilt leveren in je tarball (en die **automake** niet zelf al erin stopt). Merk op dat er dus vier groepen van bestanden zijn: 1: bestanden die in je werkdirectory staan; 2: bestanden die daarnaast ook in SVN zitten; 3: bestanden die in je tarball terecht gaan komen; en 4: bestanden die uiteindelijk op het systeem geïnstalleerd gaan worden.

**automake** wil erg graag dat bepaalde bestanden in de werkdirectory aanwezig zijn (zie de GNU coding standards (http://www.gnu.org/prep/standards/) als je wilt weten waarom). Voor nu kunnen we die bestanden vrijwel leeg laten.

```
joostvb@banach:~/uvt-sum% echo "See ChangeLog" > NEWS
joostvb@banach:~/uvt-sum% touch README ChangeLog
joostvb@banach:~/uvt-sum% echo "Joost van Baal" >  AUTHORS
```

Later zullen we relevante inhoud in deze bestanden zetten.

Als je je software onder een andere licentie dan de GNU General Public License wilt uitbrengen (dat is immers de **automake** default), doe je verder

```
joostvb@banach:~/uvt-sum% vi COPYING
```

Nu al deze bestanden aanwezig zijn, kunnen we de volgende commando's uitvoeren.

```
joostvb@banach:~/uvt-sum% aclocal
joostvb@banach:~/uvt-sum% automake --add-missing
joostvb@banach:~/uvt-sum% autoreconf
joostvb@banach:~/uvt-sum% ./configure
joostvb@banach:~/uvt-sum% make distcheck
```

(Draaien van **aclocal**, **automake** en **autoreconf** zorgt dat bestanden als `Makefile.in`'s en `configure` uit `Makefile.am`'s en `configure.ac` gemaakt worden. Aanroepen van **./configure** zorgt ervoor dat `Makefile`'s uit `Makefile.in`'s gemaakt worden. Zie de schema's in C. Generated File Dependencies (http://mdcc.cx/pub/autobook/autobook-latest/html/autobook_276.html#SEC276) en verder in het Autobook voor een uitputtend overzicht.)

Als het goed is, levert het laatste commando op:

```
============================================
uvt-sum-0.1 archives ready for distribution:
uvt-sum-0.1.tar.gz
============================================
```

Je kunt het gegenereerde bestand `uvt-sum-0.1.tar.gz` nu beschikbaar stellen aan systeembeheerders, die de software dan kunnen installeren. De software zal nu op de standaard manier (

```
tar zxf *tar.gz; cd *-*;
./configure; make; sudo make install
```

) te installeren zijn.

## 5.3. Onderhoud van `configure.ac` en `Makefile.am`'s

Wanneer je nu een bestand wijzigt, kun je daarna weer `make distcheck` draaien: alle autotools die opnieuw aangeroepen moeten worden, worden dan automatisch gedraaid. In geval van nood kun je **autoreconf** draaien, of, nog drastischer, `make maintainer-clean-recursive`. En als zelfs *dat* niet doet wat je wilt, kun je altijd `rm -r uvt-sum` doen, en een verse SVN checkout ophalen (zorg er in dat geval natuurlijk wel voor dat al je handgemaakte bestanden in SVN zitten!)

Je kunt ook nog een bestand `bootstrap` aanmaken: mensen verwachten daarin instructies te vinden die uitgevoerd dienen te worden wanneer vanuit het versiecontrolesysteem een distributie gebouwd dient te worden:

```
joostvb@banach:~/uvt-sum% echo "aclocal; automake --add-missing; autoreconf" \
  > bootstrap
```

(Zo'n bestand `bootstrap` wordt soms ook wel `autogen.sh` genoemd.)

De files die je nu via je versiecontrolesysteem gaat beheren (en die je dus aan `svn add` zult voeren) zijn, naast `man/sum-recv.pod`, `man/sum-send.pod` en `script/sum-recv`, `script/sum-send` de bestanden

```
bootstrap
ChangeLog
Makefile.am
NEWS
README
configure.ac

man/Makefile.am
script/Makefile.am
```

en, als je niet de GPL gebruikt, `COPYING`. Alle andere files die je nu in je working directory vindt, kun je verder negeren.

Merk op dat `configure.ac` en de `Makefile.am`'s veel kleiner zijn dan het `Makefile` uit Bijlage A. De andere bestanden die nu in je versiecontroleboom staan (`INSTALL`, `ChangeLog` etc.), zijn sowieso handig voor gebruikers van je sofware. Eigenlijk had je die al moeten onderhouden...

Het tooltje authello (http://non-gnu.uvt.nl/pub/authello/) van Anton Sluijtman probeert dit proces voor je te automatiseren; het is geschreven om Perl projecten the autoconfiscaten. Een uitgebreidere tool, die meer op het autoconfiscaten van C-code is gericht, is autoproject (http://packages.debian.org/autoproject).

## 5.4. Suggesties voor verbeteringen

Bovenstaand voorbeeld kan uiteraard op veel punten verbeterd worden; we lieten immers zien hoe je *zo snel mogelijk* een eenvoudig project kunt autoconfiscaten. Wat je zou kunnen doen is zowel `ChangeLog` als het versienummer automatisch genereren. Voor het genereren van `ChangeLog` kun je svn2cl (http://ch.tudelft.nl/~arthur/svn2cl/) (in Debian package subversion-tools (http://packages.debian.org/subversion-tools)) gebruiken. De versie kun je uit de datum genereren en hoef je dan niet in `configure.ac` bij te houden; die wordt dan bijvoorbeeld 20070119. Bestanden zoals `README` zou je kunnen installeren in `/usr/local/share/doc/uvt-sum`, zodat de gebruiker die eenvoudig terug kan vinden op een systeem waarop je software is geïnstalleerd. Het is netter om het bestand `bootstrap` mee te leveren in je tarball (toevoegen aan `EXTRA_DIST` dus), zodat ook mensen die geen toegang hebben tot jouw versiecontrolesysteem eenvoudig het bouwsysteem kunnen wijzigen.

In de broncode (http://mdcc.cx/pub/systraq/systraq-latest/) van systraq (http://mdcc.cx/systraq/) kun je voorbeelden vinden van al deze truukjes.

Verder kun je, wanneer je autoconf gebruikt, gebruik maken van de variabelen die configure bekend maakt. In plaats van een bestand `mijnscript` lever je een bestand `mijnscript.in`. In `mijnscript.in` kun je dan dingen als `@PACKAGE@` en `@prefix@` gebruiken. `configure` expandeert die variabelen en slaat het resultaat op in `mijnscript`. Met `$ grep ^s, config.status` kun je zien welke variabelen je allemaal tot je beschikking hebt.

## 5.5. Alternatieven voor de Autotools

Als je dit er allemaal nogal angstaanjagend uit vindt zien (wat op zich niet zo raar zou zijn), dan zijn er alternatieven voor de Autotools. We noemen er twee: SCons (http://www.scons.org/) en CMake (http://www.cmake.org/). Ga zelf na of de voordelen die deze tools opleveren, opwegen tegen de twee genoemde voordelen van de Autotools.

Naast die twee alternatieven, die vooral het bouwen van C programma's vergemakkelijken, zijn er ook systemen die specifiek voor andere talen geschreven zijn. (Helaas heeft vrijwel iedere programmeertaalgemeenschap het probleem 'hoe package ik software, geschreven in deze taal' op zijn eigen manier opgelost.)

- Voor Perl is er Module::Build (http://search.cpan.org/perldoc?Module::Build/). Waarom je niet het verouderde ExtUtils::MakeMaker (http://www.makemaker.org/) maar Module::Build (http://module-build.sourceforge.net/) moet gebruiken, kun je leren van de presentatie 'MakeMaker Is DOOMED! (http://schwern.org/~schwern/talks/MakeMaker_Is_DOOMED/slides/)'.
- Voor Python zijn er de Python Distutils (http://docs.python.org/dist/) .
- Er is Apache Ant (http://ant.apache.org/), een Java-based build tool.
- Er is de PEAR (http://pear.php.net/) PHP Extension and Application Repository.

Om van al dit soort verschillende tarball-packages een binary package (zoals RPM (http://www.rpm.org/), .deb, of BSD ports zoals pkgsrc (http://pkgsrc.org/)), te maken zijn er ook weer verschillende tools: Voor Debian packages (http://alioth.debian.org/projects/dpkg/) zijn er bijvoorbeeld het Common Debian Build System (cdbs) (http://build-common.alioth.debian.org/) en debhelper (http://kitenet.net/~joey/code/debhelper.html). Zie ook Bijlage D.

## 5.6. Welke bestanden horen waar geïnstalleerd te worden?

In bovenstaande `Makefile.am`'s zagen we de kreten `man_MANS` en `bin_SCRIPTS`: **automake** weet waar manpages en waar scripts geïnstalleerd dienen te worden (in `/usr/local/man/` en `/usr/local/bin/` in dit geval). Verder zorgt **automake** dat de permissies op die bestanden na installatie juist staan: `-rw-r--r-- root root` en in geval van een programma `-rwxr-xr-x root root`. Ook voor `/usr/local/lib/` (systeembibliotheken) en `/usr/local/share/doc` (documentatie) zijn zulke **automake** macro's.

Echter, vaak heeft programmatuur niet genoeg aan alleen programma's. Er zijn wellicht ook bestanden die door je applicatie geschreven moeten kunnen worden. (Informatie over de toestand van je applicatie bijvoorbeeld, of logbestanden.) In dat geval ligt installeren van zulke bestanden onder `/usr/local/var/` voor de hand. De systeemgebruiker die eigenaar is van het proces dat bij je applicatie hoort, zal moeten kunnen schrijven in het relevante bestand of directory. Je package zal bij een upgrade de bestanden *niet* moeten overschrijven. Zulke bestanden worden dus typisch *niet* aangemaakt of geïnstalleerd door je tarball. Zie ook wens 3: readonly install (http://non-gnu.uvt.nl/pub/uvt-unix-doc/software-wishlist/software-wishlist.html#3_readonly_install) van de LIS Unix Software verlanglijst.

Naast state- en logbestanden, zijn er vaak configuratiebestanden nodig. De volgende kwesties spelen dan een rol: Is die configuratie-data mogelijk afhankelijk van de machine waar de software op draait? Wijzigt die data normaliter vaker dan dat er nieuwe versies van de software komen? Dan wordt die data wellicht beheerd door applicatiebeheerders, en zal die niet met je package meegeleverd moeten worden. Eventueel kan je package voorbeelden in

`/usr/local/share/doc/<packagenaam>/examples/` installeren. Zie wens 4: configuratie (http://non-gnu.uvt.nl/pub/uvt-unix-doc/software-wishlist/software-wishlist.html#4_configuration) van de LIS Unix Software verlanglijst.

De bestanden die je wel met je `.tar.gz` meelevert en installeert (programma's, libraries, documentatie), zullen niet vaak zullen hoeven te wijzigen. Verder zullen deze bestanden niet schrijfbaar hoeven te zijn. Alleen bij upgrades van het package zullen die bestanden vervangen worden.

In de Filesystem Hierarchy Standard (http://www.pathname.com/fhs/) staat een stukje over wat je zoal in /usr/local (http://www.pathname.com/fhs/pub/fhs-2.3.html#USRLOCALLOCALHIERARCHY) kunt aantreffen. Het is erg prettig wanneer je kunt voorspellen waar configuratie voor je eigen software staat, en waar toestands-data staat: je kunt dan sneller je werk doen.

## 5.7. xstow

Wanneer je gevraagd wordt software vanuit een tarball te installeren, dan kun je daarvoor xstow (http://xstow.sourceforge.net/) gebruiken. Typisch worden dan de commando's

```
./configure
make
make install prefix=/opt/aap-1.0
```

uitgevoerd. Daarna wordt xstow zelf aangeroepen (b.v. als `xstow -t /usr/local aap-1.0` vanuit de directory `/opt`) en zullen daardoor symlinks aangelegd worden onder `/usr/local` (b.v. zoals `/usr/local/bin/aapd`) naar een package-specifieke directory onder `/opt` (b.v. `/opt/aap-1.0/bin/aapd`). Gebruik van xstow heeft verschillende voordelen:

1. **rollback.** In geval van problemen kan op een snelle en eenvoudige manier teruggegaan worden naar een vorige versie van de software. (`make uninstall` is over het algemeen nogal beperkt.)
2. **bescherming tegen vergissingen in `Makefile`.** Installeren kan gebeuren met minimale permissies; root-permissies zijn niet nodig. De software zal beschikbaar komen onder `/usr/local` (dus de binaries staan in het standaard `PATH` `/usr/local/bin`, en ook de manpages zijn zonder verdere aanpassingen beschikbaar), maar toch kan `make install` gedraaid worden zonder schrijfrechten onder `/usr/local`. Op deze manier kan een fout in een aangeleverd `Makefile` dus niet tot onbedoeld schrijven in b.v. `/` leiden.

Er zijn ook nadelen:

1. **prefix-support.** Een nadeel is: de software moet op een eenvoudige manier in een niet-standaard locatie te installeren zijn. Traditioneel gebruikt men hier de prefix-optie van `./configure` en/of **make** voor.
2. **geen integratie met besturingssysteem.** Verder is het op deze manier beheren van software natuurlijk over het algemeen niet in lijn met de rest van het besturingssysteem. Debian packages of RPM's die via een repository worden aangeboden, zijn in die zin met minder moeite te beheren en bij de tijd te houden.

3. **geen RPM of .deb.** Ook mist dependency-tracking, ontbreken installatie-scripts (bijvoorbeeld voor het aanmaken van systeemgebruikers, en het starten van daemons) en ontbreekt beheer van configuratie-bestanden.

Het ontbreken van installatiescripts is te ondervangen door in een bestand `INSTALL` (of `INSTALL.post`) een copy-and-paste-bare beschrijving van de uit te voeren commando's op te nemen. Het ontbreken van beheer van configuratiebestanden is te ondervangen door een voorbeeld-configuratiebestand in `/usr/local/share/doc/jouwpackage/examples/foobar.cfg` te installeren; zie ook Paragraaf 5.6.

# 6. Voorbeelden uit het echte leven, tips en truuks

We laten zien hoe je het allemaal aan elkaar knoopt. We gebruiken voorbeelden die gebaseerd zijn op de implementatie zoals gebruikt door het UvT LIS Unix team.

## 6.1. Applicatiebeheerder en MS Windows

Wanneer applicatiebeheer gescheiden is van het schrijven van de software, dan is het zinvol ook de bestanden die door die twee groepen van mensen beheerd worden op gescheiden plaatsen in een versiebeheersysteem onder te brengen.

Eén manier om bijvoorbeeld het `aap.cfg` configuratiebestand van machine klecker op te slaan, naast de code van package aap, is iets als: `.../systems/klecker/usr/local/etc/aap/aap.cfg` naast `.../sources/aap/configure.ac` in je versiecontroleboom te zetten.

De boom onder `.../systems/klecker/` kan dan met scp (en eventueel met hulp van caspar (http://mdcc.cx/caspar)) geïnstalleerd worden.

Het komt nogal eens voor dat applicatiebeheerders met een MS Windows desktop werken. Er zijn dan verschillende manieren om het allemaal aan de praat te krijgen.

- Je kunt Cygwin (http://cygwin.com/), een Linux-achtige omgeving voor MS Windows, op de desktop installeren. Binnen die omgeving kun je svn, caspar en een OpenSSH-client installeren.
- Je kunt TortoiseSVN (http://tortoisesvn.net/), een versiecontrolesysteem voor MS Windows gebaseerd op Subversion, installeren. Je SVN client draait dan dus binnen MS Windows. Je kunt PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty/) gebruiken om in te loggen op een Unix-systeem. Op dat systeem kun je dan `svn update` draaien, en via caspar `make` aanroepen. (Eventueel kan `svn update` en `make` in een SSH forced command gezet worden.)
- Je kunt (b.v. met PuTTY) inloggen op een Unix systeem, en op dat systeem `svn update` draaien, daar wijzigingen op je bestanden uitvoeren (met een Unix-editor zoals **vi**, **emacs**, **vim**, **nano** of **joe**), daar `svn commit` draaien, en daar (evt. via caspar) `make` aanroepen.

Zie ook het document van Kees Leune over gebruik van SSH vanaf MS Windows (https://infix.uvt.nl/autopub/its-id/ssh/) (alleen bereikbaar vanaf UvT IPs.) Op Caspar onder Windows (http://dolus.uvt.nl/dokuwiki/doku.php?id=pub:caspar-win), een Wikipagina door Hans van den Dool e.a. (alleen bereikbaar vanaf UvT IPs) kun je lezen hoe je OpenSSH client, SVN client, GNU Make en caspar installeert via Cygwin onder MS Windows.

Wanneer je liever iets anders gebruikt als caspar: er is een overzicht van Alternatieven voor caspar (http://mdcc.cx/pub/caspar/caspar-latest/doc/caspar.html#seealso).

## 6.2. Subversion en Id-expansie

Wanneer je je bestanden via Subversion gaat beheren, en in het verleden met CVS gewerkt hebt, heb je wellicht behoefte aan automatisch ge-expandeerde `Id` in je bestanden. Dat kan trouwens sowieso wel handig zijn: op die manier kun je op het produktiesysteem makkelijk zien waar bestanden vandaan komen.

Er is echter een probleem met zulke automatische tagexpansie: wanneer een niet-ASCII bestand zoals een `.jpg`-bestand beheert, dan kan daar, zonder dat je dat weet, toevallig `$Id$` in staan. Wanneer dat geëxpandeerd wordt, gaat je plaatje kapot. Er is helaas geen eenvoudige manier om aan een bestand te zien of de tag die erin staat door jou is toegevoegd en dus geëxpandeerd zou moeten worden.

Als je echter zeker weet dat je nooit `.jpg` of andere binary-achtige bestanden in je SVN-boom zult beheren, dan kun je de volgende settings in je bestand `~/.subversion/config`

```
 [...]
[miscellany]
 [...]
enable-auto-props = yes
 [...]
[auto-props]
 [...]
* = svn:keywords=Id URL Author Date Rev
```

zetten. In *alle* bestanden die je aan je boom toevoegd zullen eventuele tags dan geëxpandeerd worden. In Bijlage C vind je een meer robuust voorbeeld, dat wel voorzichtig met `.jpg` en dat soort bestanden omgaat.

## 6.3. Voorbeeld: caspar en sudo

We laten zien hoe het beheer van configuratiebestanden door een applicatiebeheerder eruit zou kunnen zien, wanneer caspar gebruikt wordt. Stel op machine `klecker` wordt o.a. het bestand `/opt/oracle/network/admin/tnsnames.ora` door een applicatiebeheerder onderhouden. Die beheerder heeft vanaf haar desktop ssh-toegang tot klecker. Daarna dient sudo(1) gebruikt te worden om een shell als gebruiker oracle te krijgen. Gebruiker oracle op `klecker` is eigenaar van `tnsnames.ora`.

Op enige plek in een versiebeheersysteem staat dan een bestand `systems/klecker/include/install.mk` met inhoud

```
csp_UHOST = klecker

# we want to install via csp_sudo(1)
csp_PUSH = $(csp_sucp_FUNC)

# user to sudo to
csp_XARG = oracle
```

```
include caspar/mk/caspar.mk
```

Verder zal er het bestand `systems/klecker/opt/oracle/network/admin/Makefile` in het versiebeheersysteem staan, met inhoud

```
csp_DIR = /opt/oracle/network/admin
include ../../../../include/install.mk
```

Naast dit `Makefile` staat `tnsnames.ora` in het versiebeheersysteem.

Wanneer meer bestanden in de directory `klecker:/opt/oracle/network/admin` moeten worden beheerd, kunnen die aan het versiebeheersysteem worden toegevoegd.

Wanneer meer directories op `klecker` zullen moeten worden beheerd, kunnen die ook aan het versiebeheersysteem worden toegevoegd, en van een soortgelijk `Makefile` worden voorzien. Het bestand `include/install.mk` zal dan ook door dàt `Makefile` worden gebruikt.

Wanneer meer machines dan alleen `klecker` van precies dezelfde bestanden moeten worden voorzien, dan kan er voor zo'n groep van systemen een tak in de versiebeheerboom worden gemaakt, met een bestand `include/install.mk` waarin niet `csp_UHOST` gezet wordt, maar iets als:

```
csp_UHOSTS = klecker ries
```

Zie ook Paragraaf 2.2 voor een overzicht van rollen van de verschillende systemen.

# 7. Conclusie

We hebben duidelijk proberen te maken hoe je met versiebeheer en software-packaging groepen van Unix-systemen door groepen van mensen beheersbaar kunt houden. Er is een schets gegeven van een mogelijke manier om deze hulpmiddelen in te zetten, en er is op alternatieven gewezen. Uiteraard is het laatste woord nog niet gezegd over deze materie: veel andere gezichtspunten zijn minstens zo valide, en natuurlijk kan dit document niet uitputtend zijn. Bij het bepalen van een zinvolle strategie om versiebeheer en packaging binnen een omgeving in te zetten, zijn natuurlijk veel gesprekken tussen de betrokkenen nodig. Dit document kan helpen bij het voeren van zulke gesprekken.

# 8. Referenties

We geven relevante andere documentatie, waarin de hier behandelde zaken uitgebreider behandeld worden.

Er is algemene documentatie: Eric S. Raymond (http://www.catb.org/~esr/)'s The Art of Unix Programming (http://www.catb.org/~esr/writings/taoup/). Een zeer uitgebreid (en goed) boek waarin ook de hier besproken zaken aan de orde komen.

Een software wishlist (http://non-gnu.uvt.nl/pub/uvt-unix-doc/software-wishlist/software-wishlist) van Unix systeembeheerders.

Als je weinig tijd hebt, en een UvT-er bent, kun je aan de slag met Subversion in twee pagina's (https://infix.uvt.nl/autopub/its-id/svn/).

De caspar (http://mdcc.cx/caspar) homepage; en de caspar manual (http://mdcc.cx/pub/caspar/caspar-latest/doc/caspar.html).

Er is autotools documentatie: 'Autoconf, Automake and Libtool' ook wel bekend als The Autobook (http://mdcc.cx/autobook/): een tutorial voor de GNU Autotools door Gary V. Vaughan e.a. Dit boek is ook verkrijgbaar in het Debian package autobook (http://packages.debian.org/autobook). Er zijn ook handleidingen die met **automake** en **autoconf** zelf worden meegeleverd; die zijn beschikbaar online: voor **automake** (http://sources.redhat.com/automake/automake.html) en **autoconf** (http://www.gnu.org/software/autoconf/manual/) en in de Debian packages automake1.10-doc (http://packages.debian.org/automake1.10-doc) en autoconf-doc (http://packages.debian.org/autoconf-doc).

# A. Een handgeschreven `Makefile` met DESTDIR en prefix-support

Onderstaand `Makefile` ondersteunt `DESTDIR` en prefix. Dit `Makefile` kun je zelf onderhouden: het is niet door **automake** gegenereerd (maar wel op een door **automake** gegenereerd `Makefile` gebaseerd).

```
VERSION = 0.0.2
srcdir = .
INSTALL = /usr/bin/install -c
man1dir = $(mandir)/man1
INSTALL_DATA = ${INSTALL} -m 644
DISTFILES = README $(srcdir)/Makefile INSTALL NEWS TODO $(SCRIPTS) $(etc_DATA) $(man_MANS)
bindir = ${exec_prefix}/bin
mandir = ${prefix}/man
exec_prefix = ${prefix}
mkdir_p = mkdir -p --
prefix = /usr/local
distdir = $(PACKAGE)-$(VERSION)
tar = tar chof - "$$tardir"
PACKAGE = cert-tcpdump

SCRIPTS= \
 cert_tcpdump \
 cert_tcpdump-check

man_MANS = cert_tcpdump.1

all:
```

```
install: $(SCRIPTS) $(man_MANS)
        test -z "$(DESTDIR)$(bindir)" || $(mkdir_p) "$(DESTDIR)$(bindir)"
        test -z "$(DESTDIR)$(man1dir)" || $(mkdir_p) "$(DESTDIR)$(man1dir)"
        list='$(SCRIPTS)'; for p in $$list; do \
          if test -f "$$p"; then d=; else d="$(srcdir)/"; fi; \
          if test -f $$d$$p; then \
            $(INSTALL) "$$d$$p" "$(DESTDIR)$(bindir)/$$p"; \
          else :; fi; \
        done
        list='$(man_MANS)'; for p in $$list; do \
          if test -f "$$p"; then d=; else d="$(srcdir)/"; fi; \
          if test -f $$d$$p; then \
            $(INSTALL_DATA) "$$d$$p" "$(DESTDIR)$(man1dir)/$$p"; \
          else :; fi; \
        done

uninstall:
        list='$(SCRIPTS)'; for p in $$list; do \
          rm -f "$(DESTDIR)$(bindir)/$$p"; \
        done

clean:

distdir: $(DISTFILES)
        rm -rf $(distdir)
        mkdir $(distdir)
        list='$(DISTFILES)'; for file in $$list; do \
          if test -f $$file || test -d $$file; then d=.; else d=$(srcdir); fi; \
          cp -pR $$d/$$file $(distdir)$$dir; \
        done

dist: distdir
        tardir=$(distdir) && $(tar) | gzip -c >$(distdir).tar.gz

.PHONY: all dist distdir install uninstall clean
```

Dit `Makefile` kan met de volgende parameters aangeroepen worden:

```
make dist
make prefix=/tmp/ install
make DESTDIR=/tmp/ install
```

# B. prefix-supporttester

Hier een script om prefix-support van een package-in-wording mee te testen:

```
#!/bin/sh -x

rm -rf /tmp/goed /tmp/fout
mkdir /tmp/goed /tmp/fout
if [ -e bootstrap ]; then
  sh bootstrap;
fi
if [ -e configure ]; then
  ./configure --prefix=/tmp/fout
else if [ -e Makefile.PL ]; then
  perl Makefile.pl PREFIX=/tmp/fout
  fi
fi
make install prefix=/tmp/goed
find /tmp/goed
find /tmp/fout
```

Met dank aan Anton Sluijtman en Hans van den Dool.

# C. Configuratiebestand voor Subversion

Hieronder zie je wat je in `~/.subversion/config` kunt zetten zodat je op een veilige manier Id-tagexpansie krijgt. Helaas zul je daarvoor alle veilige bestandsnamen expliciet moeten noemen. Zie ook http://wiki.apache.org/cocoon/SVNConfig.

```
[auto-props]
Makefile* = svn:eol-style=native;keywords=Id URL Author Date Rev
*.ac = svn:keywords=Id URL Author Date Rev
*.in = svn:keywords=Id URL Author Date Rev
# Scripts
*.sh  = svn:eol-style=native;svn:executable;svn:keywords=Id URL Author Date Rev
*.bat = svn:eol-style=CRLF;svn:executable;svn:keywords=Id URL Author Date Rev
# Text files, Source code
*.java = svn:eol-style=native;svn:keywords=Id URL Author Date Rev
*.txt  = svn:eol-style=native;svn:keywords=Id URL Author Date Rev
*.sql  = svn:eol-style=native;svn:keywords=Id URL Author Date Rev
# Web, XML
*.html = svn:eol-style=native;svn:keywords=Id URL Author Date Rev
*.js   = svn:eol-style=native;svn:keywords=Id URL Author Date Rev
*.css  = svn:eol-style=native;svn:keywords=Id URL Author Date Rev
*.x*   = svn:eol-style=native;svn:keywords=Id URL Author Date Rev
# DTD, Schemas, etc
```

```
*.dtd = svn:eol-style=native;svn:keywords=Id URL Author Date Rev
*.ent = svn:eol-style=native;svn:keywords=Id URL Author Date Rev
```

# D. Hoe maak je zelf een Debian package?

Als je dat *echt* wilt, kun je zelf een Debian package maken van je software. Dit is echter een klus waar je niet te licht over moet denken. Verder is het zo dat je, als je eindelijk een `.deb` hebt gemaakt, nog lang niet `apt-get install mijnpackage` kunt doen. Daarvoor zul je immers nog een apt-able repository moeten bouwen.

Je leest dit nog, dus je bent niet ontmoedigd. Goed zo! De beste tutorial over Debian packages kun je vinden in de Debian New Maintainers' Guide (http://www.debian.org/doc/manuals/maint-guide/). (Meer informatie over die tutorial is te vinden op de Debian Documentation Project (http://www.debian.org/doc/devel-manuals#maint-guide) pagina.) Een uitstekend voorbeeld van een goed gepackaged stuk software is het hello (http://packages.debian.org/hello-debhelper) Debian package.

In het kort komt het maken van een Debian package van een bestaand package neer op het schrijven van de bestanden `debian/rules`, `debian/control`, `debian/compat`, `debian/changelog` en `debian/copyright`. Als je CDBS wilt gebruiken (dat is *niet* aan te raden voor beginners), dan bestaat je bestand `debian/rules` (dat is het bestand waar het echte werk gebeurt) in het eenvoudigste geval uit twee regels:

```
include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/autotools.mk
```

De packages fair (http://non-gnu.uvt.nl/fair/) en systraq (http://packages.debian.org/systraq) zijn voorbeelden van zulke CDBS-packages.