Come creare un pacchetto Debian

Capitolo 15

<< È piuttosto comune che un amministratore, quotidianamente impegnato con diversi pacchetti Debian, decida prima o poi di creare o personalizzare un pacchetto esistente. Lo scopo di questo capitolo è quello di dare risposta alle vostre domande sull'argomento e di fornirvi gli elementi per sfruttare al meglio l'infrastruttura di Debian. Chissà forse con un pizzico di fortuna e dopo un po' di esperienza pratica attraverso i pacchetti locali, potreste desiderare spingervi oltre e partecipare al progetto Debian! >>

15.1. Ricompilazione di un pacchetto dalla sue sorgenti

La ricompilazione di un pacchetto binario è necessaria in diverse circostanze. Spesso l'amministratore necessita di una funzionalità che a sua volta può essere ottenuta: solo attraverso la compilazione (con opportuna opzione di compilazione) da una sorgente del software; oppure in una versione software più recente non inclusa nel pacchetto Debian. In quest'ultimo caso l'amministratore solitamente compila un pacchetto da una versione più recente di Debian - ad esempio `Testing` o `Unstable` - in modo che funzioni perfettamente nella distribuzione `Stable`; questa tecnica viene denominata `backporting`. Prima di implementare un'attività simile dovreste controllare se già non è stata svolta — in particolare il sistema di tracciamento dei pacchetti Debian (Debian Package Tracker) offre diverse informazioni ad hoc.

✦ https://tracker.debian.org/

15.1.1. Come recuperare le Sorgenti

Per ricompilare un pacchetto Debian dovrete prima recuperare il suo codice sorgente. Il modo più semplice è usare il comando `apt-get source source-package-name`. Questo comando richiede la presenza di una riga `deb-src` nel file `/etc/apt/sources.list` e l'aggiornamento dell'indice dei files (attraverso il comando `apt-get update`). Ciò dovrebbe essere congeniale per voi se avete seguito le istruzioni del precedente capitolo incentrato sulla configurazione di APT (in particolare il paragrafo 6.1, "Come compilare il file sources.list" a pagina 108). Occorre precisare che scaricherete i pacchetti sorgente dalla versione di Debian stabilita dalla riga `deb-src`. Se diversamente desiderate un'altra versione dovrete scaricarla manualmente da un mirror Debian o un sito Web. Dovrete recuperare due o tre files (con estensione `*.dsc` — Debian Source Control — `*.tar.comp` e talvolta `*.diff.gz` o `*.debian.tar.comp` — dove per "`comp`" si intende il formato di compressione (`gz`, `bz2` o `xz`) che può variare in base allo strumento utilizzato per la compressione; successivamente dovrete eseguire il comando `dpkg-source -x file.dsc`. Se il file `*.dsc` è direttamente disponibile tramite un determinato URL, potrete persino utilizzare il comando `dget URL` . Questo comando (incluso nel pacchetto `devscripts`) recupera il file `*.dsc` dall'indirizzo specificato, per poi analizzarne i contenuti e recuperarne automaticamente il file o i files di riferimento. Una volta scaricato il pacchetto sorgente verrà estratto localmente e ne verrà verificata l'integrità attraverso `dsverify` (a meno che non sia stata immessa l'opzione `-d` o `--download-only`). Il `keyring` di Debian è essenziale, tranne nei casi in cui si impiega l'opzione `-u`.

15.1.2. Come apportare delle modifiche

Venga considerato il pacchetto di samba.

```
$ apt source samba
Reading package lists... Done
 NOTICE: 'samba' packaging is maintained in the 'Git' version control system at:
https://salsa.debian.org/samba-team/samba.git
Please use:
git clone https://salsa.debian.org/samba-team/samba.git
```

```
to retrieve the latest (possibly unreleased) updates to the package.
Need to get 11.7 MB of source archives.
Get:1 http://security.debian.org/debian-security buster/updates/main samba 2:4.9.5+
-> dfsg-5+deb10u1 (dsc) [4,316 B]
 Get:2 http://security.debian.org/debian-security buster/updates/main samba 2:4.9.5+
-> dfsg-5+deb10u1 (tar) [11.4 MB]
Get:3 http://security.debian.org/debian-security buster/updates/main samba 2:4.9.5+
-> dfsg-5+deb10u1 (diff) [252 kB]
Fetched 11.7 MB in 1s (9,505 kB/s)
dpkg-source: info: extracting samba in samba-4.9.5+dfsg
dpkg-source: info: unpacking samba_4.9.5+dfsg.orig.tar.xz
dpkg-source: info: unpacking samba_4.9.5+dfsg-5+deb10u1.debian.tar.xz
dpkg-source: info: using patch list from debian/patches/series
dpkg-source: info: applying 07_private_lib
dpkg-source: info: applying bug_221618_precise-64bit-prototype.patch [...]
```

La sorgente del pacchetto è finalmente disponibile in una directory denominata con il nome del pacchetto sorgente e la relativa versione (per esempio `samba-4.9.5+dfsg`); dovrete apportare le modifiche all'interno della directory.

La prima modifica che dovrete apportare è al numero di versione del pacchetto in modo da poter distinguere i pacchetti ricompilati dai pacchetti originali offerti da Debian. Si supponga che la versione corrente sia `2:4.9.45+dfsg-2`: in questo caso è consigliabile modificarla in `2:4.9.5+dfsg-5falcot1`, così che l'origine del pacchetto possa essere chiara. Inoltre in questo modo la versione modificata avrà un indice numerico superiore rispetto a quella originale offerta da Debian ed il pacchetto modificato verrà facilmente installato come se fosse un aggiornamento dell'originale. Per apportare questa modifica è preferibile utilizzare il programma `dch` (`Debian CHangelog`) incluso nel pacchetto `devscripts`: ad esempio `dch --local falcot .`

```
$ cd samba-4.9.45+dfsg
$ dch --local falcot
```

Questo comando invoca un editor di testo (il `sensibile-editor` designato attraverso le variabili di ambiente `VISUAL` o `EDITOR`, altrimenti l'editor predefinito) per documentare le differenze apportate da questa ricompilazione. L'editor mostra che `dch` ha effettivamente modificato il file `debian/changelog`.

Qualora abbiate necessità di una modifica delle opzioni di compilazione, dovrete modificare il file `debian/rules`, in quanto coordina le fasi della compilazione dei pacchetti. Nei casi più semplici, individuerete facilmente le righe sia relative alla configurazione iniziale (`./configure…`), sia all'effettiva build (`$(MAKE)`… o `make…`). Se questi comandi non sono chiamati esplicitamente, sono chiamati indirettamente da altri comandi, pertanto dovrete fare riferimento alla loro documentazione per ulteriori informazioni su come modificare il loro funzionamento predefinito. Con i pacchetti che usano `dh`, potreste avere necessità di aggiungere un `override` (trad. lett. ignora o ignora e sostituisci con) per i comandi `dh_au_to_configure` o `dh_auto_build` (su questo argomento dovrete consultare le rispettive man pages).

In base alle modifiche locali ai pacchetti potrebbe essere necessario aggiornare il file `debian/control`, che contiene la descrizione dei pacchetti generati. In particolare questo file include le righe `Build-Depends` che gestiscono l'elenco delle dipendenze che devono essere soddisfatte durante la compilazione. Le `Build-Depends` si riferiscono spesso alle versioni dei pacchetti offerti dalla distribuzione da cui proviene il pacchetto sorgente, che potrebbe non essere disponibile nella distribuzione utilizzata per la ricompilazione. Non esiste un metodo automatico per determinare se una dipendenza è reale o se viene definita solo per garantire che venga tentata una compilazione con l'ultima versione di una libreria - difatti questo è l'unico metodo disponibile per forzare `autobuilder` ad usare una determinata versione di un pacchetto durante la compilazione e per raggiungere tale scopo gli sviluppatori spesso limitano le versioni delle dipendenze per la compilazione.

Se ritenete troppo rigorose le dipendenze per la compilazione dovreste sentirvi liberi di renderle meno restrittive localmente. Fate riferimento ai files che documentano il metodo standard per la compilazione del software - spesso denominati `INSTALL` - per una maggiore comprensione delle dipendenze.
Teoricamente tutte le dipendenze dovrebbero essere soddisfatte dalla distribuzione usata per la ricompilazione; qualora non fosse così si avvierà un processo ricorsivo per cui dovrà essere messo in atto il backporting dei pacchetti menzionati nel campo `Build-Depends` prima di quello del pacchetto target. Diversi pacchetti potrebbero non avere necessità del backporting e di conseguenza potrete installarli così come sono (un noto esempio è `debhelper`). Si precisa che il backporting potrebbe facilmente diventare complesso se non verrà eseguito con attenzione. Pertanto evitate i backports se non è strettamente necessario.

SUGGERIMENTO Come installare le `Build-Depends`	`apt-get` consente di installare rapidamente tutti i pacchetti menzionati nei campi `Build-Dipends` di un pacchetto sorgente disponibile da una distribuzione definita dalla riga `deb-src` nel file `/etc/apt/sources.list`. Tutto quello che dovete fare è eseguire il comando `apt-get build-dep source-package`.

15.1.3. Come avviare la Rebuild

Dopo aver apportato tutte le modifiche necessarie alle sorgenti, dovrete generare l'effettivo pacchetto binario (il file `.deb`). L'intero processo è gestito dal comando `dpkg-buildpackage` .

Esempio 15.1 Ricompilazione di un pacchetto

```
$ dpkg-buildpackage -us -uc
[…]
```

Il comando precedente potrebbe avere un esito negativo se i campi `Build-Depends` non sono stati aggiornati o se i pacchetti corrispondenti non sono stati installati. In questo caso è possibile ignorare il controllo trasmettendo l'opzione `-d` a `dpkg-buildpackage`. Ma ignorare le dipendenze potrebbe comportare un esito negativo in uno step successivo. Oppure il pacchetto potrebbe sembrare compilato correttamente e poi non essere in grado di avviarsi appropriatamente: diversi programmi disabilitano automaticamente diverse funzionalità se le librerie necessarie non sono disponibili durante la compilazione.

STRUMENTI TOOL `fakeroot`

Sostanzialmente il processo di creazione del pacchetto si limita a mettere insieme in un archivio i files esistenti (o compilati) localmente; il titolare dei diritti per la maggior parte dei files è spesso `root`. Tuttavia se effettuate una compilazione con l'identità dell'amministratore correrete dei rischi inutili; per fortuna ciò può essere evitato con l'utilità denominata `fakeroot`. Questo strumento viene utilizzato per avviare un programma in modo che ritenga di essere eseguito con l'identità di `root` e che crei i files con titolarità dei diritti e permessi arbitrari. Difatti durante la creazione dell'archivio, che diventerà il pacchetto Debian, il programma sarà indotto alla creazione di files con titolarità dei diritti arbitraria compreso `root`. Tale configurazione è così conveniente che persino `dpkg-buildpackage` per la compilazione dei pacchetti utilizza di default `fakeroot`.
Si reitera che il programma viene soltanto indotto a credere che opera con un'identità con diritti speciali quando invece si tratta dell'identità dell'utente che ha eseguito `fakeroot program` (e pertanto i permessi dei files generati sono correlati ai permessi dell'utente che li ha creati). In questo modo il programma se compromesso non può essere eseguito con i privilegi di root.

Comunque è probabile che gli sviluppatori Debian usino un programma di alto livello come ad esempio `debuild`; quest'ultimo avvia normalmente `dpkg-buildpackage` eseguendo però nel contempo un programma che verifica la conformità del pacchetto generato alla Debian Policy. Inoltre, questo script "ripulisce" l'ambiente in modo che le variabili di ambiente locali non "influiscano" sulla compilazione del pacchetto. Il comando `debuild` è uno dei tools inclusi nella suite `devscripts` di cui condivide coerenza e configurazione semplificando le attività dei manutentori.

BREVE ACCENNO
Come compilare un pacchetto in un ambiente `crooted`

Il programma `pbuilder` (incluso nel pacchetto omonimo) consente la compilazione di un pacchetto Debian in un ambiente `chroot`. Il programma crea una directory temporanea contenente un sistema con requisiti minimi per la compilazione del pacchetto (che includa i pacchetti definiti nel campo `Build-Depends`). Dopodiché la directory viene utilizzata come root directory (`/`) attraverso il comando `chroot` durante il processo di compilazione.
Questo strumento consente al processo che effettua la compilazione di operare in un ambiente esente da alterazioni dovute a personalizzazioni dell'utente. In questo modo le dipendenze di compilazione mancanti saranno rilevate rapidamente (dato che le dipendenze devono essere documentate affinché la compilazione non fallisca). Infine il tool permette di compilare un pacchetto per una versione di Debian diversa da quella utilizzata dal sistema: pertanto la macchina può utilizzare la versione Stable per il suo normale workload, mentre `pbuilder` può sfruttare la Unstable per la compilazione del pacchetto. `schroot` consente di processare un comando o una login shell in un ambiente `chrooted`

15.2. Come compilare il vostro primo pacchetto

15.2.1. Meta-Pacchetti o Fake Packages

I fake packages ed i metapacchetti sono simili fra loro e di fatto sono degli empty shell (contenitori vuoti) utilizzati meramente per l'influenza dei loro meta-dati sulla gestione dello stack (catena) dei pacchetti.

Lo scopo del fake package è far ritenere a `dpkg` ed `apt` che un pacchetto sia installato quando invece si tratta di un contenitore vuoto. Ciò consente di soddisfare le dipendenze di un pacchetto anche se il

software correlato è stato installato, ma al di fuori della portata del software di gestione dei pacchetti. Nonostante questo metodo funzioni, dovrete evitarlo se possibile in quanto non vi è alcuna garanzia che il software installato manualmente possa eguagliare le funzionalità del correlato pacchetto o che altri pacchetti, che dipendono dal software, funzionino appropriatamente.

Diversamente il metapacchetto è perlopiù una raccolta di dipendenze attraverso le quali vengono importati una serie di pacchetti in unica fase.

Per creare sia i metapacchetti, sia i fake packages potrete usare i programmi `equivs-control` e `equivs-build` (inclusi nel pacchetto Debian `equivs`). Il comando `equivs-control file` crea il file header (intestazione) del pacchetto Debian che dovrà essere modificato per indicare il nome del pacchetto desiderato, il suo numero di versione, il nome del manutentore, le sue dipendenze, la sua descrizione. Tutti gli altri campi senza un valore predefinito sono facoltativi e possono essere eliminati. I campi Copyright, Changelog, Readme ed Extra-Files non sono dei campi predefiniti nei pacchetti Debian; il loro unico scopo è essere alla portata di `equivs-build` e non rimarranno nelle intestazioni effettive del pacchetto generato.

Esempio 15.2 Il file header del `fake package` libxml-libxml-perl

```
Section: perl
Priority: optional
Standards-Version: 4.4.2

Package: libxml-libxml-perl
Version: 2.0134-1
Maintainer: Raphael Hertzog <hertzog@debian.org>
Depends: libxml2 (>= 2.7.4)
Architecture: all
Description: Fake package - module manually installed in site_perl
 This is a fake package to let the packaging system
 believe that this Debian package is installed.
 .
 In fact, the package is not installed since a newer version
 of the module has been manually compiled & installed in the
 site_perl directory.
```

Il passo successivo è generare il pacchetto Debian attraverso il comando `equivs-build file`. Fatto ciò il pacchetto sarà disponibile nella directory corrente e potrete gestirlo come qualsiasi pacchetto Debian.

15.2.2. Come gestire gli archivi dei files facilmente

Gli amministratori della Falcot Corp devono creare un pacchetto Debian in modo da poter distribuire facilmente un insieme di documenti su diverse macchine. L'amministratore incaricato a tale attività dopo aver letto "`New Maintainer's Guide`"avvia la creazione del suo primo pacchetto.

✦ https://www.debian.org/doc/manuals/maint-guide/

La prima fase consiste nella creazione di una directory `falcot-data-1.0`, che ospiterà il pacchetto sorgente target. Logicamente il pacchetto viene denominato `falcot-data` e riporta il numero di versione 1.0. Pertanto l'amministratore inserisce i files dei documenti in una subdirectory di dati. Successivamente invoca il comando `dh_make` (proveniente dal pacchetto `dh-make`) per aggiungere i files richiesti dal processo di generazione del pacchetto, che a sua volta viene memorizzato in una subdirectory di `debian`:

```
$ cd falcot-data-1.0
$ dh_make --native

Type of package: (single, indep, library, python)
[s/i/l/p] i

Maintainer name : Raphael Hertzog
Email-Address : hertzog@debian.org
Date : Fri, 04 Sep 2015 12:09:39 -0400
Package Name : falcot-data
Version : 1.0
License : gpl3
Package Type: indep
Are the details correct? [Y/n/q]
Currently there is no top level Makefile. This may require additional tuning.
Done. Please edit the files in the debian/ subdirectory now.

$
```

Il tipo selezionato per il pacchetto (`indep`) indica che il futuro pacchetto sorgente da generare sarà un singolo pacchetto binario che potrà essere utilizzato su diverse architetture (`Architecture:all`). Invece il tipo `single binary` è un'alternativa di `indep binary` ed indica un singolo pacchetto binario che dipende dall'architettura bersaglio (`Architecture:any`). È stato scelto il primo tipo nell'esempio sia in quanto il pacchetto dovrà ospitare solo documenti (e non programmi binari), sia per renderlo compatibile con tutte le architetture.

Il tipo `binary` diversamente deve essere utilizzato con un pacchetto sorgente che contiene diversi pacchetti binari. Il tipo `library` viene usato per le librerie condivise visto che devono seguire delle regole di gestione pacchetti rigorose.

SUGGERIMENTO
Nome del maintainer ed indirizzo email

La maggior parte dei programmi che si occupano della manutenzione dei pacchetti ricercano il vostro nome ed il vostro indirizzo email nelle variabili di ambiente `DEBFULLNAME` e `DEBEMAIL` o `EMAIL`. Definendo una volta per tutte le suddette variabili eviterete di doverle immettere più volte. Se normalmente usate come shell `bash`, aggiungete le seguenti righe nel vostro file `~/.bashrc` (ovviamente sostituendo i sottostanti valori con quelli pertinenti!):

```
export EMAIL="hertzog@debian.org"
export DEBFULLNAME="Raphael Hertzog"
```

Il programma `dh_make` ha creato una subdirectory di `debian` che contiene molti files. Alcuni files sono necessari, come `rules`, `control`, `changelog` e `copyright`. I files con estensione `.ex` sono files

di esempio che possono essere utilizzati per personalizzazioni se necessario (rimuovendo la loro estensione). Altrimenti, se non sono necessari è consigliabile rimuoverli. Il file `compat` deve rimanere incluso dato che è necessario per il corretto funzionamento dei programmi della suite denominata `debhelper` (i loro nomi iniziano con `dh_prefix`), utilizzati nelle varie fasi del processo di compilazione del pacchetto.

Il file `copyright` deve includere le informazioni riguardo agli autori dei documenti inclusi nel pacchetto e la relativa licenza. In questo caso, trattandosi di documenti interni, il loro utilizzo deve essere limitato alla società Falcot Corp. Il file `changelog` predefinito non necessita generalmente di modifiche; ciò nonostante l'amministratore ha preferito modificarlo sostituendo "`Initial release`" con una spiegazione leggermente più lunga e la distribuzione `unstable` con `internal`.
Il file `control` è stato aggiornato: il campo `Section` è stato sostituito con `misc` ed i campi `Homepage`, `Vcs-Git` e `Vcs-Browser` sono stati cancellati. Il campo `Depends` è stato completato da `firefox-esr | www-browser` per garantire la presenza di un browser web in grado di visualizzare i documenti inclusi nel pacchetto.

Esempio 15.3 Il file `control`

```
Source: falcot-data
Section: misc
Priority: optional
Maintainer: Raphael Hertzog <hertzog@debian.org>
Build-Depends: debhelper (>= 10)
Standards-Version: 4.4.1

Package: falcot-data
Architecture: all
Depends: firefox-esr | www-browser, ${misc:Depends}
Description: Internal Falcot Corp Documentation
   This package provides several documents describing the internal
   structure at Falcot Corp. This includes:
     - organization diagram
     - contacts for each department.
.
These documents MUST NOT leave the company.
Their use is INTERNAL ONLY.
```

Esempio 15.4 Il file `changelog`

```
falcot-data (1.0) internal; urgency=low

  * Initial Release.
  * Let's start with few documents:
    - internal company structure;
    - contacts for each department.

-- Raphael Hertzog <hertzog@debian.org> Fri, 04 Sep 2015 12:09:39 -0400
```

Esempio 15.5 Il file `copyright`

```
Format: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: falcot-data

Files: *
Copyright: 2004-2015 Falcot Corp
License:
 All rights reserved.
```

BASILARE

Il file `Makefile`

Il file `Makefile` è uno script impiegato dal programma `make`; il file `Makefile` definisce le regole che il programma `make` deve rispettare per compilare separatamente fra loro un set di files in una gerarchia di dipendenze (ad esempio un programma può essere compilato da una serie di files sorgente). Il file `Makefile` descrive le sue regole nel seguente formato:

```
target: source1 source2…
        command1
        command2
```

Questa regola può essere interpretata in questo modo: se uno dei files `source*` è più recente del file `target`, quest'ultimo deve essere generato eseguendo i comandi `command1` e `command2`.

Si precisa che le righe di comando devono iniziare con un tab character. Diversamente se la riga di comando inizia con un carattere dash (`-`), per quanto il comando possa fallire, l'intero processo non sarà interrotto.

Il file `rules` normalmente include un insieme di regole utilizzate per configurare, compilare ed installare il software in una subdirectory dedicata (omonima del pacchetto binario generato). Il contenuto della suddetta subdirectory viene archiviato nel pacchetto Debian come se la stessa subdirectory fosse la radice del filesystem. In questo caso, i files verranno installati nella subdirectory `debian/falcotdata/usr/share/falcot-data/` in modo che l'installazione del pacchetto generato sia distribuita nella subdirectory `/usr/share/falcot-data/`. Il file `rules` viene impiegato come se fosse un file `Makefile` con alcuni target predefiniti (inclusi `clean` e `binary` usati rispettivamente per "pulire" la directory della sorgente e generare il pacchetto binario).
Sebbene questo file sia il cuore del processo, spesso si limita allo stretto indispensabile affinché siano eseguiti una serie di comandi predefiniti offerti dal tool `debhelper`. Così avviene anche nel caso dei files generati da `dh_make`. Per installare i files dell'use-case preso ad esempio, occorre semplicemente personalizzare le funzionalità del comando `dh_install` creando il file `debian/falcot-data.install`:

```
data/* usr/share/falcot-data/
```

Giunti a questo punto potrete già creare il pacchetto. Tuttavia, è ancora possibile effettuare ulteriori modifiche. Difatti gli amministratori della Falcot Corp desiderano che i documenti siano facilmente accessibili dai menus dei ambienti grafici desktop: per fare ciò bisogna aggiungere un file `falcot-data.desktop` ed installarlo in `/usr/share/applications`, inserendo una seconda riga a `debian/falcot-data.install`.

Esempio 15.6 Il file falcot-data.desktop

```
[Desktop Entry]
Name=Internal Falcot Corp Documentation
Comment=Starts a browser to read the documentation
Exec=x-www-browser /usr/share/falcot-data/index.html
Terminal=false
Type=Application
Categories=Documentation;
```

Il file aggiornato `debian/falcot-data.install` si presenta così:

```
data/* usr/share/falcot-data/
falcot-data.desktop usr/share/applications/
```

Il pacchetto sorgente adesso è pronto. Non resta che generare il pacchetto binario con lo stesso metodo illustrato in precedenza per ricompilare i pacchetti: occorre eseguire all'interno della directory `falcot-data-1.0` il comando `dpkg-buildpackage -us -uc`.

15.3. Come creare un repository dei pacchetti per APT

La Falcot Corp gradualmente ha iniziato a manutenere diversi pacchetti Debian personalizzati o creati dal nulla per distribuire dati e programmi interni.

Per facilitare la distribuzione, la Falcot Corp vuole integrare i pacchetti in un archivio per pacchetti utilizzabili direttamente da APT. Per ovvie ragioni gestionali desiderano inoltre separare i pacchetti interni dai pacchetti localmente ricompilati. L'obiettivo è rendere disponibili le seguenti voci nel file `/etc/apt/sources.list.d/falcot.list`:

```
deb http://packages.falcot.com/ updates/
deb http://packages.falcot.com/ internal/
```

Gli amministratori pertanto configurano un virtual host sul loro HTTP server interno, con la seguente radice del correlato spazio web `/srv/vhosts/packages/`. La gestione dell'archivio è delegata al comando `mini-dinstall` (incluso nel pacchetto omonimo). Il suddetto comando esegue la scansione della directory `incoming/` (in questo caso `/srv/vhosts/packages/mini-dinstall/incoming/`) e rimane lì in ricezione in attesa di nuovi pacchetti; quando un pacchetto viene aggiornato, viene installato in un archivio Debian nella directory `/srv/vhosts/packages/`. Il comando `mini-dinstall` elabora il file `.changes` generato durante la creazione del pacchetto Debian. Questo tipo di file contiene l'elenco di tutti gli altri files associati alla versione del pacchetto (`*.deb`, `*.dsc`,`*.diff.gz`/`*.debian.tar.gz`, `*.orig.tar.gz` o altri files equivalenti che utilizzano altri formati di compressione) e consente a `mini-dinstall` di conoscere quali files deve installare. Per inciso, i files `.changes` includono anche il nome della distribuzione target (spesso `unstable`) definito nell'ultima voce di `debian/changelog`, e `mini-dinstall` utilizza tale informazione per stabilire dove dovrebbe installare il pacchetto. Per questo motivo gli

amministratori dovrebbero sempre modificare il suddetto campo prima di compilare un pacchetto e configurarlo con `internal` o `updates`, in base alla posizione desiderata. `mini-dinstall` dopodiché genera i files necessari per APT, ad esempio `Packages.gz`.

ALTERNATIVA apt-ftparchive e reprepo

Se l'impiego di `mini-dinstall` vi sembra troppo complesso rispetto alla vostra effettiva esigenza di un archivio Debian, potrete utilizzare direttamente il comando `apt-ftparchive`. Quest'ultimo ispeziona i contenuti di una directory e mostra (attraverso il suo output predefinito) un corrispondente file `Packages`. Nel caso della Falcot Corp, gli amministratori potrebbero caricare i pacchetti direttamente in `/srv/vhosts/packages/updates/` o `/srv/vhosts/packages/internal/` ed eseguire i seguenti comandi per creare i files `Packages.gz`:

```
$ cd /srv/vhosts/packages
$ apt-ftparchive packages updates >updates/Packages
$ gzip updates/Packages
$ apt-ftparchive packages internal >internal/Packages
$ gzip internal/Packages
```

Attraverso l'impiego del comando `apt-ftparchive sources` potrete creare, in modo del tutto simile al metodo soprastante, i files `Sources.gz`

`reprepro` è un tool designato allo stesso scopo, ma molto più avanzato. Può generare, gestire e sincronizzare un repository locale di pacchetti. In aggiunta conserva i pacchetti ed i checksums in un file Berkeley DB database, in modo che non sia necessario un server database. Attraverso `reprepro` potrete verificare le firme dei repositories dei mirrors e produrre le firme degli indici dei pacchetti generati.

La configurazione di `mini-dinstall` richiede che il file `~/.mini-dinstall.conf` venga a sua volta configurato; nel caso della Falcot Corp il suddetto file è il seguente:

```
[DEFAULT]
archive_style = flat
archivedir = /srv/vhosts/packages

verify_sigs = 0
mail_to = admin@falcot.com

generate_release = 1
release_origin = Falcot Corp
release_codename = stable

[updates]
release_label = Recompiled Debian Packages

[internal]
release_label = Internal Packages
```

La decisione di generare un file `Release` per ciascuno archivio merita la vostra attenzione. Difatti ciò consente di gestire le priorità di installazione dei pacchetti utilizzando il file di configurazione `/etc/apt/preferences` (per maggiori informazioni consultate il paragrafo 6.2.5 “Gestione delle Package Priorities“ a pagina 122).

SICUREZZA
mini-dinstall ed i permessi

`mini-dinstall` è stato progettato per essere eseguito da un qualsiasi utente regolare e non necessita di un account root. La soluzione più semplice è configurare tutto quanto all'interno dell'account utente dell'amministratore incaricato della creazione dei pacchetti Debian. Difatti solo l'amministratore ha i permessi necessari per inserire i files nella directory `incoming/` e di conseguenza viene presupposto che l'amministratore abbia verificato la provenienza di ciascun pacchetto prima della sua distribuzione senza che mini-dinstall necessiti di ripetere tale operazione. Ciò giustifica il parametro `verify_signs =0` (che indica che non occorre la verifica delle firme). Tuttavia, se il contenuto dei pacchetti è sensibile, potrete modificare tale configurazione autentificando attraverso delle chiavi pubbliche le persone autorizzate alla creazione dei pacchetti (tale configurazione richiede il parametro `extra_keyrings`); dopodiché `mini-dinstall` verificherà la provenienza di ogni pacchetto in entrata analizzando la firma integrata nel file `.changes`

L'invocazione di `mini-dinstall` di fatto avvia un demone in background. Finché il demone processa, `mini-dinstall` verifica ogni mezz'ora se ci sono nuovi pacchetti nella directory `incoming/`; quando è disponibile un nuovo pacchetto, quest'ultimo viene spostato nell'archivio e vengono rigenerati i correlati files `Packages.gz` e `Sources.gz`. Se l'esecuzione del demone è problematica, potrete invocare `mini-dinstall` in modalità batch [non interattiva] (attraverso l'opzione `-b`), ogni volta che un pacchetto viene immesso nella directory `incoming/`. Per conoscere altre funzionalità offerte da `mini-dinstall` consultate la man page `mini-dinstall(1)`.

EXTRA
Come generare un archivio firmato

La suite APT per impostazione predefinita esegue la verifica della catena delle firme crittografate dei pacchetti prima di installarli, in modo da garantirne l'autenticità (andate a consultare al riguardo il paragrafo 6.6 “Verifica dell'autenticità del pacchetto” a pagina 133). Gli archivi privati APT possono creare delle problematiche, dato che le macchine ne usufruiscono per i warnings (i messaggi di avviso) riguardo alle firme dei pacchetti. Pertanto un amministratore diligente dovrebbe integrare gli archivi privati in modo da renderli compatibili con il meccanismo di protezione di APT.
Per facilitare tale processo `mini-dinstall` include l'opzione di configurazione `release_signscript` che consente di definire uno script da utilizzare per generare la firma. Come modello potrete utilizzare lo script `sign-release.sh` offerto dal pacchetto `mini-dinstall` in `/usr/share/doc/mini-dinstall/examples/`; ovviamente la sua personalizzazione dipende dalle modifiche che avete effettuato localmente.

15.4. Come diventare il Maintainer di un Pacchetto

15.4.1. Come imparare a realizzare i Pacchetti

Realizzare un pacchetto Debian di qualità non è sempre un'attività facile; inoltre diventare un maintainer richiede molto studio ed un mix di pratica e teoria. Non si tratta quindi di una mera compilazione ed installazione del software; la complessità di tale attività richiede consapevolezza nel saper affrontare anomalie e conflitti, nonché le interazioni con migliaia di altri pacchetti software.

15.4.1.1 Le regole

Un pacchetto Debian deve essere conforme alle regole stabilite dalla Debian Policy ed ogni maintainers deve conoscerle. Ovviamente non deve conoscerle a memoria, ma deve essere consapevole della loro esistenza e deve saperle applicare per ponderare le decisioni appropriate ad un contesto non facile e senza alternative. Purtroppo a tutti i maintainers ufficiali di Debian è capitato di sbagliare per aver ignorato l'esistenza di una regola, ma non è un dramma dato che gli errori possono essere corretti grazie ai bug reports degli utenti (inoltre i bug reports sono piuttosto celeri grazie agli utenti avanzati). Il campo `Standards-Version` specifica in `debian/control` la versione della Debian Policy che il pacchetto rispetta. I Maintainers dovrebbero adeguare i loro pacchetti all'ultima versione della Debian Policy disponibile.

- https://www.debian.org/doc/debian-policy/

15.4.1.2 Le procedure

Debian non è una semplice raccolta di singoli pacchetti. I frutti del packaging di chiunque devono essere inclusi in un progetto collettivo; qualunque sviluppatore Debian coinvolto è consapevole che il progetto Debian procede come un'unica entità. Prima o poi, ciascun sviluppatore deve interagire con gli altri sviluppatori. Il Debian Developer's Reference (incluso nel pacchetto `developers-reference`) definisce sommariamente le regole comportamentali che ogni sviluppatore deve osservare quando si confronta con gli altri teams all'interno del progetto (possibilmente senza problemi), in modo che si possa beneficiare di tutte le risorse disponibili. Il suddetto documento definisce anche diversi doveri a cui ciascun sviluppatore deve attenersi.

- https://www.debian.org/doc/manuals/developers-reference/

15.4.1.3 Gli strumenti

Diversi strumenti facilitano le incombenze dei manutentori. Questo paragrafo li descrive brevemente, senza trattare tutti i loro dettagli, dato che sono ben documentati.

15.4.1.3.1 Il Programma `lintian`

Questo programma è uno degli strumenti più importanti: è il Debian package checker. Offre diversi test, creati in base alla Debian Policy, che consentono di rilevare rapidamente ed automaticamente diversi errori per correggerli prima del rilascio dei pacchetti.

Questo strumento è soltanto un mero ausilio [e non è definitivo] dato che talvolta fallisce (ad esempio quando la Debian Policy viene aggiornata e `lintian` no). Inoltre, non è esaustivo: l'assenza di errori rilevati non garantisce che un pacchetto è perfetto; al massimo che è esente dagli errori più comuni.

15.4.1.3.2 Il Programma `piuparts`

Questo è un altro importante strumento: automatizza l'installazione, l'aggiornamento, la rimozione e `purge` [la rimozione totale ed irreversibile] di un pacchetto (in un ambiente isolato) e verifica che nessuna delle suddette operazioni comportino errori. Inoltre facilita sia il rilevamento delle dipendenze mancanti, sia il rilevamento dei files ancora memorizzati nonostante il pacchetto sia stato eliminato in modalità purge.

15.4.1.3.3 devscripts

Il pacchetto devscripts include molti programmi che facilitano diversi aspetti delle funzioni di uno sviluppatore Debian:

- debuild consente di generare un pacchetto (dpkg-buildpackage) e di eseguire lintian per verificare immediatamente se è conforme alla Debian Policy.
- debclean pulisce un pacchetto sorgente dopo aver creato un pacchetto binario.
- dch consente rapidamente e facilmente di modificare un file debian/changelog in un pacchetto sorgente.
- uscan controlla se l'autore a monte ha rilasciato una nuova versione del software. Ciò però richiede un file debian/watch che descriva la posizione dei rilasci.
- debi consente l'installazione (con dpkg -i) di un pacchetto Debian che è stato appena generato senza che si debba battere sul computer il suo nome completo o percorso.
- debc serve per consultare il contenuto (con dpkg -i) di un pacchetto Debian che è stato appena generato senza che si debba battere sul computer il suo nome completo o percorso.
- bts gestisce il bug tracking system da riga di comando; questo programma genera automaticamente le email correlate.
- debrelease carica un nuovo pacchetto generato su un server remoto senza che si debba battere sul computer il nome completo o il percorso del file .changes correlato.
- debsign firma i files .dsc e .changes.
- uupdate automaticamente crea una nuova revisione di un pacchetto quando ne viene rilasciata una nuova versione a monte.

15.4.1.3.4 debhelper e dh-make

debhelper è un insieme di scripts che semplificano la creazione dei pacchetti in conformità alla Debian Policy; i suddetti scripts sono invocati da debian/rules. debhelper è stato ampiamente adottato nella comunità di Debian difatti viene impiegato dalla maggior parte dei pacchetti Debian. Tutti i comandi che debhelper include, hanno un prefisso dh_ .

Lo script dh_make (disponibile nel pacchetto dh-make) crea i files necessari per generare un pacchetto Debian nella directory che inizialmente include le sorgenti della componente del software. Come facilmente intuibile dal nome, lo script dh_make genera i files utilizzando debhelper per impostazione predefinita.

15.4.1.3.5 autopkgtest

autopkgtest esegue i tests sui pacchetti binari - tests che sono offerti dallo stesso pacchetto sorgente.

15.4.1.3.6 reprotest

reprotest compila lo stesso codice sorgente due volte in due ambienti differenti per poi verificarne le differenze. Se vengono rilevate differenze viene utilizzato diffoscope (se non è disponibile diff) in modo da visualizzarle dettagliatamente per una loro analisi.

15.4.1.3.7 dupload e dput

I comandi dupload e dput sono usati per caricare un pacchetto Debian su un server (possibilmente remoto). Ciò consente agli sviluppatori di publicare i loro pacchetti sul main server di Debian (ftp-

master.debian.org) così che vengano integrati nell'archivio e distribuiti dai mirrors. Questi comandi prendono in considerazione come parametro un file `.changes` per dedurre dai contenuti gli altri files da inviare.

15.4.2. Il processo di approvazione

Diventare uno sviluppatore Debian non è una mera questione amministrativa. Il processo comprende diverse fasi tra cui un percorso di iniziazione/selezione. Tale processo è formalizzato e documentato in modo che chiunque possa seguire i progressi dei candidati su un sito web dedicato al reclutamento dei nuovi membri.

✦ https://nm.debian.org/

EXTRA	
Processo semplificato per i "Manutentori Debian"	È stato introdotto lo status di "Debian Maintainer" [il cui acronimo è DM] ed il correlato processo di reclutamento è più rapido. Attraverso tale status i contributori possono manutenere i loro stessi pacchetti. Occorre una mera verifica di uno sviluppatore Debian su ciò che il contributore ha caricato per la prima volta che attesti attraverso una regolare istanza le capacità del futuro maintainer nel prendersi cura del proprio pacchetto.

15.4.2.1 Prerequisiti

Tutti i candidati sono tenuti ad avere una basilare padronanza della lingua inglese. Ciò è necessario in qualsiasi ambito: sia in fase ex ante per comunicare con l'esaminatore, sia ex post in quanto l'inglese è la lingua per antonomasia per gran parte della documentazione; inoltre, gli utenti che segnaleranno i bugs dei pacchetti lo faranno in inglese e di conseguenza si aspetteranno delle risposte sempre in inglese.

Il secondo prerequisito riguarda la motivazione. Il candidato per la posizione di sviluppatore Debian deve essere consapevole che il suo interesse in Debian deve perdurare per diversi mesi. Lo stesso processo di approvazione in sé dura diversi mesi e Debian ha bisogno di manutentori a lungo termine; ciascun pacchetto necessita di essere manutenuto e non solo di essere inizialmente caricato.

15.4.2.2 Registrazione

La prima (effettiva) fase consiste nel trovare uno sponsor o sostenitore; ossia uno sviluppatore Debian deve dichiarare che il candidato X può essere vantaggioso per Debian. Ciò implica che il candidato è già stato attivo nella comunità e che la sua attività è stata apprezzata. Se il candidato è restio a mostrare pubblicamente il suo operato, può mostrare i suoi frutti in forma riservata ad uno sviluppatore Debian ufficiale in modo da convincerlo a sostenerlo.

Inoltre il candidato deve generare una coppia di chiavi RSA pubblica/privata attraverso GnuPG, che dovranno essere contrassegnate da almeno due sviluppatori Debian ufficiali. La firma attesta l'autenticità del nome presente sulla chiave. Infatti, durante una `key signing party` [è di fatto una vera e propria riunione] è consuetudine che i partecipanti presentino i loro documenti d'identità (solitamente una carta d'identità o un passaporto) e gli identificatori delle chiavi. Questa fase consente di formalizzare la corrispondenza tra la persona fisica e le chiavi. Pertanto la firma richiede

un incontro reale. Se non avete mai avuto la possibilità di incontrare uno sviluppatore Debian ad una conferenza di software libero, potrete fare esplicita richiesta agli sviluppatori che abitano nelle vostre vicinanze utilizzando la seguente pagina web:

✦ https://wiki.debian.org/Keysigning

Una volta che la registrazione su nm.debian.org è stata convalidata dallo sponsor, un `Application Manager` viene assegnato al candidato. L'Application Manager si occuperà di assistere il processo del candidato sia nelle fasi previste, sia nei vari controlli.

La prima verifica riguarda l'identità. Se siete già in possesso di una chiave firmata da due sviluppatori Debian, questa fase sarà piuttosto facile da superare; in caso contrario, l'Application Manager aiuterà il candidato ad organizzare un incontro con degli sviluppatori Debian per il key-signing.

15.4.2.3 Accettazione dei principi

Queste formalità amministrative sono dovute a considerazioni filosofiche. È necessario che il candidato comprenda il contratto sociale ed accetti i principi del software libero. Infatti, per entrare in Debian occorre condividere i valori che accumunano gli sviluppatori definiti nei documenti fondanti (e trattati nel Capitolo 1, “Il Progetto Debian” a pagina 2).

In aggiunta, ciascun candidato è tenuto a conoscere come opera il progetto e come interagire appropriatamente per affrontare i problemi che indubbiamente incontrerà. Tutte queste informazioni sono generalmente documentate sia nei manuali destinati ai nuovi manutentori, sia nella guida di riferimento dedicata agli sviluppatori Debian.
Una attenta lettura del suddetto documento dovrebbe essere sufficiente per rispondere alla domande dell'esaminatore. Se le risposte non sono soddisfacenti, il candidato verrà informato. Dopodiché il candidato dovrà leggere attentamente (di nuovo) la documentazione prima di riprovarci. Qualora la documentazione disponibile non fosse sufficiente per rispondere ad un quesito, il candidato potrà dedurre una risposta avvalendosi della consulenza, attraverso una mera discussione, di un altro sviluppatore Debian oppure attraverso la sua esperienza in Debian. Tale meccanismo garantisce che i candidati siano partecipi nel progetto Debian ancor prima di farne effettivamente parte. Inoltre questa politica è stata ben ponderata in modo che le persone si aggreghino al progetto e si adattino come un pezzo aggiuntivo di un puzzle espandibile all'infinito.

Questo passaggio viene comunemente denominato, nel gergo degli sviluppatori coinvolti nel processo di selezione dei nuovi membri, `Philosophy & Procedures (P&P)`.

15.4.2.4 Verifica delle competenze

Qualsiasi candidatura per diventare uno sviluppatore Debian ufficiale deve essere convalidata. Occorre dimostrare che la richiesta dello status di membro effettivo del progetto è legittima e facilita l'attività del candidato che contribuisce al progetto Debian. Solitamente la candidatura viene giustificata in quanto lo status di sviluppatore Debian facilita la manutenzione di un pacchetto Debian, ma non è l'unica causa che può essere utilizzata per avallare una richiesta. Difatti alcuni sviluppatori aderiscono al progetto per contribuire al porting su una specifica architettura, altri vogliono migliorare la documentazione, ecc.

Pertanto questa fase rappresenta un'opportunità per ciascun candidato per dichiarare l'obiettivo che intende raggiungere nel progetto Debian e per dimostrare quanto ha già svolto in tal senso. Debian inoltre è un progetto piuttosto pragmatico e non è sufficiente dichiarare qualcosa, se le azioni non corrispondono a quanto precedentemente asserito. In generale, quando come ruolo viene proposta la manutenzione di un pacchetto, occorre trovare in fase ex ante uno sponsor tra gli sviluppatori ufficiali per verificare tecnicamente la prima versione del pacchetto e caricarla nei servers Debian.

COMUNITÀ Sponsorizzazione	Gli sviluppatori Debian possono sponsorizzare i pacchetti preparati da terzi, ovvero possono pubblicarli nei repositories ufficiali di Debian dopo aver effettuato una revisione approfondita. Ciò consente ad individui estranei al progetto, che non hanno ancora affrontato il processo di selezione, di contribuire occasionalmente al progetto. Nel contempo tale meccanismo garantisce che i pacchetti offerti da Debian siano sempre sottoposti a verifica da un membro ufficiale.

Infine, l'esaminatore verifica le competenze tecniche (packaging) del candidato tramite un ampio questionario. Non sono ammessi errori, ma il tempo per rispondere alle domande non è limitato. Viene resa disponibile tutta la documentazione e se le prime risposte non sono soddisfacenti sono possibili altri tentativi. Il questionario non è discriminatorio, ma garantisce un livello minimo di conoscenza dei nuovi contributori.
Questo fase è denominata `Tasks & Skills` (`T&S`) nel gergo degli esaminatori.

15.4.2.5 Approvazione finale

L'ultimo passaggio è la convalida dell'intero processo da parte di un DAM (`Debian Account Manager`). Il DAM revisionerà tutte le informazioni che l'esaminatore ha raccolto sul candidato per poi stabilire se creargli o meno un account sui servers Debian. Se occorrono ulteriori informazioni posticiperà la creazione dell'account. Le opposizioni sono piuttosto rare se l'esaminatore ha svolto bene il suo lavoro, ma a volte capitano. Non sono mai definitive ed il candidato è libero di riprovare nuovamente.

La decisione del DAM è definitiva, quasi mai appellabile, e ciò spiega come mai chi si ritrova in tale posizione è stato spesso in passato oggetto di critiche.

Parole chiave

Futuro
Migliorie
Opinioni