Da Zero a RPM

Introduzione al packaging

Gianluca Sforna
Fedora Project Ambassador

Prerequisiti

- saper usare un editor di testo
- saper installare RPM
- saper compilare un software dai sorgenti

fedora

Nota bene

- useremo un esempio semplice
- i pacchetti 'veri' sono più complicati
- a volte MOLTO più complicati

In caso di dubbi, chiedete!

Perchè pacchettizziamo

- standardizza deployment
  - sicuro di averlo installato
- semplifica l'ambiente
  - sicuro di come trovarlo
- gestione conformità
  - sicuro di cosa è presente

fedora

Come pacchettizziamo

https://fedoraproject.org/wiki/Packaging:Guidelines

- documento in evoluzione
- colleziona le “best practices”
- esistono eccezioni
  - quelle comuni sono documentate
  - per il resto rivolgersi al Fedora Engineering Steering Committee (FESCo)

fedora

Ok, ma quando si inizia?

ci vuole un progetto:

- semplice
- significativo
- che ci eviti la maledizione del programmatore

???

La maledizione

Quando si affronta un nuovo linguaggio o tecnologia, il primo progetto deve essere 'Hello world' o non saremo mai in grado di usare questo linguaggio o tecnologia

Hello RPM

Webpage Screenshot

http://www.gnu.org/software/hello/

fedora

Cuciniamo un RPM

Preparare un RPM è come cucinare

- Strumenti da cucina (rpmbuild, mock)
- Ingredienti (sorgenti + patch)
- Ricetta (spec file)

fedora

Gli strumenti

- installiamo l'indispensabile con

```
# yum install @fedora-packager
```

- creiamo ambiente di build nella home

```
$ rpmdev-setuptree
```


fedora

Lo spec file

- il file spec è come una ricetta
- simile ad uno script shell
- elenca i contenuti dell'RPM
  - sorgenti
  - patch
  - altri file
- descrive il processo di build

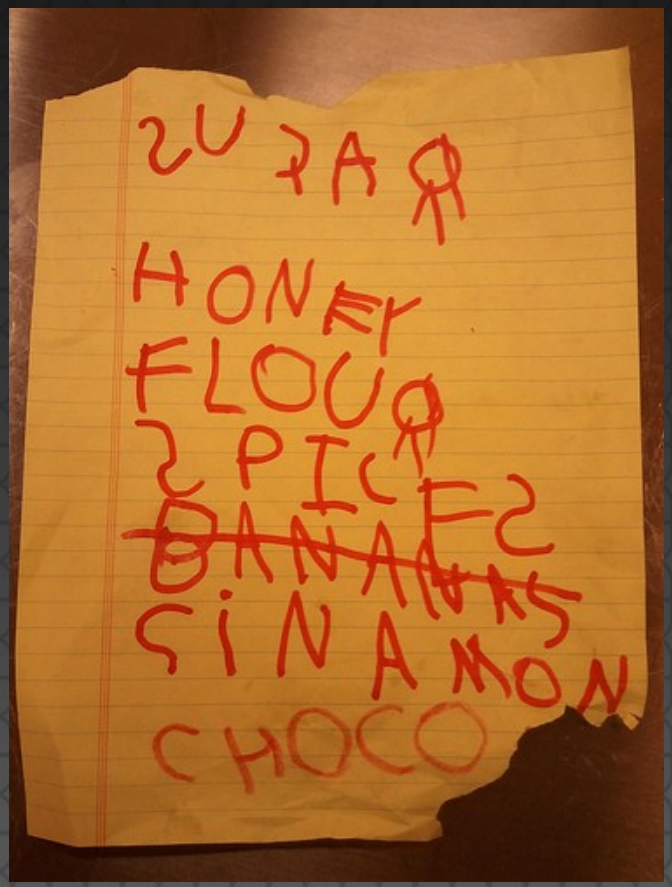


Image CC-BY-SA
http://www.flickr.com/photos/ted_major/5045483028

fedora

Anatomia di uno spec

- preambolo
- %prep (setup)
- %build
- %install
- %files
- %changelog

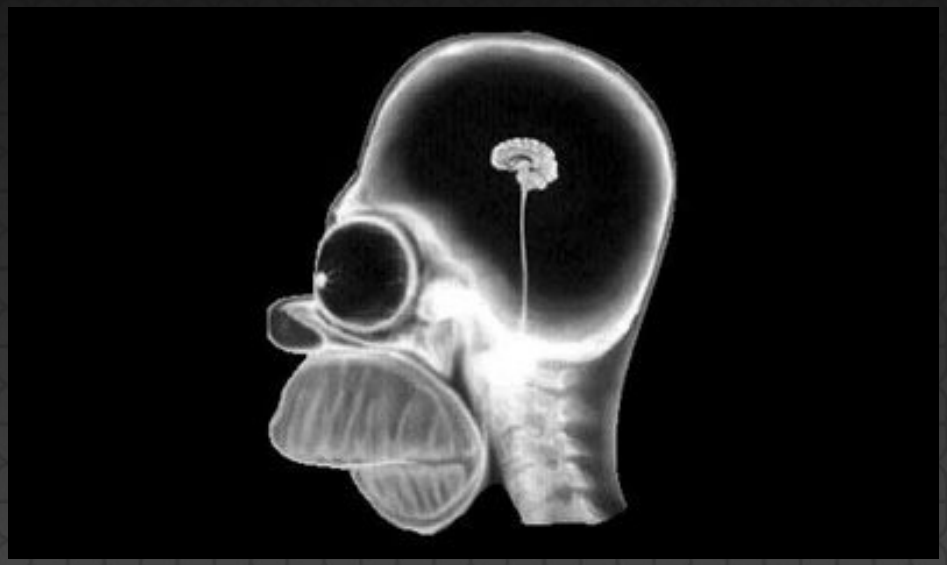


Image CC-BY
http://www.flickr.com/photos/27620885@N02/2671077524/

fedora

Il nostro primo spec

- Portiamoci nella directory SPECS

```
$ cd rpmbuild/SPECS/
```

- Prepariamo un nuovo spec

```
$ rpmdev-newspec hello
hello.spec created; type minimal, rpm version >=
4.9.
```

Preambolo

```
Name:      hello
Version:
Release:   1%{?dist}
Summary:
URL:
License:
Source0:

%description
```

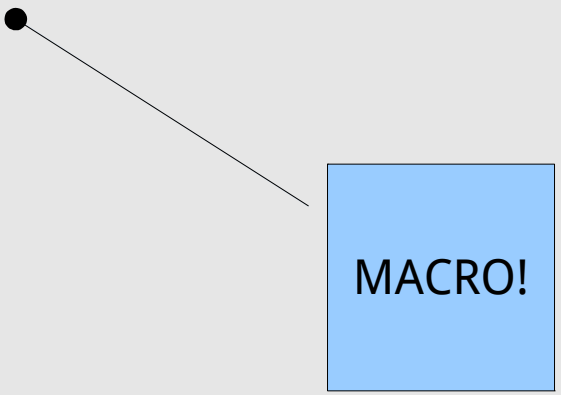
Usare spazi o tabulatori, a scelta.
Non mescolare lo stile nello stesso spec.

fedora

Preambolo

```
Name:      hello
Version:   2.8
Release:   1%{?dist}
Summary:
URL:
License:
Source0:

%description
```



- Nel caso più semplice, la versione è quella del pacchetto upstream
- in caso di pacchetti pre o post release, regole specifiche per campi Version and Release
  https://fedoraproject.org/wiki/Packaging:NamingGuidelines#Package_Version

fedora

Macro

- Simili alle variabili negli script shell
  - possono comportarsi da stringhe o interi ma è più semplice trattarle sempre da stringhe
- le più comuni macro sono già definite
  - rpm --showrc mostra tutte le macro definite
  - rpm --eval %{macroname} mostra l'espansione della macro
  - quasi tutte le macro di sistema inziano con _ (es. %{_bindir})

Macro %{?dist}

https://fedoraproject.org/wiki/Packaging:DistTag

- esiste in Fedora e RHEL 5+.
- serve ad aggiungere un identificativo (dist tag) alla fine del campo release
- ? significa “se definito, aggiungilo, altrimenti non fare nulla”
- verrà quindi convertito in “1.fc17” in Fedora 17, e “1.el5” in RHEL 5.

Preambolo

```
Name:     hello
Version:  2.8
Release:  1%{?dist}
Summary:  Prints a Familiar, Friendly Greeting
URL:
License:
Source0:

%description
Hello prints a friendly greeting. It also serves as a sample GNU
package, showing practices that may be useful for GNU projects.
```

- Summary è una breve descrizione (<80 caratteri) di quello che fa il pacchetto, senza punto alla fine
- La descrizione più estesa va in %description

Preambolo

```
Name:     hello
Version:  2.8
Release:  1%{?dist}
Summary:  Prints a Familiar, Friendly Greeting
URL:      http://www.gnu.org/software/hello/
License:
Source0:

%description
Hello prints a friendly greeting. It also serves as a sample GNU
package, showing practices that may be useful for GNU projects.
```

- URL fa riferimento alla home page del software

Preambolo

```
Name:     hello
Version:  2.8
Release:  1%{?dist}
Summary:  Prints a Familiar, Friendly Greeting
URL:      http://www.gnu.org/software/hello/
License:  TODO
Source0:

%description
Hello prints a friendly greeting. It also serves as a sample GNU
package, showing practices that may be useful for GNU projects.
```

- License identifica la licenza/e dei files installati dal pacchetto.
- Per determinare il valore corretto dovremo guardare i sorgenti. Per ora mettiamo TODO

fedora

Preambolo

```
Name:     hello
Version:  2.8
Release:  1%{?dist}
Summary:  Prints a Familiar, Friendly Greeting
URL:      http://www.gnu.org/software/hello/
License:  TODO
Source0:  ftp://ftp.gnu.org/gnu/hello/hello-2.8.tar.gz

%description
Hello prints a friendly greeting. It also serves as a sample GNU
package, showing practices that may be useful for GNU projects.
```

- Source0 indica quali sorgenti usare.
- Si può avere più di un Source# se necessario
- Usare la URL completa da cui si è scaricato il file

Preambolo

```
Name:     hello
Version:  2.8
Release:  1%{?dist}
Summary:  Prints a Familiar, Friendly Greeting
URL:      http://www.gnu.org/software/hello/
License:  TODO
Source0:  ftp://ftp.gnu.org/gnu/hello/%{name}-%{version}.tar.gz

%description
Hello prints a friendly greeting. It also serves as a sample GNU
package, showing practices that may be useful for GNU projects.
```

- Ancora macro!
- in caso di update, basterà cambiare Version

spec file: prep

```
%prep
%setup -q
%patch0 -p1
```

- Scompatta sorgenti in ~/rpmbuild/BUILD/
- Applica eventuali patch
- Altri comandi pre-build

spec file: build

```
%build
%configure
make %{?_smp_mflags}
```

- creazione componenti binarie
- macro %configure
  - utile per progetti basati su autotools
  - esegue ./configure con sane opzioni di default
- da adattare a seconda del metodo di build usato (scons, cmake, etc) dal progetto

spec file: install

```
%install
rm -rf %{buildroot}
make install DESTDIR=%{buildroot}
```

- creazione della buildroot
- preparazione struttura del filesystem
- copia dei file compilati nella buildroot
- eventuale pulizia file installati non necessari

fedora

spec file: files

- elenca il contenuto del pacchetto
  - se non appare in %files, non ci sarà nel pacchetto
  - RPM si fermerà in presenza di file non pacchettizzati
    - MAI cercare di aggirare il controllo
- Per ora lasciamolo vuoto

fedora

spec file: changelog

```
%changelog
* Sat Oct 27 2012 Gianluca Sforna <giallu@gmail.com> 2.8-1
- Initial package
```

- usato per annotare i cambiamenti al pacchetto
- non è alternativo al changelog dei sorgenti
- da aggiornare ad OGNI cambiamento

fedora

Licenza

- rpmbuild -bp hello.spec
  -b: build, -p: prep (esegue solo sezione prep)
- se non ci sono errori troverete i sorgenti in ~/rpmbuild/BUILD/hello-2.8
- assegnare la giusta licenza è fondamentale.
- alcuni suggerimenti:
  - cercate nei file COPYING, LICENSE o README
  - controllate i sorgenti, il codice migliore ha la licenza indicata in ognuno dei file

Licenza /2

- annotare le licenze trovate (anche più di una)
- confrontarle con la lista ufficiale sul wiki
  https://fedoraproject.org/wiki/Licensing#Software_License_List
- che licenza mettiamo per hello?
- trovare la giusta licenza può essere molto complicato
- per ogni dubbio scrivere alla mailing list:
  fedora-legal@lists.fedoraproject.org

fedora

%doc

- COPYING va messo nell'RPM
- aggiungiamolo a %files con la macro %doc
- %doc marca i files come documentazione
- li copia direttamente dai sorgenti:
  ~/rpmbuild/BUILD/hello-2.8/
  nella “docdir” del pacchetto
- mettiamolo nello spec:

```
%files
%doc COPYING
```

Dai sorgenti al binario

- rpmbuild -ba hello.spec
  -b: build, -a: all packages (src e binari)
- rpmbuild -bb hello.spec
  -b: build, -b: binary packages (solo binari)
- rpmbuild -bc hello.spec
  -b: build, -c: compile (esegue solo sezione prep)
- man rpmbuild

Proviamo la build

- rpmbuild -ba hello.spec
- ERRORE. Ma ci dice quali files aggiungere!

```
RPM build errors:
    Installed (but unpackaged) file(s) found:
   /usr/bin/hello
   /usr/share/info/dir
   /usr/share/info/hello.info.gz
   /usr/share/locale/bg/LC_MESSAGES/hello.mo
   /usr/share/locale/ca/LC_MESSAGES/hello.mo
   /usr/share/locale/da/LC_MESSAGES/hello.mo
...
```

- che ci fanno i file sotto /usr/share/locale ?

Traduzioni

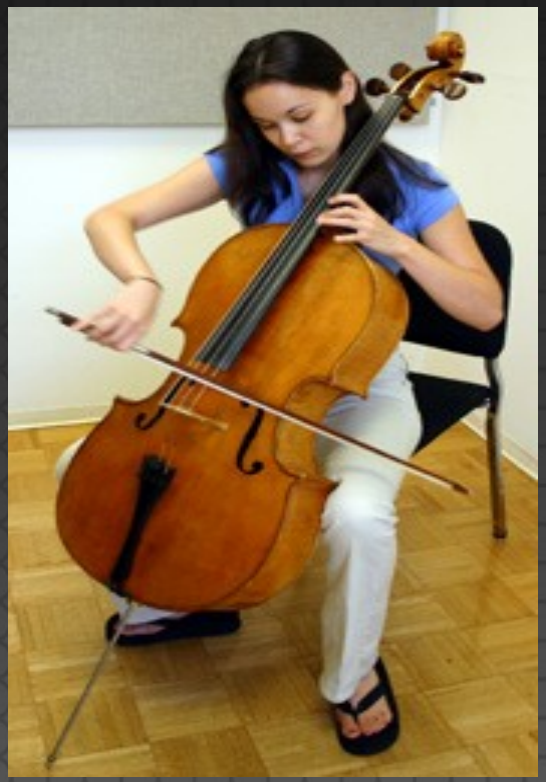
- potremmo aggiungere i files a mano
- supporto diretto nel build system

  Tre step:

  - aggiungo nel preambolo:
    BuildRequires: gettext
  - aggiungo in %install:
    %{find_lang} %{name}
  - modifico %files:
    %files -f %{name}.lang

Best Practices

- K.I.S.S.
- una patch per ogni modifica
- evitare pre/post quando possibile
- usare sempre il changelog
- ispirarsi a pacchetti Fedora
- usare macro (correttamente)
  - essere coerenti
  - preferire macro di sistema



fedora

Altre Best Practices

- usare rpmlint, correggere gli errori segnalati
- includere config/script come files (Source#)
- Commenti!
  - ...ma che lo spec rimanga leggibile
  - pensare a chi dovrà lavorare al pacchetto dopo di noi.
- MAI eseguire rpm da uno spec file
  - come in Ghostbusters.
    Incrociare i flussi? male.

fedora

Errori di inesperienza

- generatori di pacchetti
  - "funziona" non è lo stesso di "fatto bene"
- pacchettizzare binari, non partendo dai sorgenti
  - non sempre evitabile, ma potendo scegliere meglio non farlo
- disattivare controllo file non pacchettizzati
  - solo se si è in cerca di guai...

fedora

Link utili

Linee guida per il packaging:

- http://fedoraproject.org/wiki/Packaging:Guidelines
- http://fedoraproject.org/wiki/Packaging:ReviewGuidelines

Maximum RPM:

- http://rpm.org/max-rpm-snapshot/

Fedora GIT Tree (contiene migliaia di spec)

- http://pkgs.fedoraproject.org/cgit/

Rpmlint website:

- http://rpmlint.zarb.org

GRAZIE

Contatti:

http://morefedora.blogspot.com
http://giallu.fedorapeople.org/events

giallu@fedoraproject.org
@giallu su identi.ca e twitter

Preamble: Other items we didn't need for enum

- Patch0 – If you need to apply a patch to the software being packaged, you can add a numbered patch entry here:

```
Patch0:     foo-1.2.3-fixbugs.patch
```

- BuildRequires – This lists the packages which need to be present to build the software. rubygem-color is very very simple. Most packages have at least one BuildRequires:

```
BuildRequires:  bar >= 2.0
```

  You can list as many packages as BuildRequires as you need, although, you should try to avoid redundant items. Also, BuildRequires can be versioned (see above).

Preamble: Explicit Requires

- Requires – This lists any packages which we know are necessary to be present on the system to run the software in our package, once it is installed. RPM usually does a very good job of autodetecting dependencies and adding them for you especially when the software is in C, C++, or Perl.
  - Be careful about adding explicit Requires here, as most packages will not need any.
  - You can add versioned Requires as needed, in exactly the same way as versioned BuildRequires are done.

%if condition

- How to handle different OSes with one spec? Use macros and conditions:
- %if 0%{?rhel} == 6 || 0%{?fedora} < 17
- Requires: ruby(abi) = 1.8
- %else
- Requires: ruby(abi) = 1.9.1
- %endif

Preamble: Explicit Provides

- Provides: tell rpm that this package provides something else.
  E.g. Httpd provides www-server