Pacchettizzare in Debian: come e perché

Matteo F. Vescovi

(Free Software Users Group Padova)

Sesta serata a tema del Primo Ciclo

Padova, 7 Dicembre 2011

Sommario

Sommario

Quali sono gli strumenti dei quali si avrà bisogno?

- Ovviamente, un sistema Debian (od Ubuntu) con accesso root
- Alcuni pacchetti:
  - **build-essential** ha le dipendenze su tutto i pacchetti che si presuppone debbano essere disponibili sulla macchina di uno sviluppatore, includendo la dipendenza su dpkg-dev che contiene gli strumenti di base specifici di Debian nella creazione di pacchetti
  - **devscripts** contiene molti scripts utili ai manutentori Debian

A questi si consiglia di affiancare altri pacchetti di sviluppo, quali:

- **git** per la gestione del versioning del sorgente
- **lintian** per il controllo della qualità del pacchetto
- **quilt** per la gestione delle patches applicate al sorgente

Quali sono gli strumenti dei quali si avrà bisogno?

- Ovviamente, un sistema Debian (od Ubuntu) con accesso root
- Alcuni pacchetti:

  build-essential ha le dipendenze su tutto i pacchetti che si presuppone debbano essere disponibili sulla macchina di uno sviluppatore, includendo la dipendenza su dpkg-dev che contiene gli strumenti di base specifici di Debian nella creazione di pacchetti

  devscripts contiene molti scripts utili ai manutentori Debian

A questi si consiglia di affiancare altri pacchetti di sviluppo, quali:

git per la gestione del versioning del sorgente

lintian per il controllo della qualità del pacchetto

quilt per la gestione delle patches applicate al sorgente

Quali sono gli strumenti dei quali si avrà bisogno?

- Ovviamente, un sistema Debian (od Ubuntu) con accesso root
- Alcuni pacchetti:

  build-essential ha le dipendenze su tutto i pacchetti che si presuppone debbano essere disponibili sulla macchina di uno sviluppatore, includendo la dipendenza su `dpkg-dev` che contiene gli strumenti di base specifici di Debian nella creazione di pacchetti

  devscripts contiene molti scripts utili ai manutentori Debian

A questi si consiglia di affiancare altri pacchetti di sviluppo, quali:

- git per la gestione del versioning del sorgente
- lintian per il controllo della qualità del pacchetto
- quilt per la gestione delle patches applicate al sorgente

Quali sono gli strumenti dei quali si avrà bisogno?

- Ovviamente, un sistema Debian (od Ubuntu) con accesso root
- Alcuni pacchetti:

  build-essential ha le dipendenze su tutto i pacchetti che si presuppone debbano essere disponibili sulla macchina di uno sviluppatore, includendo la dipendenza su `dpkg-dev` che contiene gli strumenti di base specifici di Debian nella creazione di pacchetti

  devscripts contiene molti scripts utili ai manutentori Debian

A questi si consiglia di affiancare altri pacchetti di sviluppo, quali:

- git per la gestione del versioning del sorgente
- lintian per il controllo della qualità del pacchetto
- quilt per la gestione delle patches applicate al sorgente

Quali sono gli strumenti dei quali si avrà bisogno?

- Ovviamente, un sistema Debian (od Ubuntu) con accesso root
- Alcuni pacchetti:

  build-essential ha le dipendenze su tutto i pacchetti che si presuppone debbano essere disponibili sulla macchina di uno sviluppatore, includendo la dipendenza su `dpkg-dev` che contiene gli strumenti di base specifici di Debian nella creazione di pacchetti

  devscripts contiene molti scripts utili ai manutentori Debian

A questi si consiglia di affiancare altri pacchetti di sviluppo, quali:

- git per la gestione del versioning del sorgente
- lintian per il controllo della qualità del pacchetto
- quilt per la gestione delle patches applicate al sorgente

Quali sono gli strumenti dei quali si avrà bisogno?

- Ovviamente, un sistema Debian (od Ubuntu) con accesso root
- Alcuni pacchetti:

  build-essential ha le dipendenze su tutto i pacchetti che si presuppone debbano essere disponibili sulla macchina di uno sviluppatore, includendo la dipendenza su `dpkg-dev` che contiene gli strumenti di base specifici di Debian nella creazione di pacchetti

  devscripts contiene molti scripts utili ai manutentori Debian

A questi si consiglia di affiancare altri pacchetti di sviluppo, quali:

git per la gestione del versioning del sorgente

lintian per il controllo della qualità del pacchetto

quilt per la gestione delle patches applicate al sorgente

Quali sono gli strumenti dei quali si avrà bisogno?

- Ovviamente, un sistema Debian (od Ubuntu) con accesso root
- Alcuni pacchetti:

  build-essential ha le dipendenze su tutto i pacchetti che si presuppone debbano essere disponibili sulla macchina di uno sviluppatore, includendo la dipendenza su `dpkg-dev` che contiene gli strumenti di base specifici di Debian nella creazione di pacchetti

  devscripts contiene molti scripts utili ai manutentori Debian

A questi si consiglia di affiancare altri pacchetti di sviluppo, quali:

git per la gestione del versioning del sorgente

lintian per il controllo della qualità del pacchetto

quilt per la gestione delle patches applicate al sorgente

Quali sono gli strumenti dei quali si avrà bisogno?

- Ovviamente, un sistema Debian (od Ubuntu) con accesso root
- Alcuni pacchetti:

  build-essential ha le dipendenze su tutto i pacchetti che si presuppone debbano essere disponibili sulla macchina di uno sviluppatore, includendo la dipendenza su `dpkg-dev` che contiene gli strumenti di base specifici di Debian nella creazione di pacchetti

  devscripts contiene molti scripts utili ai manutentori Debian

A questi si consiglia di affiancare altri pacchetti di sviluppo, quali:

git per la gestione del versioning del sorgente

lintian per il controllo della qualità del pacchetto

quilt per la gestione delle patches applicate al sorgente

Creare un pacchetto sorgente di base

- Scaricare il codice sorgente “upstream”
- Rinominare il sorgente
- Scompattare il sorgente scaricato
- lanciare `dh-make`

Viene cosí creata la directory `debian/`, che contiene tutti i files necessari alla pacchettizzazione. . . e molto altro.

Creare un pacchetto sorgente di base

- Scaricare il codice sorgente “upstream”
- Rinominare il sorgente
- Scompattare il sorgente scaricato
- lanciare `dh-make`

Viene cosí creata la directory `debian/`, che contiene tutti i files necessari alla pacchettizzazione. . . e molto altro.

Creare un pacchetto sorgente di base

- Scaricare il codice sorgente “upstream”
- Rinominare il sorgente
- Scompattare il sorgente scaricato
- lanciare `dh-make`

Viene cosí creata la directory `debian/`, che contiene tutti i files necessari alla pacchettizzazione. . . e molto altro.

Creare un pacchetto sorgente di base

- Scaricare il codice sorgente “upstream”
- Rinominare il sorgente
- Scompattare il sorgente scaricato
- lanciare `dh-make`

Viene cosí creata la directory `debian/`, che contiene tutti i files necessari alla pacchettizzazione. . . e molto altro.

Creare un pacchetto sorgente di base

- Scaricare il codice sorgente “upstream”
- Rinominare il sorgente
- Scompattare il sorgente scaricato
- lanciare `dh-make`

Viene cosí creata la directory `debian/`, che contiene tutti i files necessari alla pacchettizzazione. . . e molto altro.

Creare un pacchetto sorgente di base

- Scaricare il codice sorgente “upstream”
- Rinominare il sorgente
- Scompattare il sorgente scaricato
- lanciare `dh-make`

Viene cosí creata la directory `debian/`, che contiene tutti i files necessari alla pacchettizzazione. . . e molto altro.

Files in debian/

changelog file contenente la “storia” del pacchetto

control file di meta-data circa il pacchetto

copyright file contenente le informazioni di copyright del pacchetto

rules file che specifica come debba esser costruito il pacchetto

Gli altri files presenti (e non sempre necessari) sono:

- `compat`
- `watch`
- i target di dh_install (`*.dirs`, `*.docs`, etc)
- gli scripts del maintainer (`*.preinst`, `*.postinst`, etc)
- `source/format`, che definisce il formato del sorgente
- `patches/`, che contiene le patches al sorgente upstream

Files in debian/

changelog file contenente la "storia" del pacchetto

control file di meta-data circa il pacchetto

copyright file contenente le informazioni di copyright del pacchetto

rules file che specifica come debba esser costruito il pacchetto

Gli altri files presenti (e non sempre necessari) sono:

- `compat`
- `watch`
- i target di dh_install (`*.dirs`, `*.docs`, etc)
- gli scripts del maintainer (`*.preinst`, `*.postinst`, etc)
- `source/format`, che definisce il formato del sorgente
- `patches/`, che contiene le patches al sorgente upstream

Files in debian/

changelog file contenente la “storia” del pacchetto

control file di meta-data circa il pacchetto

copyright file contenente le informazioni di copyright del pacchetto

rules file che specifica come debba esser costruito il pacchetto

Gli altri files presenti (e non sempre necessari) sono:

- `compat`
- `watch`
- i target di `dh_install` (`*.dirs`, `*.docs`, etc)
- gli scripts del maintainer (`*.preinst`, `*.postinst`, etc)
- `source/format`, che definisce il formato del sorgente
- `patches/`, che contiene le patches al sorgente upstream

Files in debian/

changelog file contenente la “storia” del pacchetto

control file di meta-data circa il pacchetto

copyright file contenente le informazioni di copyright del pacchetto

rules file che specifica come debba esser costruito il pacchetto

Gli altri files presenti (e non sempre necessari) sono:

- `compat`
- `watch`
- i target di `dh_install` (`*.dirs`, `*.docs`, etc)
- gli scripts del maintainer (`*.preinst`, `*.postinst`, etc)
- `source/format`, che definisce il formato del sorgente
- `patches/`, che contiene le patches al sorgente upstream

Files in debian/

changelog file contenente la “storia” del pacchetto

control file di meta-data circa il pacchetto

copyright file contenente le informazioni di copyright del pacchetto

rules file che specifica come debba esser costruito il pacchetto

Gli altri files presenti (e non sempre necessari) sono:

- `compat`
- `watch`
- i target di `dh_install` (`*.dirs`, `*.docs`, etc)
- gli scripts del maintainer (`*.preinst`, `*.postinst`, etc)
- `source/format`, che definisce il formato del sorgente
- `patches/`, che contiene le patches al sorgente upstream

Files in debian/

changelog file contenente la “storia” del pacchetto

control file di meta-data circa il pacchetto

copyright file contenente le informazioni di copyright del pacchetto

rules file che specifica come debba esser costruito il pacchetto

Gli altri files presenti (e non sempre necessari) sono:

- `compat`
- `watch`
- i target di dh_install (`*.dirs`, `*.docs`, etc)
- gli scripts del maintainer (`*.preinst`, `*.postinst`, etc)
- `source/format`, che definisce il formato del sorgente
- `patches/`, che contiene le patches al sorgente upstream

Files in debian/

changelog file contenente la “storia” del pacchetto

control file di meta-data circa il pacchetto

copyright file contenente le informazioni di copyright del pacchetto

rules file che specifica come debba esser costruito il pacchetto

Gli altri files presenti (e non sempre necessari) sono:

- `compat`
- `watch`
- i target di dh_install (`*.dirs`, `*.docs`, etc)
- gli scripts del maintainer (`*.preinst`, `*.postinst`, etc)
- `source/format`, che definisce il formato del sorgente
- `patches/`, che contiene le patches al sorgente upstream

Files in debian/

changelog file contenente la “storia” del pacchetto

control file di meta-data circa il pacchetto

copyright file contenente le informazioni di copyright del pacchetto

rules file che specifica come debba esser costruito il pacchetto

Gli altri files presenti (e non sempre necessari) sono:

- `compat`
- `watch`
- i target di dh_install (`*.dirs`, `*.docs`, `etc`)
- gli scripts del maintainer (`*.preinst`, `*.postinst`, `etc`)
- `source/format`, che definisce il formato del sorgente
- `patches/`, che contiene le patches al sorgente upstream

Files in debian/

changelog file contenente la “storia” del pacchetto

control file di meta-data circa il pacchetto

copyright file contenente le informazioni di copyright del pacchetto

rules file che specifica come debba esser costruito il pacchetto

Gli altri files presenti (e non sempre necessari) sono:

- `compat`
- `watch`
- i target di dh_install (`*.dirs`, `*.docs`, `etc`)
- gli scripts del maintainer (`*.preinst`, `*.postinst`, `etc`)
- `source/format`, che definisce il formato del sorgente
- `patches/`, che contiene le patches al sorgente upstream

Files in debian/

changelog file contenente la "storia" del pacchetto

control file di meta-data circa il pacchetto

copyright file contenente le informazioni di copyright del pacchetto

rules file che specifica come debba esser costruito il pacchetto

Gli altri files presenti (e non sempre necessari) sono:

- `compat`
- `watch`
- i target di dh_install (`*.dirs`, `*.docs`, etc)
- gli scripts del maintainer (`*.preinst`, `*.postinst`, etc)
- `source/format`, che definisce il formato del sorgente
- `patches/`, che contiene le patches al sorgente upstream

Files in debian/

changelog file contenente la “storia” del pacchetto

control file di meta-data circa il pacchetto

copyright file contenente le informazioni di copyright del pacchetto

rules file che specifica come debba esser costruito il pacchetto

Gli altri files presenti (e non sempre necessari) sono:

- `compat`
- `watch`
- i target di dh_install (`*.dirs`, `*.docs`, `etc`)
- gli scripts del maintainer (`*.preinst`, `*.postinst`, `etc`)
- `source/format`, che definisce il formato del sorgente
- `patches/`, che contiene le patches al sorgente upstream

debian/changelog

- Stila la lista dei cambiamenti nella pacchettizzazione Debian
- Definisce la versione attuale del pacchetto
- È modificato manualmente oppure attraverso `dch`
- Sfrutta formati speciali per chiudere automaticamente i bugs di Debian (`Closes: #123456`) od Ubuntu (`LP: #654321`)

debian/changelog

- Stila la lista dei cambiamenti nella pacchettizzazione Debian
- Definisce la versione attuale del pacchetto
- È modificato manualmente oppure attraverso `dch`
- Sfrutta formati speciali per chiudere automaticamente i bugs di Debian (`Closes: #123456`) od Ubuntu (`LP: #654321`)

debian/changelog

- Stila la lista dei cambiamenti nella pacchettizzazione Debian
- Definisce la versione attuale del pacchetto
- È modificato manualmente oppure attraverso `dch`
- Sfrutta formati speciali per chiudere automaticamente i bugs di Debian (`Closes: #123456`) od Ubuntu (`LP: #654321`)

debian/changelog

- Stila la lista dei cambiamenti nella pacchettizzazione Debian
- Definisce la versione attuale del pacchetto
- È modificato manualmente oppure attraverso `dch`
- Sfrutta formati speciali per chiudere automaticamente i bugs di Debian (`Closes: #123456`) od Ubuntu (`LP: #654321`)

debian/control

- Contiene i meta-data:
  - per il pacchetto sorgente in sè
  - per tutti i pacchetti binari generati da tale sorgente
- Definisce:
  - il nome del pacchetto
  - la sezione cui tale pacchetto appartiene
  - la priorità del pacchetto nell'upload
  - il maintainer (nome e mail)
  - gli uploaders
  - etc...
- Definisce tutte le descrizioni, seguendo le direttive del capitolo 5 del *Debian Policy Manual*

debian/control

- Contiene i meta-data:
  - per il pacchetto sorgente in sè
  - per tutti i pacchetti binari generati da tale sorgente
- Definisce:
  - il nome del pacchetto
  - la sezione cui tale pacchetto appartiene
  - la priorità del pacchetto nell'upload
  - il maintainer (nome e mail)
  - gli uploaders
  - etc...
- Definisce tutte le descrizioni, seguendo le direttive del capitolo 5 del *Debian Policy Manual*

debian/control

- Contiene i meta-data:
  - per il pacchetto sorgente in sè
  - per tutti i pacchetti binari generati da tale sorgente
- Definisce:
  - il nome del pacchetto
  - la sezione cui tale pacchetto appartiene
  - la priorità del pacchetto nell'upload
  - il maintainer (nome e mail)
  - gli uploaders
  - etc...
- Definisce tutte le descrizioni, seguendo le direttive del capitolo 5 del *Debian Policy Manual*

debian/control

- Contiene i meta-data:
  - per il pacchetto sorgente in sè
  - per tutti i pacchetti binari generati da tale sorgente
- Definisce:
  - il nome del pacchetto
  - la sezione cui tale pacchetto appartiene
  - la priorità del pacchetto nell'upload
  - il maintainer (nome e mail)
  - gli uploaders
  - etc. . .
- Definisce tutte le descrizioni, seguendo le direttive del capitolo 5 del *Debian Policy Manual*

debian/control

- Contiene i meta-data:
  - per il pacchetto sorgente in sè
  - per tutti i pacchetti binari generati da tale sorgente
- Definisce:
  - il nome del pacchetto
  - la sezione cui tale pacchetto appartiene
  - la priorità del pacchetto nell'upload
  - il maintainer (nome e mail)
  - gli uploaders
  - etc. . .
- Definisce tutte le descrizioni, seguendo le direttive del capitolo 5 del *Debian Policy Manual*

debian/control

- Contiene i meta-data:
  - per il pacchetto sorgente in sè
  - per tutti i pacchetti binari generati da tale sorgente
- Definisce:
  - il nome del pacchetto
  - la sezione cui tale pacchetto appartiene
  - la priorità del pacchetto nell'upload
  - il maintainer (nome e mail)
  - gli uploaders
  - etc. . .
- Definisce tutte le descrizioni, seguendo le direttive del capitolo 5 del *Debian Policy Manual*

debian/control

- Contiene i meta-data:
  - per il pacchetto sorgente in sè
  - per tutti i pacchetti binari generati da tale sorgente
- Definisce:
  - il nome del pacchetto
  - la sezione cui tale pacchetto appartiene
  - la priorità del pacchetto nell'upload
  - il maintainer (nome e mail)
  - gli uploaders
  - etc. . .
- Definisce tutte le descrizioni, seguendo le direttive del capitolo 5 del *Debian Policy Manual*

debian/control

- Contiene i meta-data:
  - per il pacchetto sorgente in sè
  - per tutti i pacchetti binari generati da tale sorgente
- Definisce:
  - il nome del pacchetto
  - la sezione cui tale pacchetto appartiene
  - la priorità del pacchetto nell'upload
  - il maintainer (nome e mail)
  - gli uploaders
  - etc. . .
- Definisce tutte le descrizioni, seguendo le direttive del capitolo 5 del *Debian Policy Manual*

debian/control

- Contiene i meta-data:
  - per il pacchetto sorgente in sè
  - per tutti i pacchetti binari generati da tale sorgente
- Definisce:
  - il nome del pacchetto
  - la sezione cui tale pacchetto appartiene
  - la priorità del pacchetto nell'upload
  - il maintainer (nome e mail)
  - gli uploaders
  - etc. . .
- Definisce tutte le descrizioni, seguendo le direttive del capitolo 5 del *Debian Policy Manual*

debian/control

- Contiene i meta-data:
  - per il pacchetto sorgente in sè
  - per tutti i pacchetti binari generati da tale sorgente
- Definisce:
  - il nome del pacchetto
  - la sezione cui tale pacchetto appartiene
  - la priorità del pacchetto nell'upload
  - il maintainer (nome e mail)
  - gli uploaders
  - etc...
- Definisce tutte le descrizioni, seguendo le direttive del capitolo 5 del *Debian Policy Manual*

debian/control

- Contiene i meta-data:
  - per il pacchetto sorgente in sè
  - per tutti i pacchetti binari generati da tale sorgente
- Definisce:
  - il nome del pacchetto
  - la sezione cui tale pacchetto appartiene
  - la priorità del pacchetto nell'upload
  - il maintainer (nome e mail)
  - gli uploaders
  - etc...
- Definisce tutte le descrizioni, seguendo le direttive del capitolo 5 del *Debian Policy Manual*

debian/rules

- È il *makefile* della pacchettizzazione
- È l'interfaccia utilizzata per generare i pacchetti Debian
- È documentato nel §4.9 del *Debian Policy Manual*

debian/rules

- È il *makefile* della pacchettizzazione
- È l'interfaccia utilizzata per generare i pacchetti Debian
- È documentato nel §4.9 del *Debian Policy Manual*

debian/rules

- È il *makefile* della pacchettizzazione
- È l'interfaccia utilizzata per generare i pacchetti Debian
- È documentato nel §4.9 del *Debian Policy Manual*

Helpers per la pacchettizzazione – *debhelper*

. . . perché non usarli sarebbe da pazzi!

- La miglior pratica è l'uso di un *aiutante* di pacchettizzazione
- In assoluto il più noto è debhelper, usato per il 98% dei pacchetti presenti negli archivi Debian
- Suoi obiettivi:
  1. "travasare" le attività comuni della pacchettizzazione in strumenti standard utilizzabili da tutti i pacchetti
  2. correggere i difetti di pacchettizzazione per tutti i pacchetti in un colpo solo, grazie alla famiglia `dh_*`
- È richiamabile da `debian/rules`
- È configurabile sfruttando i parametri del comando o i files contenuti in `debian/` (e.g., *`package`*`.install`)

Come si potrà sfruttare tutto questo?

`debian/compat` dovrà essere impostato per utilizzare una versione di compatibilità ≥ 7

Helpers per la pacchettizzazione – *debhelper*

. . . perché non usarli sarebbe da pazzi!

- La miglior pratica è l'uso di un *aiutante* di pacchettizzazione
- In assoluto il più noto è debhelper, usato per il 98% dei pacchetti presenti negli archivi Debian
- Suoi obiettivi:
  1. "travasare" le attività comuni della pacchettizzazione in strumenti standard utilizzabili da tutti i pacchetti
  2. correggere i difetti di pacchettizzazione per tutti i pacchetti in un colpo solo, grazie alla famiglia `dh_*`
- È richiamabile da `debian/rules`
- È configurabile sfruttando i parametri del comando o i files contenuti in `debian/` (e.g., *`package`*`.install`)

Come si potrà sfruttare tutto questo?

`debian/compat` dovrà essere impostato per utilizzare una versione di compatibilità ≥ 7

Helpers per la pacchettizzazione – *debhelper*

. . . perché non usarli sarebbe da pazzi!

- La miglior pratica è l'uso di un *aiutante* di pacchettizzazione
- In assoluto il più noto è debhelper, usato per il 98% dei pacchetti presenti negli archivi Debian
- Suoi obiettivi:
  1. "travasare" le attività comuni della pacchettizzazione in strumenti standard utilizzabili da tutti i pacchetti
  2. correggere i difetti di pacchettizzazione per tutti i pacchetti in un colpo solo, grazie alla famiglia `dh_*`
- È richiamabile da `debian/rules`
- È configurabile sfruttando i parametri del comando o i files contenuti in `debian/` (e.g., *`package`*`.install`)

Come si potrà sfruttare tutto questo?

`debian/compat` dovrà essere impostato per utilizzare una versione di compatibilità ≥ 7

Helpers per la pacchettizzazione – *debhelper*

. . . perché non usarli sarebbe da pazzi!

- La miglior pratica è l'uso di un *aiutante* di pacchettizzazione
- In assoluto il più noto è debhelper, usato per il 98% dei pacchetti presenti negli archivi Debian
- Suoi obiettivi:
  1. "travasare" le attività comuni della pacchettizzazione in strumenti standard utilizzabili da tutti i pacchetti
  2. correggere i difetti di pacchettizzazione per tutti i pacchetti in un colpo solo, grazie alla famiglia `dh_*`
- È richiamabile da `debian/rules`
- È configurabile sfruttando i parametri del comando o i files contenuti in `debian/` (e.g., *`package`*`.install`)

Come si potrà sfruttare tutto questo?

`debian/compat` dovrà essere impostato per utilizzare una versione di compatibilità ≥ 7

Helpers per la pacchettizzazione – *debhelper*

. . . perché non usarli sarebbe da pazzi!

- La miglior pratica è l'uso di un *aiutante* di pacchettizzazione
- In assoluto il più noto è debhelper, usato per il 98% dei pacchetti presenti negli archivi Debian
- Suoi obiettivi:
  1. "travasare" le attività comuni della pacchettizzazione in strumenti standard utilizzabili da tutti i pacchetti
  2. correggere i difetti di pacchettizzazione per tutti i pacchetti in un colpo solo, grazie alla famiglia `dh_*`
- È richiamabile da `debian/rules`
- È configurabile sfruttando i parametri del comando o i files contenuti in `debian/` (e.g., *`package`*`.install`)

Come si potrà sfruttare tutto questo?

`debian/compat` dovrà essere impostato per utilizzare una versione di compatibilità ≥ 7

Helpers per la pacchettizzazione – *debhelper*

. . . perché non usarli sarebbe da pazzi!

- La miglior pratica è l'uso di un *aiutante* di pacchettizzazione
- In assoluto il più noto è debhelper, usato per il 98% dei pacchetti presenti negli archivi Debian
- Suoi obiettivi:
  1. "travasare" le attività comuni della pacchettizzazione in strumenti standard utilizzabili da tutti i pacchetti
  2. correggere i difetti di pacchettizzazione per tutti i pacchetti in un colpo solo, grazie alla famiglia `dh_*`
- È richiamabile da `debian/rules`
- È configurabile sfruttando i parametri del comando o i files contenuti in `debian/` (e.g., *`package`*`.install`)

Come si potrà sfruttare tutto questo?

`debian/compat` dovrà essere impostato per utilizzare una versione di compatibilità ≥ 7

Helpers per la pacchettizzazione – *debhelper*

. . . perché non usarli sarebbe da pazzi!

- La miglior pratica è l'uso di un *aiutante* di pacchettizzazione
- In assoluto il più noto è debhelper, usato per il 98% dei pacchetti presenti negli archivi Debian
- Suoi obiettivi:
  1. "travasare" le attività comuni della pacchettizzazione in strumenti standard utilizzabili da tutti i pacchetti
  2. correggere i difetti di pacchettizzazione per tutti i pacchetti in un colpo solo, grazie alla famiglia `dh_*`
- È richiamabile da `debian/rules`
- È configurabile sfruttando i parametri del comando o i files contenuti in `debian/` (e.g., *`package`*`.install`)

Come si potrà sfruttare tutto questo?

`debian/compat` dovrà essere impostato per utilizzare una versione di compatibilità ≥ 7

Helpers per la pacchettizzazione – *debhelper*

. . . perché non usarli sarebbe da pazzi!

- La miglior pratica è l'uso di un *aiutante* di pacchettizzazione
- In assoluto il più noto è debhelper, usato per il 98% dei pacchetti presenti negli archivi Debian
- Suoi obiettivi:
  1. "travasare" le attività comuni della pacchettizzazione in strumenti standard utilizzabili da tutti i pacchetti
  2. correggere i difetti di pacchettizzazione per tutti i pacchetti in un colpo solo, grazie alla famiglia `dh_*`
- È richiamabile da `debian/rules`
- È configurabile sfruttando i parametri del comando o i files contenuti in `debian/` (e.g., *`package`*`.install`)

Come si potrà sfruttare tutto questo?

`debian/compat` dovrà essere impostato per utilizzare una versione di compatibilità ≥ 7

Helpers per la pacchettizzazione – *debhelper*

. . . perché non usarli sarebbe da pazzi!

- La miglior pratica è l'uso di un *aiutante* di pacchettizzazione
- In assoluto il più noto è debhelper, usato per il 98% dei pacchetti presenti negli archivi Debian
- Suoi obiettivi:
  1. "travasare" le attività comuni della pacchettizzazione in strumenti standard utilizzabili da tutti i pacchetti
  2. correggere i difetti di pacchettizzazione per tutti i pacchetti in un colpo solo, grazie alla famiglia `dh_*`
- È richiamabile da `debian/rules`
- È configurabile sfruttando i parametri del comando o i files contenuti in `debian/` (e.g., *`package`*`.install`)

Come si potrà sfruttare tutto questo?

`debian/compat` dovrà essere impostato per utilizzare una versione di compatibilità ≥ 7

CDBS e dh (aka debhelper 7 or dh7)

- *CDBS*:
  - introdotto nel 2005, basato sulla "magia avanzata" di GNU make
  - supporta per Perl, Ruby, GNOME, KDE, Java, ...
  - non sempre ben visto dagli sviluppatori
- dh:
  - introdotto nel 2008 come sostituto di *CDBS*
  - il comando **dh** richiama tutti gli `dh_*`
  - semplifica drasticamente `debian/rules`, dove ora si avrà solo la lista degli overrides sui defaults
  - più facile da personalizzare rispetto a *CDBS*

CDBS e dh (aka debhelper 7 or dh7)

- *CDBS*:
  - introdotto nel 2005, basato sulla “magia avanzata” di GNU make
  - supporta per Perl, Ruby, GNOME, KDE, Java, ...
  - non sempre ben visto dagli sviluppatori
- dh:
  - introdotto nel 2008 come sostituto di *CDBS*
  - il comando **dh** richiama tutti gli `dh_*`
  - semplifica drasticamente `debian/rules`, dove ora si avrà solo la lista degli overrides sui defaults
  - più facile da personalizzare rispetto a *CDBS*

CDBS e `dh` (aka debhelper 7 or `dh7`)

- *CDBS*:
  - introdotto nel 2005, basato sulla "magia avanzata" di GNU make
  - supporta per Perl, Ruby, GNOME, KDE, Java, ...
  - non sempre ben visto dagli sviluppatori
- `dh`:
  - introdotto nel 2008 come sostituto di *CDBS*
  - il comando **dh** richiama tutti gli `dh_*`
  - semplifica drasticamente `debian/rules`, dove ora si avrà solo la lista degli overrides sui defaults
  - più facile da personalizzare rispetto a *CDBS*

CDBS e dh (aka debhelper 7 or dh7)

- *CDBS*:
  - introdotto nel 2005, basato sulla “magia avanzata” di GNU make
  - supporta per Perl, Ruby, GNOME, KDE, Java, ...
  - non sempre ben visto dagli sviluppatori
- dh:
  - introdotto nel 2008 come sostituto di *CDBS*
  - il comando **dh** richiama tutti gli `dh_*`
  - semplifica drasticamente `debian/rules`, dove ora si avrà solo la lista degli overrides sui defaults
  - più facile da personalizzare rispetto a *CDBS*

CDBS e `dh` (aka debhelper 7 or `dh7`)

- *CDBS*:
  - introdotto nel 2005, basato sulla “magia avanzata” di GNU make
  - supporta per Perl, Ruby, GNOME, KDE, Java, ...
  - non sempre ben visto dagli sviluppatori
- `dh`:
  - introdotto nel 2008 come sostituto di *CDBS*
  - il comando **dh** richiama tutti gli `dh_*`
  - semplifica drasticamente `debian/rules`, dove ora si avrà solo la lista degli overrides sui defaults
  - più facile da personalizzare rispetto a *CDBS*

CDBS e `dh` (aka debhelper 7 or `dh7`)

- *CDBS*:
  - introdotto nel 2005, basato sulla "magia avanzata" di GNU make
  - supporta per Perl, Ruby, GNOME, KDE, Java, ...
  - non sempre ben visto dagli sviluppatori
- `dh`:
  - introdotto nel 2008 come sostituto di *CDBS*
  - il comando **dh** richiama tutti gli `dh_*`
  - semplifica drasticamente `debian/rules`, dove ora si avrà solo la lista degli overrides sui defaults
  - più facile da personalizzare rispetto a *CDBS*

CDBS e `dh` (aka debhelper 7 or `dh7`)

- *CDBS*:
  - introdotto nel 2005, basato sulla "magia avanzata" di GNU make
  - supporta per Perl, Ruby, GNOME, KDE, Java, ...
  - non sempre ben visto dagli sviluppatori
- `dh`:
  - introdotto nel 2008 come sostituto di *CDBS*
  - il comando **dh** richiama tutti gli `dh_*`
  - semplifica drasticamente `debian/rules`, dove ora si avrà solo la lista degli overrides sui defaults
  - più facile da personalizzare rispetto a *CDBS*

CDBS e `dh` (aka debhelper 7 or `dh7`)

- *CDBS*:
  - introdotto nel 2005, basato sulla "magia avanzata" di GNU make
  - supporta per Perl, Ruby, GNOME, KDE, Java, ...
  - non sempre ben visto dagli sviluppatori
- `dh`:
  - introdotto nel 2008 come sostituto di *CDBS*
  - il comando **dh** richiama tutti gli `dh_*`
  - semplifica drasticamente `debian/rules`, dove ora si avrà solo la lista degli overrides sui defaults
  - più facile da personalizzare rispetto a *CDBS*

CDBS e `dh` (aka debhelper 7 or `dh7`)

- *CDBS*:
  - introdotto nel 2005, basato sulla "magia avanzata" di GNU make
  - supporta per Perl, Ruby, GNOME, KDE, Java, ...
  - non sempre ben visto dagli sviluppatori
- `dh`:
  - introdotto nel 2008 come sostituto di *CDBS*
  - il comando **dh** richiama tutti gli `dh_*`
  - semplifica drasticamente `debian/rules`, dove ora si avrà solo la lista degli overrides sui defaults
  - più facile da personalizzare rispetto a *CDBS*

Generare i pacchetti

- `apt-get build-dep` *`package`*
  che installa le *dipendenze per la generazione*
- `debuild`
  che genera il pacchetto, lo testa con *lintian* e lo firma con *GnuPG*

Comunque è sempre meglio generare i pacchetti in ambienti minimali e "puliti" (cioè senza pacchetti inutili o che potrebbero falsare l'installazione).
Tra questi citiamo:

`pbuilder` helper per generare pacchetti in un chroot (assieme al fratello "wrapper" `cowbuilder`)

`sbuild` usato sui builders ufficiali Debian per la generazione automatica dei pacchetti sulle varie architetture

Generare i pacchetti

- `apt-get build-dep` *`package`*
  che installa le *dipendenze per la generazione*
- `debuild`
  che genera il pacchetto, lo testa con *lintian* e lo firma con *GnuPG*

Comunque è sempre meglio generare i pacchetti in ambienti minimali e "puliti" (cioè senza pacchetti inutili o che potrebbero falsare l'installazione).
Tra questi citiamo:

- `pbuilder` helper per generare pacchetti in un chroot (assieme al fratello "wrapper" `cowbuilder`)
- `sbuild` usato sui builders ufficiali Debian per la generazione automatica dei pacchetti sulle varie architetture

Generare i pacchetti

- `apt-get build-dep` *`package`*
  che installa le *dipendenze per la generazione*
- `debuild`
  che genera il pacchetto, lo testa con *lintian* e lo firma con *GnuPG*

Comunque è sempre meglio generare i pacchetti in ambienti minimali e "puliti" (cioè senza pacchetti inutili o che potrebbero falsare l'installazione).
Tra questi citiamo:

- `pbuilder` helper per generare pacchetti in un chroot (assieme al fratello "wrapper" `cowbuilder`)
- `sbuild` usato sui builders ufficiali Debian per la generazione automatica dei pacchetti sulle varie architetture

Generare i pacchetti

- `apt-get build-dep` *`package`*
  che installa le *dipendenze per la generazione*
- `debuild`
  che genera il pacchetto, lo testa con *lintian* e lo firma con *GnuPG*

Comunque è sempre meglio generare i pacchetti in ambienti minimali e "puliti" (cioè senza pacchetti inutili o che potrebbero falsare l'installazione).
Tra questi citiamo:

`pbuilder` helper per generare pacchetti in un chroot (assieme al fratello "wrapper" `cowbuilder`)

`sbuild` usato sui builders ufficiali Debian per la generazione automatica dei pacchetti sulle varie architetture

Generare i pacchetti

- `apt-get build-dep` *`package`*
  che installa le *dipendenze per la generazione*
- `debuild`
  che genera il pacchetto, lo testa con *lintian* e lo firma con *GnuPG*

Comunque è sempre meglio generare i pacchetti in ambienti minimali e "puliti" (cioè senza pacchetti inutili o che potrebbero falsare l'installazione).
Tra questi citiamo:

`pbuilder` helper per generare pacchetti in un chroot (assieme al fratello "wrapper" `cowbuilder`)

`sbuild` usato sui builders ufficiali Debian per la generazione automatica dei pacchetti sulle varie architetture

Installare e testare i pacchetti

- Installare il pacchetto localmente:
  `debi`, che userà il file `.changes` per sapere cosa installare
- Fare una lista dei contenuti del pacchetto:
  `debc ../package.changes`
- Confrontare il pacchetto con una versione precedente:
  `debdiff ../package_1_*.changes ../package_2_*.changes`
- Controllare il pacchetto con `lintian` (analizzatore statico):
  `lintian ../package.changes`
  che con vari parametri (`-iI`) fornisce maggiori informazioni

Quando il pacchetto è pronto, si potrà usare `dput` per uploadarlo sugli archivi. . . qualora se ne abbiano le credenziali ;-)

Installare e testare i pacchetti

- Installare il pacchetto localmente:
  `debi`, che userà il file `.changes` per sapere cosa installare
- Fare una lista dei contenuti del pacchetto:
  `debc ../package.changes`
- Confrontare il pacchetto con una versione precedente:
  `debdiff ../package_1_*.changes ../package_2_*.changes`
- Controllare il pacchetto con `lintian` (analizzatore statico):
  `lintian ../package.changes`
  che con vari parametri (`-iI`) fornisce maggiori informazioni

Quando il pacchetto è pronto, si potrà usare `dput` per uploadarlo sugli archivi. . . qualora se ne abbiano le credenziali ;-)

Installare e testare i pacchetti

- Installare il pacchetto localmente:
  `debi`, che userà il file `.changes` per sapere cosa installare
- Fare una lista dei contenuti del pacchetto:
  `debc ../package.changes`
- Confrontare il pacchetto con una versione precedente:
  `debdiff ../package_1_*.changes ../package_2_*.changes`
- Controllare il pacchetto con `lintian` (analizzatore statico):
  `lintian ../package.changes`
  che con vari parametri (`-iI`) fornisce maggiori informazioni

Quando il pacchetto è pronto, si potrà usare `dput` per uploadarlo sugli archivi. . . qualora se ne abbiano le credenziali ;-)

Installare e testare i pacchetti

- Installare il pacchetto localmente:
  `debi`, che userà il file `.changes` per sapere cosa installare
- Fare una lista dei contenuti del pacchetto:
  `debc ../package.changes`
- Confrontare il pacchetto con una versione precedente:
  `debdiff ../package_1_*.changes ../package_2_*.changes`
- Controllare il pacchetto con `lintian` (analizzatore statico):
  `lintian ../package.changes`
  che con vari parametri (`-iI`) fornisce maggiori informazioni

Quando il pacchetto è pronto, si potrà usare `dput` per uploadarlo sugli archivi. . . qualora se ne abbiano le credenziali ;-)

Installare e testare i pacchetti

- Installare il pacchetto localmente:
  `debi`, che userà il file `.changes` per sapere cosa installare
- Fare una lista dei contenuti del pacchetto:
  `debc ../package.changes`
- Confrontare il pacchetto con una versione precedente:
  `debdiff ../package_1_*.changes ../package_2_*.changes`
- Controllare il pacchetto con `lintian` (analizzatore statico):
  `lintian ../package.changes`
  che con vari parametri (`-iI`) fornisce maggiori informazioni

Quando il pacchetto è pronto, si potrà usare `dput` per uploadarlo sugli archivi. . . qualora se ne abbiano le credenziali ;-)

Installare e testare i pacchetti

- Installare il pacchetto localmente:
  `debi`, che userà il file `.changes` per sapere cosa installare
- Fare una lista dei contenuti del pacchetto:
  `debc ../package.changes`
- Confrontare il pacchetto con una versione precedente:
  `debdiff ../package_1_*.changes ../package_2_*.changes`
- Controllare il pacchetto con `lintian` (analizzatore statico):
  `lintian ../package.changes`
  che con vari parametri (`-iI`) fornisce maggiori informazioni

Quando il pacchetto è pronto, si potrà usare `dput` per uploadarlo sugli archivi. . . qualora se ne abbiano le credenziali ;-)

Pacchettizzare con i VCS (bzr, git, hg, svn, . . .)

Sono disponibili molti strumenti per gestire *branches* e *tags* del vostro VCS preferito. Ad esempio: `git-buildpackage`, `svn-buildpackage`.

Nel caso di `git-buildpackage`:

- `upstream` è il branch per tracciare il sorgente upstream
- `master` è il branch che traccia la pacchettizzazione Debian
- `pristine-tar` è il branch che permette di ricostruire il tarball di upstream
- `debian/version` tagga ogni versione uploadata

Pacchettizzare con i VCS (bzr, git, hg, svn, . . .)

Sono disponibili molti strumenti per gestire *branches* e *tags* del vostro VCS preferito. Ad esempio: `git-buildpackage`, `svn-buildpackage`.

Nel caso di `git-buildpackage`:

- `upstream` è il branch per tracciare il sorgente upstream
- `master` è il branch che traccia la pacchettizzazione Debian
- `pristine-tar` è il branch che permette di ricostruire il tarball di upstream
- `debian/version` tagga ogni versione uploadata

Pacchettizzare con i VCS (bzr, git, hg, svn, . . .)

Sono disponibili molti strumenti per gestire *branches* e *tags* del vostro VCS preferito. Ad esempio: `git-buildpackage`, `svn-buildpackage`.

Nel caso di `git-buildpackage`:

- `upstream` è il branch per tracciare il sorgente upstream
- `master` è il branch che traccia la pacchettizzazione Debian
- `pristine-tar` è il branch che permette di ricostruire il tarball di upstream
- `debian/version` tagga ogni versione uploadata

Pacchettizzare con i VCS (bzr, git, hg, svn, . . .)

Sono disponibili molti strumenti per gestire *branches* e *tags* del vostro VCS preferito. Ad esempio: `git-buildpackage`, `svn-buildpackage`.

Nel caso di `git-buildpackage`:

- `upstream` è il branch per tracciare il sorgente upstream
- `master` è il branch che traccia la pacchettizzazione Debian
- `pristine-tar` è il branch che permette di ricostruire il tarball di upstream
- `debian/version` tagga ogni versione uploadata

Pacchettizzare con i VCS (bzr, git, hg, svn, . . .)

Sono disponibili molti strumenti per gestire *branches* e *tags* del vostro VCS preferito. Ad esempio: `git-buildpackage`, `svn-buildpackage`.

Nel caso di `git-buildpackage`:

- `upstream` è il branch per tracciare il sorgente upstream
- `master` è il branch che traccia la pacchettizzazione Debian
- `pristine-tar` è il branch che permette di ricostruire il tarball di upstream
- `debian/version` tagga ogni versione uploadata

Dove è possibile trovare aiuto?

Si potrà aver bisogno di aiuto:

- per ottenere risposte alle proprie domande o per farsi revisionare il codice del pacchetto
- per la richiesta di *sponsoring*, una volta che il pacchetto è pronto

Si potrà ottenere quell'aiuto:

- dagli altri membri di un team di pacchettizzazione, dato che:
  - loro conoscono specificatamente il pacchetto
  - si può diventare membro del team, se è il caso
- dal gruppo dei *Debian Mentors* qualora il proprio pacchetto non possa essere classificato specificatamente per un team

Dove è possibile trovare aiuto?

Si potrà aver bisogno di aiuto:

- per ottenere risposte alle proprie domande o per farsi revisionare il codice del pacchetto
- per la richiesta di *sponsoring*, una volta che il pacchetto è pronto

Si potrà ottenere quell'aiuto:

- dagli altri membri di un team di pacchettizzazione, dato che:
  - loro conoscono specificatamente il pacchetto
  - si può diventare membri del team, se è il caso
- dal gruppo dei *Debian Mentors* qualora il proprio pacchetto non possa essere classificato specificatamente per un team

Dove è possibile trovare aiuto?

Si potrà aver bisogno di aiuto:

- per ottenere risposte alle proprie domande o per farsi revisionare il codice del pacchetto
- per la richiesta di *sponsoring*, una volta che il pacchetto è pronto

Si potrà ottenere quell'aiuto:

- dagli altri membri di un team di pacchettizzazione, dato che:
  - loro conoscono specificatamente il pacchetto
  - si può diventare membri del team, se è il caso
- dal gruppo dei *Debian Mentors* qualora il proprio pacchetto non possa essere classificato specificatamente per un team

Dove è possibile trovare aiuto?

Si potrà aver bisogno di aiuto:

- per ottenere risposte alle proprie domande o per farsi revisionare il codice del pacchetto
- per la richiesta di *sponsoring*, una volta che il pacchetto è pronto

Si potrà ottenere quell'aiuto:

- dagli altri membri di un team di pacchettizzazione, dato che:
  - loro conoscono specificatamente il pacchetto
  - si può diventare membro del team, se è il caso
- dal gruppo dei *Debian Mentors* qualora il proprio pacchetto non possa essere classificato specificatamente per un team

Dove è possibile trovare aiuto?

Si potrà aver bisogno di aiuto:

- per ottenere risposte alle proprie domande o per farsi revisionare il codice del pacchetto
- per la richiesta di *sponsoring*, una volta che il pacchetto è pronto

Si potrà ottenere quell'aiuto:

- dagli altri membri di un team di pacchettizzazione, dato che:
  - loro conoscono specificatamente il pacchetto
  - si può diventare membro del team, se è il caso
- dal gruppo dei *Debian Mentors* qualora il proprio pacchetto non possa essere classificato specificatamente per un team

Dove è possibile trovare aiuto?

Si potrà aver bisogno di aiuto:

- per ottenere risposte alle proprie domande o per farsi revisionare il codice del pacchetto
- per la richiesta di *sponsoring*, una volta che il pacchetto è pronto

Si potrà ottenere quell'aiuto:

- dagli altri membri di un team di pacchettizzazione, dato che:
  - loro conoscono specificatamente il pacchetto
  - si può diventare membro del team, se è il caso
- dal gruppo dei *Debian Mentors* qualora il proprio pacchetto non possa essere classificato specificatamente per un team

Dove è possibile trovare aiuto?

Si potrà aver bisogno di aiuto:

- per ottenere risposte alle proprie domande o per farsi revisionare il codice del pacchetto
- per la richiesta di *sponsoring*, una volta che il pacchetto è pronto

Si potrà ottenere quell'aiuto:

- dagli altri membri di un team di pacchettizzazione, dato che:
  - loro conoscono specificatamente il pacchetto
  - si può diventare membro del team, se è il caso
- dal gruppo dei *Debian Mentors* qualora il proprio pacchetto non possa essere classificato specificatamente per un team

Sommario

Perché pacchettizzare? Il punto di vista di Debian

- Creare il migliore sistema operativo possibile!
- Rendere facile l'installazione del software, tramite:
  - pre-compilazione, configurazione ottimizzata, licenze a posto
  - dipendenze rispettate, aggiornamenti efficaci, supporto per la sicurezza
- Facilitare la disinstallazione dei programmi
- Avere la certezza che il software funziona e funziona in simbiosi con il resto

Perché pacchettizzare? Il punto di vista di Debian

- Creare il migliore sistema operativo possibile!
- Rendere facile l'installazione del software, tramite:
  - pre-compilazione, configurazione ottimizzata, licenze a posto
  - dipendenze rispettate, aggiornamenti efficaci, supporto per la sicurezza
- Facilitare la disinstallazione dei programmi
- Avere la certezza che il software funziona e funziona in simbiosi con il resto

Perché pacchettizzare? Il punto di vista di Debian

- Creare il migliore sistema operativo possibile!
- Rendere facile l'installazione del software, tramite:
  - pre-compilazione, configurazione ottimizzata, licenze a posto
  - dipendenze rispettate, aggiornamenti efficaci, supporto per la sicurezza
- Facilitare la disinstallazione dei programmi
- Avere la certezza che il software funziona e funziona in simbiosi con il resto

Perché pacchettizzare? Il punto di vista di Debian

- Creare il migliore sistema operativo possibile!
- Rendere facile l'installazione del software, tramite:
  - pre-compilazione, configurazione ottimizzata, licenze a posto
  - dipendenze rispettate, aggiornamenti efficaci, supporto per la sicurezza
- Facilitare la disinstallazione dei programmi
- Avere la certezza che il software funziona e funziona in simbiosi con il resto

Perché pacchettizzare? Il punto di vista di Debian

- Creare il migliore sistema operativo possibile!
- Rendere facile l'installazione del software, tramite:
  - pre-compilazione, configurazione ottimizzata, licenze a posto
  - dipendenze rispettate, aggiornamenti efficaci, supporto per la sicurezza
- Facilitare la disinstallazione dei programmi
- Avere la certezza che il software funziona e funziona in simbiosi con il resto

Perché pacchettizzare? Il punto di vista di Debian

- Creare il migliore sistema operativo possibile!
- Rendere facile l'installazione del software, tramite:
  - pre-compilazione, configurazione ottimizzata, licenze a posto
  - dipendenze rispettate, aggiornamenti efficaci, supporto per la sicurezza
- Facilitare la disinstallazione dei programmi
- Avere la certezza che il software funziona e funziona in simbiosi con il resto

Non fermiamoci ai singoli pacchetti!

Quello di cui abbiamo veramente bisogno è:

- qualità
- integrazione
- standards, che sono normati da:
  - *Debian Policy Manual*
  - *Filesystem Hierarchy Standard* (aka FHS)
  - *Linux Standard Base* (aka LSB)

Non fermiamoci ai singoli pacchetti!

Quello di cui abbiamo veramente bisogno è:

- qualità
- integrazione
- standards, che sono normati da:
  - *Debian Policy Manual*
  - *Filesystem Hierarchy Standard* (aka FHS)
  - *Linux Standard Base* (aka LSB)

Non fermiamoci ai singoli pacchetti!

Quello di cui abbiamo veramente bisogno è:

- qualità
- integrazione
- standards, che sono normati da:
  - *Debian Policy Manual*
  - *Filesystem Hierarchy Standard* (aka FHS)
  - *Linux Standard Base* (aka LSB)

Non fermiamoci ai singoli pacchetti!

Quello di cui abbiamo veramente bisogno è:

- qualità
- integrazione
- standards, che sono normati da:
  - *Debian Policy Manual*
  - *Filesystem Hierarchy Standard* (aka FHS)
  - *Linux Standard Base* (aka LSB)

Non fermiamoci ai singoli pacchetti!

Quello di cui abbiamo veramente bisogno è:

- qualità
- integrazione
- standards, che sono normati da:
  - *Debian Policy Manual*
  - *Filesystem Hierarchy Standard* (aka FHS)
  - *Linux Standard Base* (aka LSB)

Non fermiamoci ai singoli pacchetti!

Quello di cui abbiamo veramente bisogno è:

- qualità
- integrazione
- standards, che sono normati da:
  - *Debian Policy Manual*
  - *Filesystem Hierarchy Standard* (aka FHS)
  - *Linux Standard Base* (aka LSB)

Sommario

Documentazione ufficiale

. . . che proprio si deve leggere!

- Debian Developers' Corner
  `http://www.debian.org/devel`
  che linka ad una miriade di risorse per lo sviluppo in Debian
- Debian New Maintainers' Guide
  `http://www.debian.org/doc/maint-guide`
  che è LA guida introduttiva alla pacchettizzazione in Debian
- Debian Developer's Reference
  `http://www.debian.org/doc/developers-reference`
  che comprende la maggior parte delle procedure di sviluppo e gestione di Debian
- Debian Policy Manual
  `http://www.debian.org/doc/debian-policy/`
  che specifica tutti i requisiti che ogni pacchetto deve soddisfare

Documentazione ufficiale

. . . che proprio si deve leggere!

- Debian Developers' Corner
  `http://www.debian.org/devel`
  che linka ad una miriade di risorse per lo sviluppo in Debian
- Debian New Maintainers' Guide
  `http://www.debian.org/doc/maint-guide`
  che è LA guida introduttiva alla pacchettizzazione in Debian
- Debian Developer's Reference
  `http://www.debian.org/doc/developers-reference`
  che comprende la maggior parte delle procedure di sviluppo e gestione di Debian
- Debian Policy Manual
  `http://www.debian.org/doc/debian-policy`
  che specifica tutti i requisiti che ogni pacchetto deve soddisfare

Documentazione ufficiale

. . . che proprio si deve leggere!

- Debian Developers' Corner
  `http://www.debian.org/devel`
  che linka ad una miriade di risorse per lo sviluppo in Debian
- Debian New Maintainers' Guide
  `http://www.debian.org/doc/maint-guide`
  che è LA guida introduttiva alla pacchettizzazione in Debian
- Debian Developer's Reference
  `http://www.debian.org/doc/developers-reference`
  che comprende la maggior parte delle procedure di sviluppo e gestione di Debian
- Debian Policy Manual
  `http://www.debian.org/doc/debian-policy`
  che specifica tutti i requisiti che ogni pacchetto deve soddisfare

Documentazione ufficiale

. . . che proprio si deve leggere!

- Debian Developers' Corner
  `http://www.debian.org/devel`
  che linka ad una miriade di risorse per lo sviluppo in Debian
- Debian New Maintainers' Guide
  `http://www.debian.org/doc/maint-guide`
  che è LA guida introduttiva alla pacchettizzazione in Debian
- Debian Developer's Reference
  `http://www.debian.org/doc/developers-reference`
  che comprende la maggior parte delle procedure di sviluppo e gestione di Debian
- Debian Policy Manual
  `http://www.debian.org/doc/debian-policy`
  che specifica tutti i requisiti che ogni pacchetto deve soddisfare

Grazie!

per l'attenzione, spero vi siate divertiti!

Domande?

Matteo F. Vescovi
mfv@fsugpadova.org