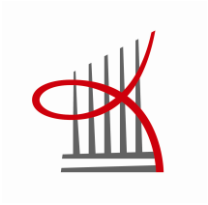


DEBIAN-PAKETOINTI OHJELMISTOJULKAISUJEN APUNA

Jarno Lukinmaa

Opinnäytetyö
Toukokuu 2012
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

JARNO LUKINMAA:
Debian-paketointi ohjelmistojulkaisujen apuna

Opinnäytetyö 36 sivua, josta liitteitä 6 sivua
Toukokuu 2012

Ohjelmistojulkaisut ovat yrityksille usein hyvin haastavia tilanteita, sillä ohjelman käyttäjillä saattaa olla hyvinkin erilaiset lähtötilanteet: ohjelmistoa ei ole koskaan ennen asennettu, ohjelmisto tarvitsee kolmansien osapuolien kirjastoja toimiakseen tai ohjelmistosta on jo asennettuna vanha versio.

Tämän opinnäytetyön tarkoituksena oli tutustuttaa työn tekijä siihen, miten Debian-paketointia voidaan käyttää hyväksi ohjelmistojulkaisuita suunnitellessa ja toteuttaessa sekä toimia samalla myös ohjeena lukijoille.

Opinnäytetyön lukemalla voi kuka tahansa helposti paketoida ohjelmansa ja julkaista sen käyttäjille.

Asiasanat: debian-paketointi, ohjelmistojulkaisu, linux, paketinhallinta, metapaketit

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree programme in Information Technology
Option of Software Engineering

JARNO LUKINMAA:
Using Debian packaging in software releases

Bachelor's thesis 36 pages, appendices 6 pages
May 2012

One of the most difficult situations for software companies is the release of new software. This is because the end users' circumstances vary often: some do not have any prior installation of the software, the software might need a 3rd party libraries to work or some users have an older version of the software installed.

The purpose of this thesis was to get to know how Debian packaging can be used to make software releases easier and also double as an instruction for readers.

By reading this thesis, anyone can easily package their software and release it for users around the world.

Key words: debian packaging, software release, linux, package management, meta-packages

SISÄLLYS

LYHENTEET JA TERMIT

Debian	linux-jakelupaketti
dpkg	Debian Package Manager. Debianin käyttämä paketinhallintajärjestelmä
Debian-paketti	dpkg:n käyttämä ohjelmistopaketti
APT	Advanced Packaging Tool. Ohjelmistopakettien asennukseen ja hallintaan käytetty apuohjelma
GNU	Gnu’s Not Unix. Projekti, jonka tavoitteena on kehittää täysin vapaa käyttöjärjestelmä

1 JOHDANTO

Yritysten julkaistessa ohjelmistoja ja/tai uusia versioita jo olemassa olevasta ohjelmistosta on tilanne usein hyvin hankala: käyttäjillä on mahdolliseti vanhoja versioita asennettuna, ohjelmassa on riippuvuuksia kolmannen osapuolen ohjelmiston tiettyyn versioon tai ohjelmiston tarvitsee poistaa edellisen version luomia tiedostoja.

Debian-paketit pystyvät vastaamaan näihin ongelmiin ja tämän opinnäytetyön tarkoitus onkin tutustuttaa sekä tekijä, että lukija, dpkg-paketinhallintajärjestelmän ja Debian-pakettien tarjoamiin ratkaisuihin.

Debianissa ja siitä johdetuissa Linux-jakelupaketeissa (mm. Ubuntu, Knoppix ja Linspire) käytetään dpkg-paketinhallintajärjestelmää, joka pohjautuu .deb (Debian) -paketteihin. Näiden pakettien sekä APT-työkalun avulla käyttäjän on helppo ja nopea päivittää käyttämänsä ohjelmat.

Työ on jaettu kolmeen loogiseen osaan. Ensimmäisessä osiossa käydään läpi yleistä tietoa Debianista ja Debian-paketeista sekä käydään läpi minkälainen ympäristö ja työkalut tarvitaan, jotta päästään alkuun. Toisessa osiossa käydään läpi tarkemmin Debian-paketin sisältö, miten sitä voidaan muuttaa sekä mitä muutokset aiheuttavat. Viimeisessä osiossa käydään esimerkein läpi muutama yleisin käyttötapaus ja pakettien asentamista.

Työn pääpaino on tutustua Debian-paketteihin: mitä ne ovat, miten niitä voidaan luoda sekä muuttaa omiin tarkoituksiin sopiviksi ja miten niitä voidaan käyttää ohjelmistojulkaisujen apuna.

2 TEKNOLOGIAT

Tässä osiossa käydään lävitse tässä työssä käytetyt pääasialliset teknologiat. Siinä tutustutaan mm. Debianiin kehitysalustana, sen käyttämään paketinhallintajärjestelmään sekä alkuun pääsemiseen tarvittaviin työkaluihin.

2.1 Debian

Debian GNU/Linux on Linux-ytimeen perustuva Linux-jakelupaketti. GNU-nimitys tulee siitä, että suuri osa Debianin tarjoamista perustyökaluista sekä ohjelmista on peräisin GNU-hankkeesta. GNU-hankkeen tavoin Debian koostuu pääosin vapaasti lisensoiduista ohjelmista, joskin mukana on myös muilla tavoilla lisensoituja ohjelmia. (Wikipedia: Debian. Viitattu 2.5.2012)

Debian tunnetaankin mahdollisuuksien paljoudesta. Tämänhetkinen vakaa julkaisu sisältää yli 29 000 ohjelmistopakettia yhdelletoista erilaiselle tietokonearkkitehtuurille, jotka käyttävät Linux kerneliä. Debianin perusasennus käyttää GNOME-työpöytäsovellusta ja sisältää ohjelmia kuten LibreOffice, Iceweasel (uudelleenbrandattu Firefox), Evolution sähköposti, CD/DVD-poltto-ohjelmat, musiikki- ja videosoittimet, kuvankäsittelyohjelman sekä PDF-lukijan. (Wikipedia: Debian. Viitattu 2.5.2012)

2.1.1 Debianin historia

Debianin perustaja on Ian Murdoc. Vuonna 1993 hän halusi luoda Linux-distribuution, joka olisi yhtä avoin kuin Linux- ja GNU-hankkeet olivat. Nimensä ”Debian” hanke sai Murdocin ja hänen silloisen tyttöystävänsä Debran etunimistä. (Wikipedia: Debian. Viitattu 2.5.2012)

Nykyisestä suosiostaan huomaamatta Debian-hanke ei tahtonut aluksi ottaa tuulta alleen. Sen ensimmäiset 0.9-alkuiset versiot ilmestyivätkin vasta vuosina 1994 ja 1995 Free Software Foundationin GNU Projektin sponsoroimana. Ensimmäinen porttaus ei-

i386-suoritinarkkitehtuurille tehtiin vuonna 1995 ja ensimmäinen 1.x versio Debianista julkaistiin vuonna 1996. (Wikipedia: Debian. Viitattu 2.5.2012)

Vuonna 1996 Debian-projektin johtoon nousi Bruce Perens, joka mm. loi ns. yhteiskuntasopimuksen Debianille varmistaen näin sen pysymisen täysin vapaana järjestelmänä ja ”rikkoi” Murdockin luoman sekä yksin ylläpitämän Debianin perustan (nk. ”ydin paketit”) jakaen sen monien ylläpitäjien kesken. (Wikipedia: Debian. Viitattu 2.5.2012)

Perens jätti projektin vuonna 1998, jonka jälkeen hankkeen merkittävämmät muutokset olivat ATP-työkalun käyttöönotto sekä uudenlainen arkistointi- ja julkaisunhallintajärjestelmä, joka mahdollisti ohjelmistojen jatkuvan testauksen seuraavaa Debian-julkaisua silmälläpitäen. (Wikipedia: Debian. Viitattu 2.5.2012)

2.2 Paketinhallinta

Debian on yksi aikaisimmista Linux-jakeluista, joka rakentuu paketeista. Tämä onkin yksi Debianin vahvuuksista, sillä APT-työkalu, laajat pakettisäilöt ja pakettien tarkastuksessa käytettävät tiukat laatuvaatimukset takaavat korkealaatuiset julkaisut, helpon päivitettävyyden julkaisujen välillä sekä automaattisen pakettien asentamisen ja poistamisen.

Seuraavissa alakappaleissa käydää läpi paketinhallinan toiminta.

2.2.1 Riippuvuudet

Asennettava ohjelma saattaa tarvita toimiakseen esimerkiksi ohjelmakirjastoja tai apuohjelmia. Näitä kutsutaan ohjelman sisältävän paketin riippuvuuksiksi. Nämä riippuvuudet voidaan kuvata esimerkiksi tiedoston nimellä tai versionumerolla.

Riippuvuuksilla on myös erilaisia tasoja: jotkut ohjelmat eivät toimi ilman, että jokin toinen paketti on asennettuna, jolloin esimerkiksi Debian-paketeissa tällaista riippuvuutta kuvataan depends-termillä. Riippuvuus voi olla myös sellainen, että

ohjelman mukana suositellaan toista pakettia, joka voi olla vaikkapa kielipaketti, käyttöohje tai graafinen käyttöliittymä.

Ohjelmapakettia asentaessa jotkin paketinhallintaohjelmat eivät suostu asentamaan pakettia mikäli paketin riippuvuudet eivät ole kunnossa. Tässä tapauksessa ohjelma ilmoittaa virheelliset kohdat ja käyttäjä voi itse ladata tarvittavat paketit. Kehittyneemmät paketinhallintaohjelmat kuten apt osaavat asentaa kaivatut riippuvuudet ja ratkaista monimutkaisiakin riippuvuusongelmia.

Käyttäjä voi toki ns. pakottaa paketit asentumaan riippuvuuksista huolimatta, mutta tällöin paketinhallintajärjestelmä ei pysty hyödyntämään asennettuja ohjelmia eikä huomaa mahdollisia tilanteita, joissa asennettu ohjelma kilpailee toisen paketin kanssa jostain resurssista.

2.2.2 Päivitys

Paketinhallintajärjestelmä osaa hakea tiedon uusista päivityksistä Linux-jakelun päivityspalvelimelta (mikäli lähteet ovat kunnossa, kts. kohta 2.2.4) sekä tarjoaa mahdollisuuden asentaa nämä päivitykset. Päivittäminen onnistuu yleensä muutamalla hiiren klikkauksella tai komentorivikomennolla tai sen voi asettaa täysin automaattiseksi.

Myös koko Linux-jakelun voi päivittää tällä tapaa uuteen versioon, mutta on kuitenkin yleensä käytännöllisistä syistä helpompi hoitaa tällainen suurempi päivitys vanhanaikaisemmalla tavalla – uudelleenasennuksella.

2.2.3 Paketin sisältö

Asennettava paketti koostuu yleensä itse tiedostoarkistosta, asennusskripteistä sekä pakettia koskevista tiedoista kuten riippuvuudet tai tiedostojen tarkistussummat. Paketit ovat yleensä binääripaketteja, mutta joissan Linux-jakeluissa, kuten Gentoossa, ohjelmat ovat lähdekoodimuodossa. Debian-pakettien sisällöstä tarkemmin kappaleessa 3.

2.2.4 Lähteet

Tarkastaakseen päivitykset, tarvitsee paketinhallintajärjestelmälle kertoa pakettilähteet. Asennuksen yhteydessä määritellään usein oletuksena Linux-jakelun virallinen päivityspalvelin, mutta pakettilähteisiin on mahdollista lisätä myös esimerkiksi käyttäjäyhteisön ylläpitämiä lähteitä, joista löytyy hieman huonommin ylläpidettyjä paketteja. Näiden pakettien kanssa tulee olla kuitenkin varovainen, sillä huonosti tehty paketti saattaa rikkoa käyttäjän järjestelmän.

2.3 Työkalut ja ympäristö

Luodaksesi tai jakaaksesi Debian-paketteja sinun ei tarvitse olla virallinen Debian-kehittäjä. Jos kuitenkin haluat lähettää paketteja ja toimia jonkin virallisen Debian-julkaisun paketin ylläpitäjänä, tulee sinun hankkia itsellesi virallisen Debian-kehittäjän status. Tämä työ keskittyy epävirallisien pakettien tekemiseen ja jakamiseen esimerkiksi työpaikoilla tai harrasteprojekteissa, mutta mikäli olet kiinnostunut virallisista paketeista, löytyy lisätietoa Debian-projektin nettisivuilta kohdasta ”Kuinka voit auttaa?”.

Vaikka Debian-pakettien tekeminen ja päivittäminen onkin helpoimmillaan suhteellisen simppeli prosessi, on muutamat perusasiat ja -työkalut hyvä olla kunnossa. Aivan ensimmäikseksi tarvitaan luonnollisesti Debian-pohjainen käyttöjärjestelmä sekä ohjema, joka halutaan paketoida. Seuraavana muutama yleisesti käytetty työkalu:

dpkg-dev sisältää joukon kehitystyökaluja, joita tarvitaan Debian-pakettien purkamiseen, rakentamiseen sekä lähettämiseen.

autoconf on kattava paketti M4 makroja, joilla tuotetaan shelli-skriptoja konfiguroimaan ohjelmiston lähdekoodipaketteja automaattisesti. Nämä skriptit voivat sopeuttaa luotavan paketin monille Unix-pohjaisille systeemeille ilman käyttäjän manuaalista väliintuloa.

dh-make työkalupaketti mahdollistaa lähdekoodipakettien muuntamisen sellaiseen formaattiin, että siitä voidaan rakentaa Debian-paketteja.

devscripts on kokoelma skriptoja, jotka helpottavat Debian-paketin ylläpitäjän elämää.

fakeroot:n avulla voidaan ajaa käskyjä Debian-ympäristössä siten, että sillä vaikuttaisi olevan root-oikeudet tiedostojen käsittelyyn. Tämän avulla käyttäjä pystyy luomaan arkistoja kuten esimerksiksi tar, ar, ja .deb-paketit, joissa on root-tunnuksen käyttämiä tai luomia tiedostoja. Ilman tätä työkaluja käyttäjällä olisi arkistoja luodessa oltava aina root-oikeudet käytössään.

gnupg on GNU:n käyttämä työkalu suojattuun kommunikointiin sekä tiedon säilyttämiseen. Sitä voidaan käyttää salaamaan tietoa ja luomaan digitaalisia allekirjoituksia. Se tarjoaa kattavan avaimenhallintajärjestelmän ja tukee OpenPGP-standardia.

build-essential sisältää listan paketeista, joita pidetään olennaisina Debian-pakettien kääntämiseen. Tämä paketti on myös riippuvainen kyseisen listan paketeista, jotta niiden asentaminen olisi helppoa. Asennettuasi tämän paketin, tarvitsee käyttäjän asentaan ainoastaan paketin käännönaikaiset riippuvuudet kääntääkseen tietyn paketin.

lintian analysoi Debian-paketteja ja reportoi bugeista sekä käytäntörikkomuksista. Se sisältää automaattisen Debian-käytäntöjen tarkituksen sekä tarkistaa yleisimmät virheet paketista.

libtool on geneerinen apuskripta kirjastoille. Se peittää monimutkaisten erityiskirjastojen, kuten esimerkiksi jaetut kirjastot, generoimisen simppelin rajapinnan taakse.

automake on työkalu, jolla luodaan Makefile.in-tiedosto Makefile.am-tiedostoista. Työkalun idea on, että sen avulla Makefile-tiedoston ylläpito siirretään käyttäjältä automake:n harteille.

pbuilder on työkalu, joka tarjoaa käyttäjälle automaattisen Debian-pakettien rakennusjärjestelmän henkilökohtaiseen työasemakehitykseen. Se rakentaa paketit chrootin avulla luodussa puhtaasa ympäristössä, jolla varmistetaan että paketti voidaan rakentaa myös useimmilla Debian-asennuksilla.

3 DEBIAN-PAKETTI

Aikaisemmissa luvuissa tutustuimme Debian-pakettien käyttöön ja tarkoitukseen. Tässä luvussa on tarkoitus käydä läpi Debian-paketin rakenne ja tutkia, että mistä osista se koostuu. Mutta mikä oikeastaan on Debian-paketti?

Lyhyesti, Debian-paketti on kokoelma tiedostoja ja ohjeita mitä tehdä niille. Paketti sisältää yleensä ohjelman tai ohjelmia, mutta joskus siinä on vain ohjeita, käyttöliittymäteemoja tai muita tiedostoja joita on helpompi levittää asennettavana pakettina.

Paketti sisältää ohjeet missä sen sisältämät tiedostot tulisi sijaita tiedostojärjestelmässä, mitä kirjastoja tai muita ohjelmia paketti tarvitsee asentuakseen; asennusohjeita ja konfiguraatio skriptejä. Monia paketteja ei suositella käytettäväksi, tai ei edes voi käyttää paketin mukana tulevan konfiguraatiotiedoston oletusasetuksilla. Esimerkiksi Apachen tapauksesa asennus täytyy konfiguroida paketin asennuksen jälkeen, jotta se toimisi.

Paketit sisältävät yleensä esi-käännetyn ohjelmiston, mutta myös lähdekoodi voidaan paketoida. Jotkin käyttäjät saattavat mieluummin asentaa ohjelmat lähdekoodista, tai jakamasi ohjelmisto saattaa tarvita kustomointia ennen sen kääntämistä. Jos siis joskus haluat jakaa ohjelmistoasi vapaan lähdekoodin lisenssillä, saatat haluta luoda myös lähdekoodipaketin normaalin ns. binääri-paketin rinnalle.

3.1 Sisältö

Debian-paketteja luodessa sen sisältö jakaantuu kahdenlaisiin tiedostoihin: pakollisiin tiedostoihin, joita ilman pakettia ei voida luoda ja valinnaisiin tiedostoihin, joita saatetaan tarvita mikäli halutaan paketin tekevän jotain perustoiminnallisuudesta poikkeavaa kuten esimerkiksi suorittaa ajoitettuja tehtäviä säännöllisesti.

3.1.1 Pakolliset tiedostot

control-tiedosto sisältää lukuisia arvoja, joita dpkg, apt-get, aptitude ja muut paketinhallintatyökalut käyttävät pakettien hallintaan. Alla esimerkkinä Debianin versio GNU-projektin ”hei maailma”-ohjelman control-tiedostosta.

```
1  Package: hello
2  Priority: optional
3  Section: devel
4  Installed-Size: 45
5  Maintainer: Firstname Lastname <firstname.lastname@example.com>
6  Architecture: i386
7  Version: 1.3-16
8  Depends: libc6 (>= 2.1)
9  Description: The classic greeting, and a good example
10 The GNU hello program produces a familiar, friendly greeting.  It
11 allows nonprogrammers to use a classic computer science tool which
12 would otherwise be unavailable to them.
13 .
14 Seriously, though: this is an example of how to do a Debian package.
15 It is the Debian version of the GNU Project's `hello world' program
16 (which is itself an example for the GNU Project).
```

KUVA 1. Control-tiedoston esimerkki.

Package-kenttä kertoo paketin nimen. Tätä nimeä hyödyntäen paketinhallintatyökalut voivat manipuloida pakettia ja tämä nimi on myös yleensä, mutta ei aina, sama kuin lähdekoodipaketin nimi.

Priority-kenttä indikoi kuinka tärkeää tämä paketti on asentaa. Tätä tarvitaan, jotta ohjelmat kuten dselect tai aptitude osaavat lajitella paketin oikeaan kategoriaan. Esimerkin tapauksessa paketti kuuluisi optional-kategoriaan, joten sen asentaminen ei ole käyttäjälle pakollista, aivan kuten esimerksiksi Windows-käyttöjärjestelmissä automaattipäivitysominaisuus tarjoavaa mahdollisuutta asentaa vaikkapa kielipaketteja tai ohjekirjoja.

Section-kenttä määrittelee sen, mistä osiosta kyseinen paketti löytyy Debianin FTP-palvelimelta.

Installed-size -kenttä kertoo kuinka paljon levylitaa paketti tarvitsee. Asennusohjelmat käyttävät tätä tietoa päätelläkseen, että onko ohjelman asentamiseen käytössä tarpeeksi levytilaa vai ei.

Mainteiner-kenttä kertoo paketin senhetkisen ylläpitäjän nimen ja sähköpostiosoitteen, jotta mahdollisissa ongelmatapauksissa ylläpitäjään voidaan olla yhteydessä.

Architecture-kenttä määrittää millä arkkitehtuurilla paketti toimii.

Version-kenttä kertoo sekä upstream-kehittäjän version numeron sekä tämän Debian-paketin revision.

Depends-kenttä listaa paketit, jotka täytyvät olla asennettuna ennen kuin tätä pakettia voidaan asentaa onnistuneesti. Jos ohjelmasta on oltava tietty versio, haluttu versio voidaan ilmoittaa sulkeissa (esimerkissä tarvitaan libc6-paketin versio 2.1 tai uudempi). Tässä kohtaa on mahdollista olla myös muita ehtoja kuten Recommends, Suggests tai Provides. Näistä lisää kappaleessa 4.

Description-kenttään voi kirjoittaa lyhyen kuvauksen paketin ominaisuuksista.

copyright-tiedosto sisältää tiedot paketin tekijänoikeuksista ja käytetyistä lisensseistä. dh_make osaa tehdä tästä pohjan, johon käyttäjä voi helposti täyttää puuttuvat tiedot kuten esimerkiksi mistä paketti on saatu, tekijänoikeustiedot sekä lisenssit.

changelog-tiedostoon listataan pakettiin ajansaatossa tehdyt muutokset. Sen tulee olla sellaisessa formaatissa, josta dpkg sekä muut paketinhallintaohjelmat osaavat lukea paketin versionumeron, reviision, jakelun sekä tärkeyden.

Tämä on yksi tärkemmistä tiedostoista paketin luojalle, sillä kaikki muutokset on hyvä olla kirjattuna ylös, jotta mahdolliset käyttäjät näkevät onko paketissa sellaisia ongelmia joista heidän olisi syytä olla tietoisia. Alla ”hei maailma” -esimerkin changelog.

```
1  hello (1.3-16) unstable; urgency=low
2
3   * Initial release.
4
5  -- Firstname Lastname <firstname.lastname@example.com>  Sun, 5 May 2012 19:306:31 +0200
6
```

KUVA 2. Changelog-tiedoston esimerkki.

rules-tiedosto on Makefilen kaltainen tiedosto, jonka perusteella varsinainen paketti luodaan. Tämän tiedoston perusteella ohjelma käännetään ja asennetaan hakemistoon debian/paketin_nimi, esimerkin tapauksessa debian/hello. Tähän hakemistoon asentuvien tiedostojen perusteella luodaan varsinainen paketti.

Jos käytössä on autoconf-työkalu, osaa paketin luova työkalu tehdä tarvittavat aestukset itse, eikä käyttäjän tarvitse periaatteessa edes koskea rules-tiedostoon. Vähänkään monimutkaisempien pakettien kohdalla sitä on kuitenkin yleensä muokattava. Liitteenä dh_make:n luoma malli rules-tiedostosta.

3.1.2 Valinnaiset tiedostot

dirs-tiedostossa luetellaan hakemistot, joiden pitää olla olemassa kun paketissa olevaa ohjelmaa asennataan, mutta joita ohjelma ei normaalin asennusprosessin aikana luo. Esimerkiksi /usr/bin on usein tällainen hakemisto.

.ex-päätteiset tiedostot dh_make luo nämä tiedostot automaattisesti. Näitä muokkaamalla voi käyttäjä esimerkiksi lisätä tehtäviä cronille tai kuvakkeen työpöytäympäristön valikkoon. Ex-nimitys tulee englannin kielen sanasta example, esimerkki, ja näitä tiedostoja ei huomioda pakettia tehdessä.

Ottaakseen jonkin näistä tiedostoista käyttöön, on käyttäjän vaihdettava tiedoston nimi ja poistettava .ex-pääte sen perästä. Lisäksi on syytä tarkistaa, että rules-tiedosto on lisättyjen tiedostojen kanssa ajantasalla. Ns. turhat .ex-tiedostot on syytä poistaa.

manpage.1-tiedosto on ohjelman man-sivu. Ottaakseen tämän käyttöön on käyttjän selvitettävä ensin mihin kategoriaan tehty man-sivu asennetaan ja muutettava pisteen jälkeen oleva numero vastaamaan tätä kategoriaa. Lisäksi tulee muuttaa rules-tiedostoa siten, että man-sivu myös asennetaan paketin mukana.

Tämä tiedosto on yleensä debian-hakemistossa muotoa paketinnimi.numero.

manpage.sgml tiedosto on SGML-muodossa kirjoitettu man-sivu. Kuten edeltävässä kohdassa, myös tämän tiedoston kohdalla on rules-tiedostoa muutettava, mutta tämän lisäksi on käännöksenaikaiseksi riippuvuudeksi lisättävä paketti docbook-to-man.

cron.d-tiedostoon lisätään rivit, jotka halutaan lisätä käyttöjärjestelmä cron-järjestelmään. Tätä tarvitaan jos halutaan esimerkiksi ladata päivityksiä tietyn aikavälein tai putsata järjestelmästä joitain tiedostoja.

init.d-tiedosto on ohjelman init-skripti, joka asennetaan kohdelaitteelle polkuun /etc/init.d/. Tämä skripti pitää huolen siitä, että asentamasi ohjelma käynnistyy aina muun järjestelmän käynnistymisen yhteydessä, eli esimerkiksi silloin, kun käyttäjä käynnistää tietokoneensa uudelleen.

3.2 Metapaketit

Julkaistavat ohjelmistot ovat usein jakautuneet moniin, erillisiin osiin ja tähän ongelmaan tarjoaa ratkaisun Debianin metapaketit.

Metapaketit ovat muuten tavallisia Debian-paketteja, mutta ne eivät sisällä lainkaan asennettavia tiedostoja. Tällaiseen pakettiin voidaan koota koko ohjelmistokokonaisuus riippuvuksia käyttäen. Riippuvuuksien ja muiden Debian-pakettien tarjoamien palveluiden kuten conflicts tai replaces avulla voidaan paketeista tehdä juuri sellaisia, kun tarvitaan.

Esimerkkinä voimme ajatella vaikkapa Microsoft Office -pakettia. Se koostuu useista ohjelmista, kuten Word, Excel, Power Point ja niin edelleen. Metapakettina meillä olisi tässä tapauksessa Office, jonka riippuvuuksia olisivat eri versiot kyseisen ohjelmistopaketin tarjoamista ohjelmista. Esimerkkiä pidemmälle vietynä, meillä voisi olla jopa paketti Windows 7, jonka riippuvuuksissa olisi kaikki Windows-käytttöjärjestelmän tarvitsemat tai tarjoamat ohjelmistot.

4 PAKETIN LUOMINEN

Kun ympäristö, työkalut sekä paketinhallinta&paketit ovat hallussa, voidaan siirtyä itse käytäntöön. Tässä osiossa käymme lävitse esimerkin avulla, kuinka Debian-paketti luodaan ns. tyhjästä. Käytämme esimerkkiohjelmana GNU Hello -ohjelmaa, joka on GNU-projektin esimerkkiohjelma.

4.1 Asennettava ohjelma

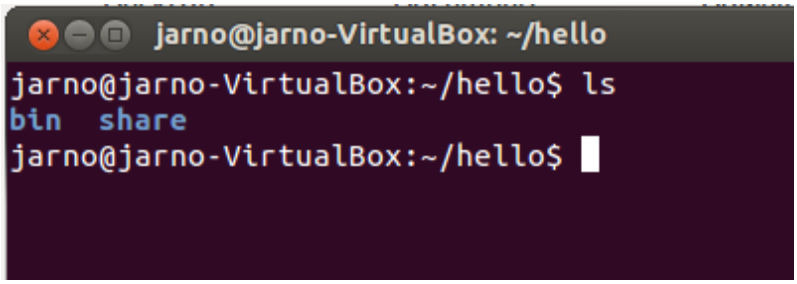
Paketin luominen aloitetaan paketoitavan ohjelman tekemisestä/hankkimisesta. Tällä kertaa käytämmä pohjana GNU-projektin Hello-ohjelman versiota 2.8, jonka voi ladata osoitteesta http://www.gnu.org/software/hello/. Ladattuamme paketin puramme sen, jolloin syntyy hakemisto hello-2.8.

Latauksen jälkeen testaamme, että ohjelma kääntyy ja sen saa ajattua. Jatkoa ajatellen ajamme seuraavan komennon luomassamme hello-2.8 -hakemistossa:

./configure –prefix=/home/jarno/hello

Tämän jälkeen käännämme ja asennamme ohjelman seuraavilla komennoilla:

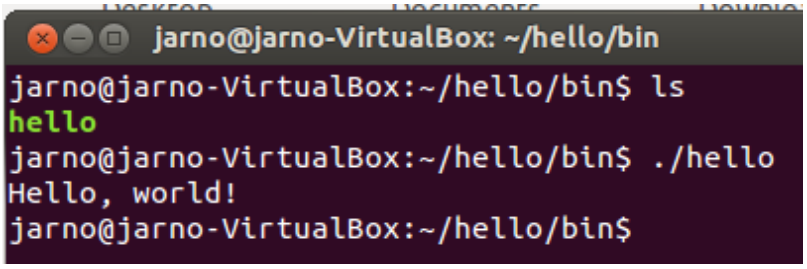
make
make install



KUVA 3. Asennus valmis.

Asennus näyttää onnistuneen, testataan vielä ohjelman ajaminen:

./hello



KUVA 4. Ohjelman ajo onnistunut

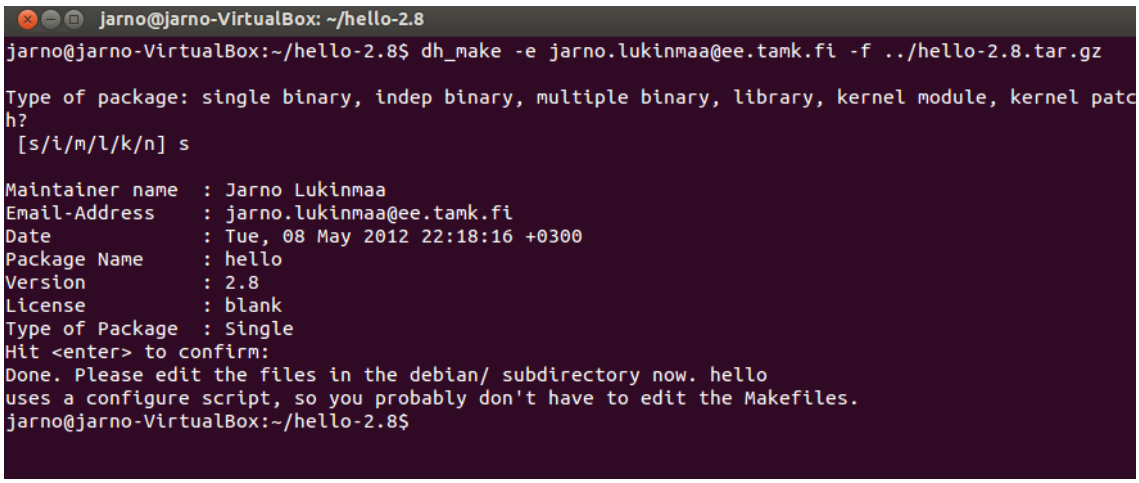
Ohjelma kääntyi oikein ja saimme sen onnistuneesti ajettua. Putsataan vielä käännöshakemisto (~/hello-2.8) seuraavalla komennolla:

make distclean

4.2 Pohja

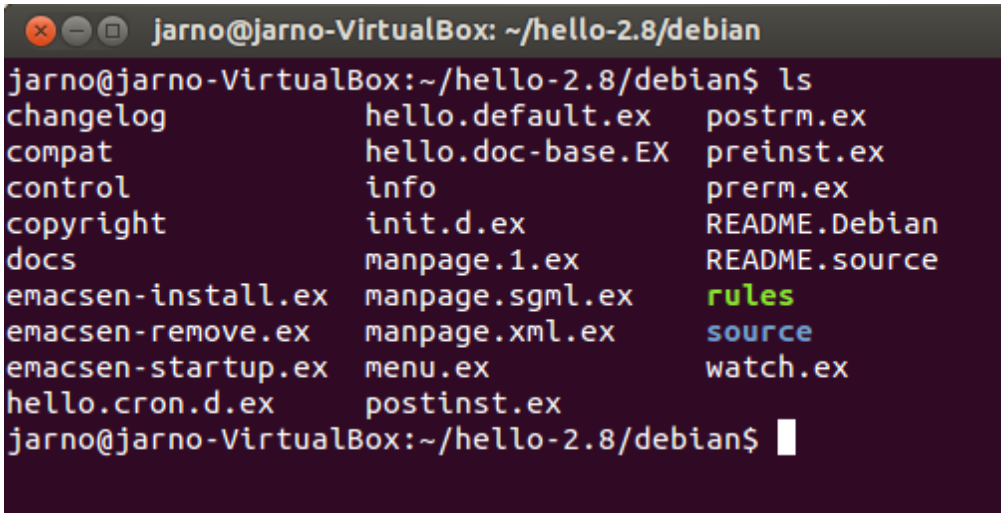
Paketin pohjan, eli tarvittavat tiedostot voi luoda manuaalisestikin, mutta tässä esimerkissä käytämme dh_make-komentoa, joka luo tarvittavat tiedostot ja hakemistot automaattisesti. Suunnistamme ensin luomaamme hakemistoon hello-2.8, jossa ajamme seuraavan komennon:

dh_make -e jarno.lukinmaa@ee.tamk.fi -f ../hello-2.8.tar.gz



KUVA 5. Paketin pohja luotu.

Ohjelma kysyy, että minkälaisen paketin haluamme luoda. Vastaamme tähän s, koska käytämme simppeliä Hello-ohjelmaa pohjana. Komennon ajettuamme on dh_make luonut ohjelmamme hakemistoon debian-hakemiston, josta kaikki paketin luomiseen tarvittavat tiedostot löytyvät.



KUVA 6. Debian-hakemiston sisältö.

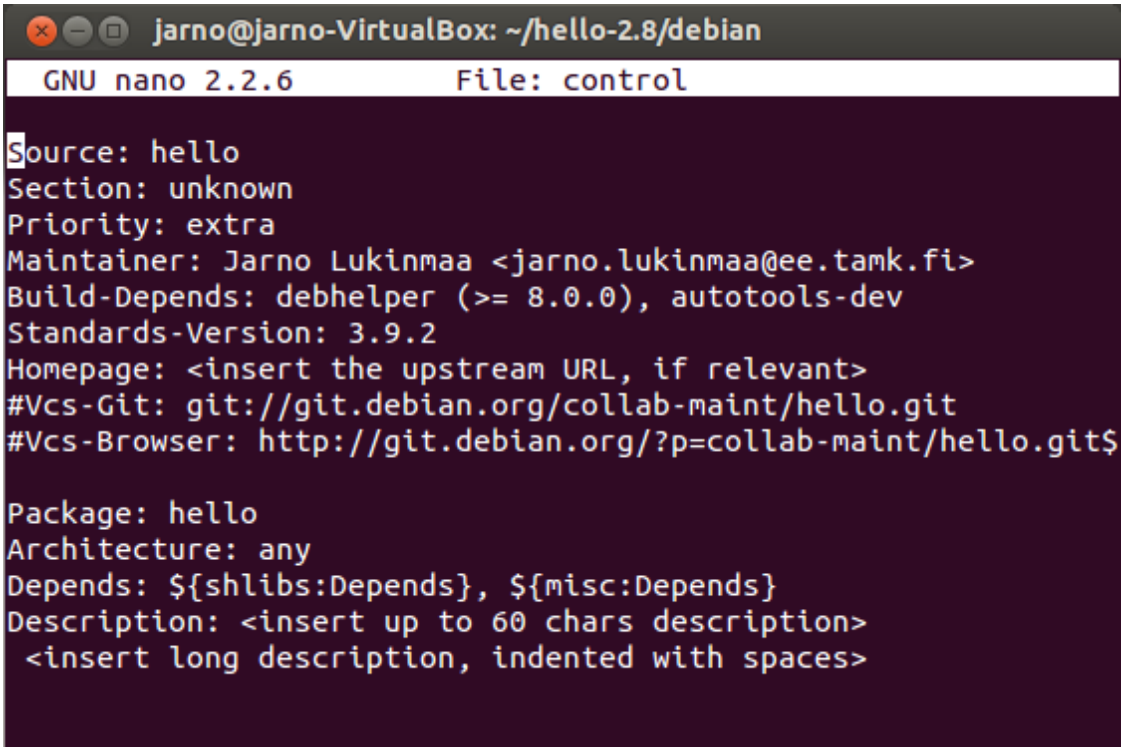
4.3 Asennushakemistot

Koska käytämme Autoconf-työkalua, ei meidän tarvitse huolehtia tästä vaiheesta juurikaan. Mikäli olisimme hoitaneet tämän kohdan manuaalisesti, olisi meidän tarvinnut luoda debian-hakemiston alle erillinen alihakemisto, jota paketin luova työkalu käyttää väliaikaisesti pakettia luodessaan. Tämän alihakemiston nimi on yleensä sama kuin paketin nimi, joten esimerkkimme tapauksessa meidän olisi tarvinnut luoda hakemisto debian/hello/.

4.4 Konfiguraatiotiedostot

4.4.1 Control

Seuraavaksi tarkastelemme dh_make:n meille luomia oletus konfiguraatiotiedostoja. Aloitamme debian/control-tiedostosta, jossa käydään läpi paketin perustiedot.

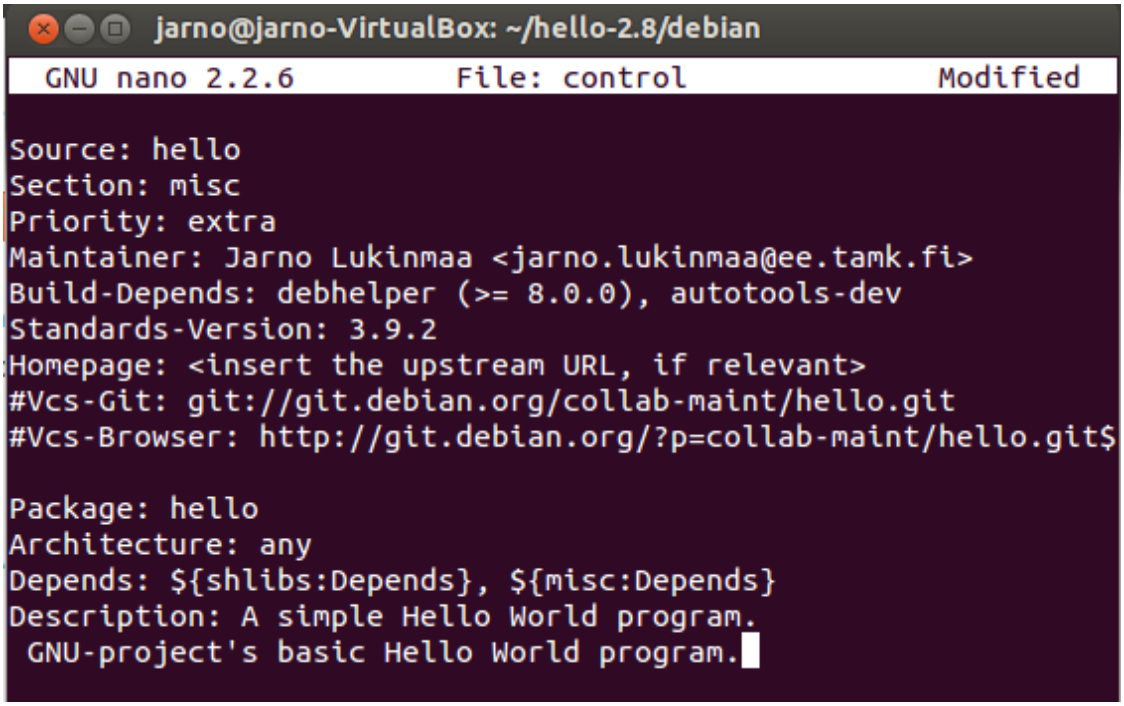


KUVA 7. Control-tiedoston sisältö ennen muokkausta.

Vaikka luomamme control-tiedoston sisältö sinällään onkin täysin kelvollinen paketin luomista ajatellen, on siihen silti hyvä tehdä joitain muutoksia, sillä oletusarvot ovat turhankin geneeriset. Ensimmäiseksi paketille tarvitsee valita jokin osio, johon se kuuluu. Osiot on listattuna Debian policyssa (http://www.debian.org/doc/debian-policy/ch-archive.html#s-subsections) ja esimerkin paketille voisi misc-osio sopia kuvauksen perusteella hyvin.

Koska pakettimme ei ole kovinkaan olellinen muun järjestelmän kannalta, niin jätetään sen tärkeysasteeksi extra. Paketin tekijä jätetään koskemattomaksi, sillä se määritettiin jo pakettia luodessa. Mikäli kuitenkin tässä kohdassa esiintyi virheitä, voi sen nyt jälkikäteen korjata. Esimerkkimme ollessa näin simppeli, ei tarvitse Build-Debends-kohtaan listata tämän enempää paketteja kuin siinä nyt on. Normaalisti tähän kohtaan listattaisiin paketit, joita tämän kyseisen paketin kääntämiseen tarvittaisiin.

Standards-Version -kohta kertoo käytettävän Debian policyn version, joten tähän meidän ei tarvitse koskea. Myöskin paketin nimen jätämme rauhaan tällä kertaa. Arkkitehtuurista huolehtelemisen jätämme paketointityökalun harteille, joten käytämme tässä kohtaa arvoa ”any”. Esimerkin vuoksi lisäämme paketille kuvaukset description-kohtaan.



KUVA 8. Muokattu control-tiedosto.

Hyppäsimme esimerkissämme Depends-kohdan ylitse, sillä Hello-paketti ei tarvitse muita paketteja toimiakseen. Aina näin ei voida kuitenkaan tehdä, joten käymme seruvaaksi läpi mitä muita vaihtoehtoja riippuvuuksien (Depends) lisäksi on olemassa.

Recommends: Tämän paketin kanssa suositeltavat paketit. Riippuvuuksista poiketen nämä paketit eivät ole pakollisia asentaa, mutta erittäin suositeltavia.

Suggests: Ehdotetut paketit. Tähän listataan sellaiset paketit, joista tämä paketti hyötyy tavalla tai toisella, mutta jotka eivät kuitenkaan ole paketin toiminnan kannalta välttämättömiä.

Enhances: Lista paketeista, joita tämä paketti parantaa tai tehostaa.

Pre-Depends: Toimii hieman samalla tavalla kuin riippuvuudetkin, mutta sillä erotuksella, että pakettia asennettaessa paketti pakottaa paketinhallintatyökalun asentamaan tähän listatut paketit.

Breaks: Lista paketeista, jotka tämä paketti tavalla tai toisella rikkoo tai joiden kanssa se ei ole yhteensopiva. Paketinhallintatyökalu ei suostu asentamaan pakettia, jollei tähän

listattuja toisia paketteja ole otettu pois käytöstä. Käytetään yleensä, kun jonkin ohjelmiston uusi versio ei ole suoraan yhtäänsopiva vanhan version kanssa.

Conflicts: Toimii vähän kuten Breaks, mutta on tätä ankarampi. Siinä missä paketti voi olla Breaks-kohtaan listatun paketin kanssa purettuna yhtä aikaa, ei Conflicts anna edes purkaa asennettavaa pakettia mikäli asennettuna on jokin Conflicts-kohdassa listatuista paketeista.

Mikäli tällainen konflikti syntyy pakettia asennettaessa, on asennettuna oleva paketti poistetava tai otettava pois käytöstä ennen kuin paketti voidaan asentaa. Mikäli asennettava paketti korvaa jo asennetun paketin tai kaksi yhtä tärkeää pakettia on konfliktissa, paketinhallintajärjestelmä poistaa yleensä konfliktin aiheuttavan paketin.

Breaks- sekä Conflicts-ehtojen käyttöä tulee välttää ellei asennettava paketti yksinkertaisesti voi olla asennettuna toisen paketin kansa samaan aikaan tai asennettava paketti rikkoo jo jonkin asennettuna olevan paketin.

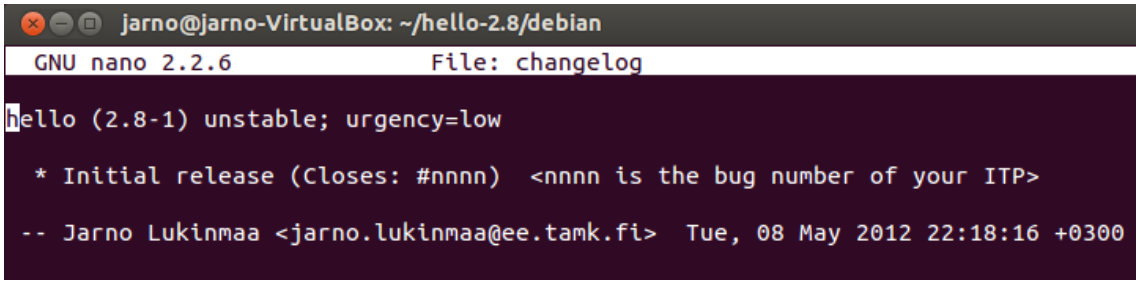
Replaces: Lista paketeista, joiden tiedostoja paketti ylikirjoittaa tai jotka se korvaa kokonaan.

4.4.2 Rules

Koska käytimme Autoconf-työkaluaa ja pakettimme on muutenkin suhteellisen simppeli, ei meidän tarvitse koskea tähän tiedostoon ollenkaan. Dh_make:n meille luoma malli löytyy liittestä 1.

4.4.3 Changelog

Pakettimme ollessa ensimmäinen julkaisu, ei meidän tarvitse koskea dh_make:n meille luomaan pohjaan. Mikäli muutoksia tarvitsisi kuitenkin tehdä, niin changelog-tiedostoa on mahdollista muokata joko suoraan tekstieditorilla kuten esimerkiksi nano tai vaihtoehtoisesti käyttää dch(debchange)-työkalua, jonka avulla voidaan luoda valmis pohja uudelle versiolle.



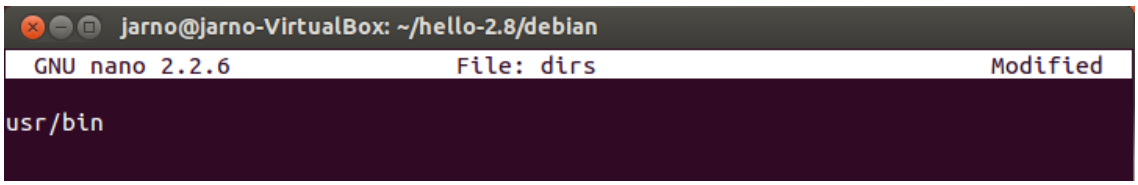
KUVA 9: Valmis changelog-tiedosto.

4.4.4 Copyright

Vaikka copyright-tiedoston muoto onkin vapaa, käytämme tässä tapauksessa dh_make:n meille luomaa pohjaa. Dh_make:n luoma pohja löytyy liitteessä 2.

4.4.5 Dirs

Tähän tiedostoon määritämme Hello-ohjelmamme käyttämät hakemistot. Ajettavat ohjelmat löytyvät Debianissa usein /usr/bin-hakemistosta, joten lisäämme pakettimme dirs-tiedostoon kyseisen hakemiston ilman ensimmäistä /-merkkiä.

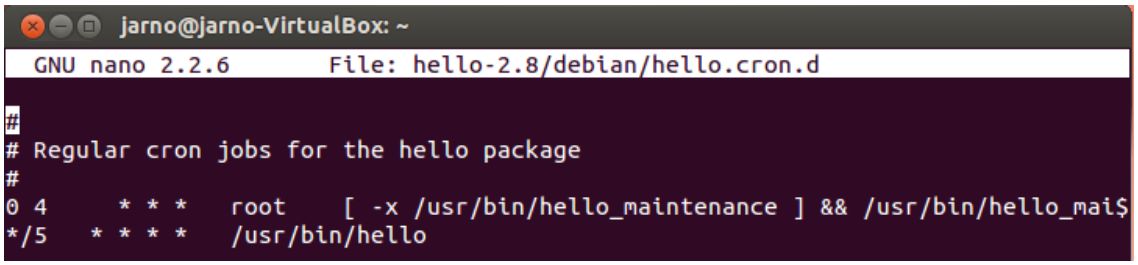


KUVA 10. Valmis dirs-tiedosto.

4.4.6 Muut tiedostot

Saadaksemme jotain hieman erikoisempaa työstämäämme pakettiin, editoimme myös muita, ei niinkään pakollisia, tiedostoja. Koska dh_make nimeää valinnaiset tiedostot .ex-päätteellä, tarvitsee kunkin käytettävän tiedoston kohdalla poistaa tämä nimestä. Uudelleen nimeäminen onnistuu joko graafisen tiedostojärjestelmän selaimen kautta tai konsolissa komennolla mv.

Aloitetaan cron-d -tiedostosta, jossa asetamme crontabin ajamaan Hello-ohjelmamme 5 minuutin välein. Poistamme aluksi .ex-päätteen tiedoston perästä ja muokkaamme siitä seuraavan näköisen:

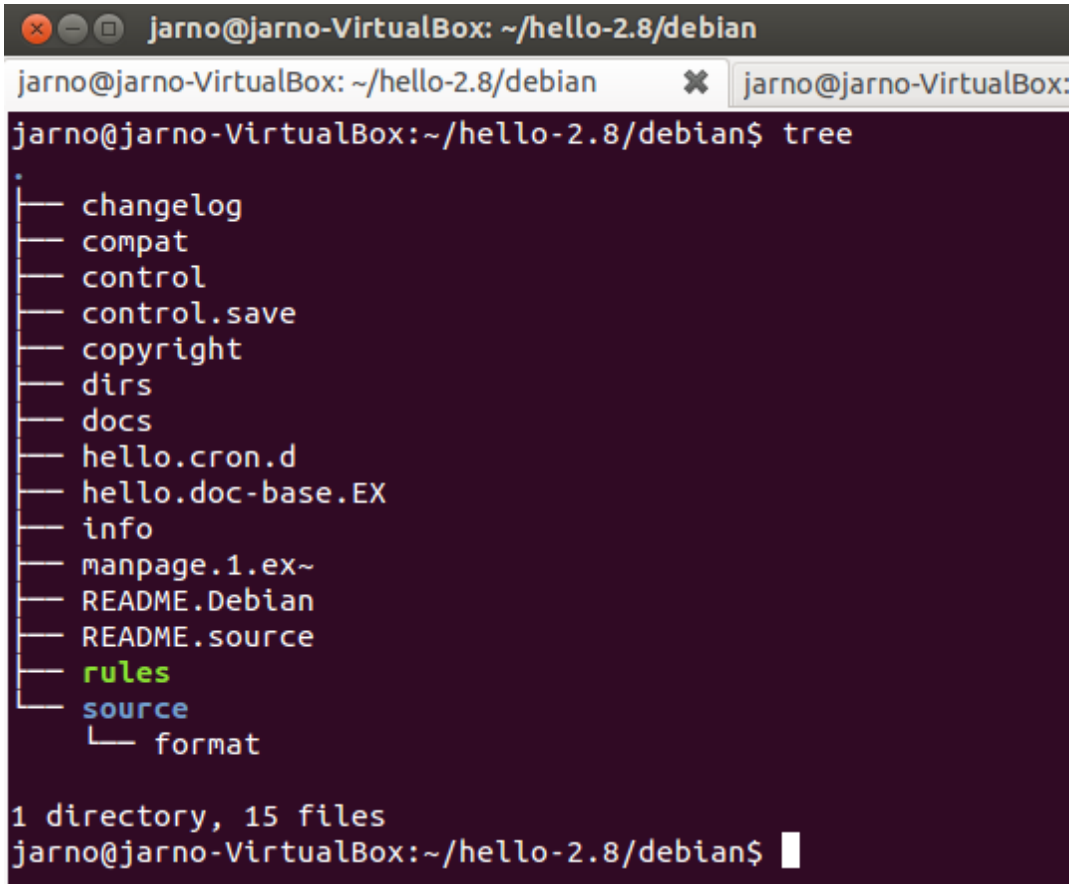


KUVA 11. Valmis cron.d-tiedosto

Ensimmäinen tehtävä on dh_make:n luoma ylläpitotehtävä joka suoritetaan joka yö kello 4 aamulla ja toinen on lisäämämme 5 minuutin välein suoritettava tehtävä, joka ajaa Hello-ohjelmamme.

4.5 Paketin kasaus

Nyt kun kaikki tarvitsemamme konfiguraatiotiedostot ovat kunnossa, on aika kasata itse paketti! Poistamme aluksi kaikki turhat .ex-päätteiset tiedostot, koska emme tarvitse niitä ja haluamme pitää paketin siistinä. Siistityn debian-hakemistomme sisältö näyttää seuraavalta:

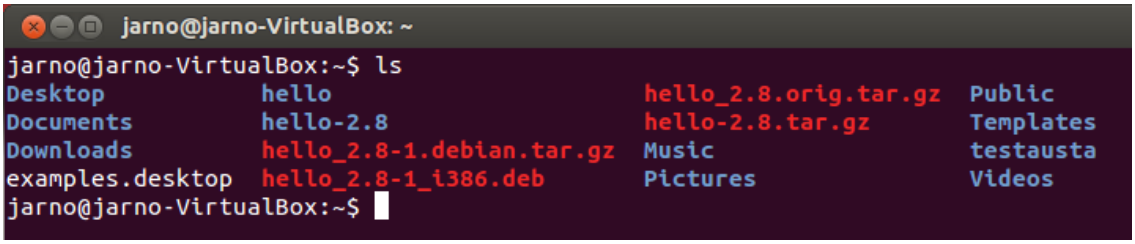


KUVA 12. Paketin sisältö ennen rakentamista.

Nyt kun paketti on siistitty kaikesta turhasta, voimme rakentaa paketin ajamalla seuraavan komennon hello-2.8 -hakemiston juuressa:

debuild

Paketti rakentuu nykyisen hakemistomme ylähakemistoon:

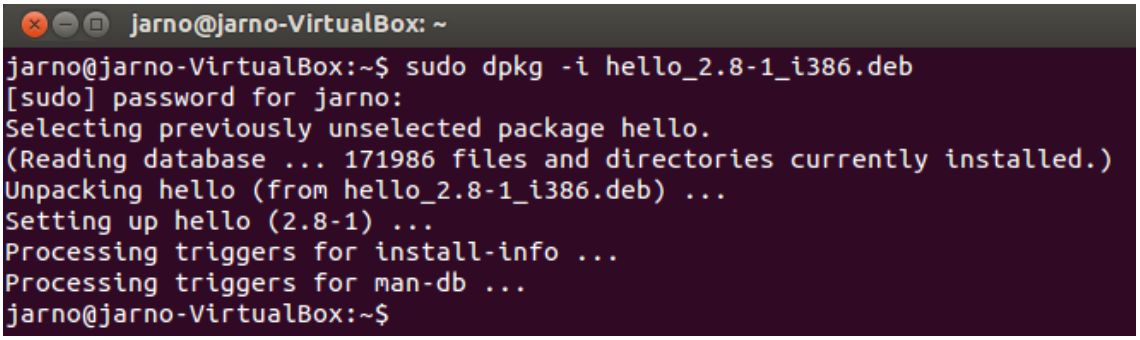


KUVA 13. Pakettimme on valmis.

5 PAKETIN ASENTAMINEN

Seuraavaksi testaamme, että luomamme paketti asentuu ja toimii muutenkin oikealla tavalla. Asennamme paketin seuraavalla komennolla:

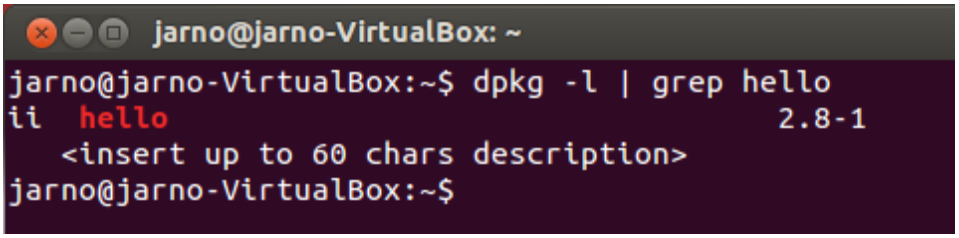
sudo dpkg -i hello_2.8-1_i386.deb



KUVA 14. Asennus näyttäisi onnistuneen.

Varmistamme vielä, että paketti on asentunut oikein seuraavalla komennolla, joka listaa asennetut paketit ja suodattaa niistä hello-nimiset paketit:

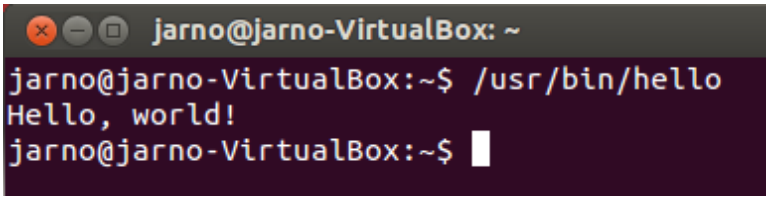
dpkg -l | grep hello



KUVA 15. Myös paketinhallinta löytää pakettimme.

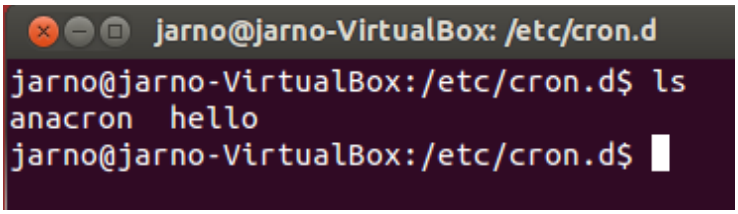
Paketti asentui onnistuneesti, joten testaamme vielä ajaa paketoimamme ohjelman komennolla:

/usr/bin/hello



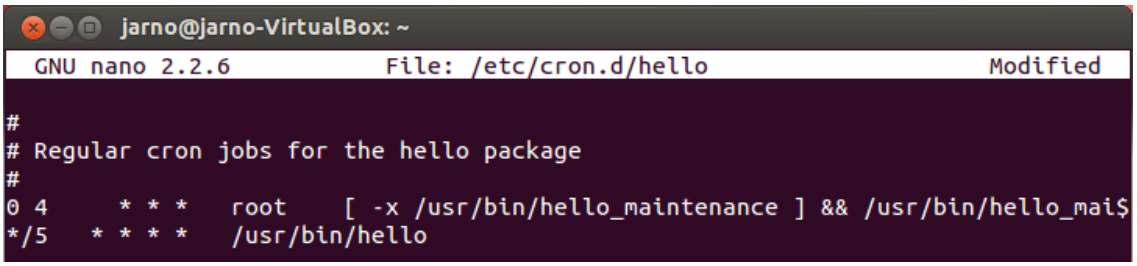
KUVA 16. Ohjelman suoritus onnistui.

Myös ohjelman ajaminen onnistui. Lopuksi tarkistamme vielä, että pakettiin lisäämämme crontab-taski asentui onnistuneesti. Tätä varten avaamme hakemiston /etc/cron.d/ josta pitäisi löytyä hello-taskimme:



KUVA 17. Cron.d-hakemiston sisältö.

Lopuksi avaamme vielä itse taskin, jotta voimme varmistua sen oikeellisuudesta:



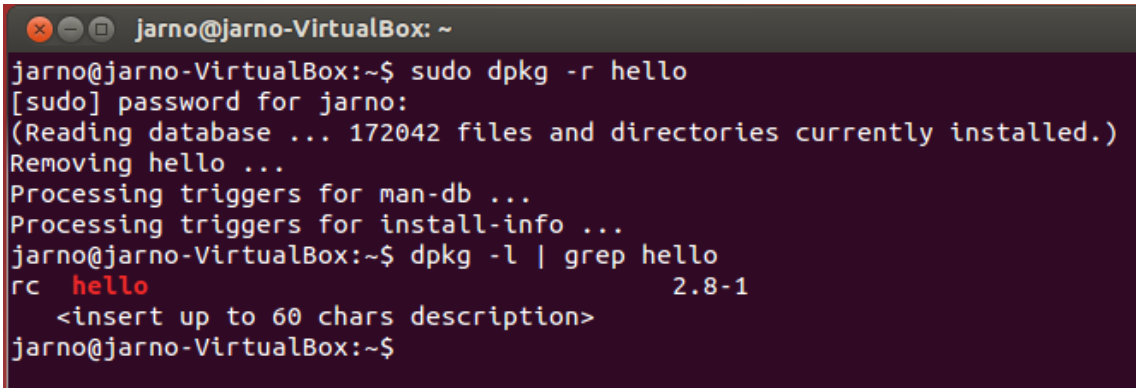
KUVA 18. hello-taskin sisältö.

Kaikki näyttäisi olevan kunnossa, joten pakettimme on nyt täysin asennettu.

6 PAKETIN POISTAMINEN

Käydään vielä lopuksi läpi miten asentamamme paketin saa poistettua. Poistaminen onnistuu seuraavalla komennolla:

sudo dpkg -r hello



KUVA 19. Paketti on poistettu.

Käyttämämme dpkg-parametri -r (remove) eli poisti pakettimme, mutta jätti paketin konfiguraatiotiedostot järjestelmäämme. Mikäli haluaisimme poistaa paketin kokonaan, voisimme käyttää --purge-parametriä dpkg:lle.

7 POHDINTA

Debian-paketit ja niiden kehitykseen käytetyt työkalut olivat työn tekijälle entuudestaan vain osittain tuttuja töiden puolesta. Perusteet olivat kunnossa, joten omaa osamista oli kohtalaisen helppo lähteä kehittämään Internetistä löytyvien oppaiden sekä ohjeiden avulla. Erityisesti Debian New Maintainer's Guide sekä Debian Policy Manual auttoivat niissä tilanteissa, kun oma tietämys ei ollut riittävää.

Työlle asetettu tavoite tutustuttaa sekä tekijä, että lukija Debian-pakettien käyttöön toteutui hyvin. Oma osaamiseni karttui valtavasti erilaistan työkalujen ja ongelmakohtien tullessa tutuiksi. Täysin alusta asti itse tehdyn Debian-paketin tekeminen oli minulle uutta ja tässä työssä esitetyn ohjeen avulla kuka tahansa pystyy helposti luomaan itselleen uusia paketteja, tai päivittämään jonkun toisen jo luomaa pakettia.

Työstä syntyi yleispätevä ohje Debian-pakettien luomiseen ja niiden käyttämiseen, mutta aivan kaikkiin kysymyksiin se ei vastaa, joten lukijan on tällaisten tilanteiden ilmetessä itse selviydyttävä ongelmakohdista. Onneksi aiheesta löytyy hyvin paljon oppimateriaalia, joten ohjelmistojen paketoiminen erikoisimmissakin tapauksissa pitäisi onnistua pienehköllä selvitystyöllä.

LÄHTEET

Wikipedia: GNU. Luettu 10.5.2012.
http://en.wikipedia.org/wiki/GNU

Wikipedia: Debian. Viitattu 2.5.2012.
http://en.wikipedia.org/wiki/Debian

Debian New Maintainer's Guide. Luettu 8.5.2012.
http://www.debian.org/doc/manuals/maint-guide/

Debian Policy Manual. Luettu 6.5.2012.
http://www.debian.org/doc/debian-policy/index.html

LIITTEET

Liite 1. Rules-tiedosto

1 (4)

```
#!/usr/bin/make -f
# -*- makefile -*-
# Sample debian/rules that uses debhelper.
# This file was originally written by Joey Hess and Craig Small.
# As a special exception, when this file is copied by dh-make into a
# dh-make output file, you may use that output file without restriction.
# This special exception was added by Craig Small in version 0.37 of dh-make.

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1


# These are used for cross-compiling and for saving the configure script
# from having to guess our platform (since we know it already)
DEB_HOST_GNU_TYPE   ?= $(shell dpkg-architecture -qDEB_HOST_GNU_TYPE)
DEB_BUILD_GNU_TYPE  ?= $(shell dpkg-architecture -
qDEB_BUILD_GNU_TYPE)


CFLAGS = -Wall -g

ifneq (,$(findstring noopt,$(DEB_BUILD_OPTIONS)))
	CFLAGS += -O0
else
	CFLAGS += -O2
endif

config.status: configure
		dh_testdir
```

```
		# Add here commands to configure the package.
		./configure --host=$(DEB_HOST_GNU_TYPE) --
build=$(DEB_BUILD_GNU_TYPE) --prefix=/usr \
 --mandir=\$${prefix}/share/man --infodir=\$${prefix}/share/info
CFLAGS="$(CFLAGS)" LDFLAGS="-Wl,-z,defs"


build: build-stamp

build-stamp:  config.status
		dh_testdir

		# Add here commands to compile the package.
		$(MAKE)
		#docbook-to-man debian/hello.sgml > hello.1

		touch $@

clean:
		dh_testdir
		dh_testroot
		rm -f build-stamp

		# Add here commands to clean up after the build process.
		-$(MAKE) distclean
ifneq "$(wildcard /usr/share/misc/config.sub)" ""
		cp -f /usr/share/misc/config.sub config.sub
endif
ifneq "$(wildcard /usr/share/misc/config.guess)" ""
		cp -f /usr/share/misc/config.guess config.guess
endif
```

3(4)

```
		dh_clean

install: build
		dh_testdir
		dh_testroot
		dh_clean -k
		dh_installdirs

		# Add here commands to install the package into debian/hello.
		$(MAKE) DESTDIR=$(CURDIR)/debian/hello install


# Build architecture-independent files here.
binary-indep: build install
# We have nothing to do by default.

# Build architecture-dependent files here.
binary-arch: build install
		dh_testdir
		dh_testroot
		dh_installchangelogs ChangeLog
		dh_installdocs
#		dh_installexamples
#	dh_install
#	dh_installmenu
#	dh_installdebconf
#	dh_installlogrotate
#	dh_installemacsen
#	dh_installpam
#	dh_installmime
#	dh_python
#	dh_installinit
		dh_installcron
```

4 (4)

```
#       dh_installinfo
                dh_installman
                dh_link
                dh_strip
                dh_compress
                dh_fixperms
#       dh_perl
#       dh_makeshlibs
                dh_installdeb
                dh_shlibdeps
                dh_gencontrol
                dh_md5sums
                dh_builddeb

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

Liite 2. Copyright-tiedosto


```
Format: http://dep.debian.net/deps/dep5
Upstream-Name: hello
Source: <url://example.com>

Files: *
Copyright: <years> <put author's name and email here>
           <years> <likewise for another author>
License: <special license>
 <Put the license of the package here indented by 1 space>
 <This follows the format of Description: lines in control file>
 .
 <Including paragraphs>

# If you want to use GPL v2 or later for the /debian/* files use
# the following clauses, or change it to suit. Delete these two lines
Files: debian/*
Copyright: 2012 Jarno Lukinmaa <jarno.lukinmaa@ee.tamk.fi>
License: GPL-2+
 This package is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.
 .
 This package 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.  See the
 GNU General Public License for more details.
 .
 You should have received a copy of the GNU General Public License
 along with this program. If not, see <http://www.gnu.org/licenses/>
 .
 On Debian systems, the complete text of the GNU General
```

2 (2)

Public License version 2 can be found in "/usr/share/common-licenses/GPL-2".

Please also look if there are files or directories which have a
different copyright/license attached and list them here.