FUSS - Manuale per lo sviluppatore

Release 0.1

FUSS Lab

30 ago 2019

Indice

Il presente manuale è una guida alla manutenzione ed allo sviluppo della distribuzione FUSS GNU/Linux basata, lato server, su Debian 8 «Jessie» mentre lato client su Debian 9 «Stretch».

CAPITOLO 1

Guide del progetto FUSS

Le guide del progetto fuss (user, referent, tech e dev) sono scritte in reStructuredText e pubblicate usando il generatore di documentazione sphinx

1.1 Repository

I sorgenti delle guide sono in repository git hostati su gitlab; avendo un'account sulla piattaforma ed una chiave ssh configurata possono essere clonati con:

```
git clone git@gitlab.com:fusslab/fuss-user-guide.git
git clone git@gitlab.com:fusslab/fuss-referent-guide.git
git clone git@gitlab.com:fusslab/fuss-tech-guide.git
git clone git@gitlab.com:fusslab/fuss-dev-guide.git
```

altrimenti si può usare l'accesso https:

```
git clone https://gitlab.com/fusslab/fuss-user-guide.git
git clone https://gitlab.com/fusslab/fuss-referent-guide.git
git clone https://gitlab.com/fusslab/fuss-tech-guide.git
git clone https://gitlab.com/fusslab/fuss-dev-guide.git
```

1.2 Creazione di nuove pagine

Per aggiungere una pagina ad una guida:

- Creare un nuovo file con estensione `.rst`, ad esempio `titolo-dell-articolo.rst`; è opportuno che il nome del file contenga solo lettere minuscole, numeri e il trattino `-`, senza lettere accentate, spazi né altri caratteri speciali.
- Aggiungere l'articolo al *doctree* in `index.rst`, aggiungendo il nome del file senza estensione (es. `titolo-dell-articolo`) nella posizione opportuna rispetto agli altri articoli.

1.3 Build locale

Mentre si stanno facendo modifiche, può essere utile generare localmente le pagine html per avere un'anteprima di quanto sarà poi pubblicato.

1.3.1 Setup

Per la generazione è necessario installare le seguenti dipendenze (su FUSS client, o una versione recente di Debian o derivate):

```
apt install python3-sphinx python3-sphinx-rtd-theme python3-stemmer
```

1.3.2 Build

Per generare le pagine html:

```
cd <path/del/repository>/docs
make html
```

e si può poi vedere il risultato con:

```
sensible-browser _build/html/index.html
```

1.4 Aggiornamento versione pubblicata

La versione pubblicata su readthedocs viene aggiornata automaticamente ogni volta che viene aggiornato il branch `master` su gitlab; se si ha accesso in scrittura ai repository è sufficiente:

```
cd <path/del/repository>
git push origin master
```

(oppure semplicemente `git push` se ci si trova già sul branch `master`)

Se non si ha accesso in scrittura al repository si può invece creare una merge request

1.5 Linee di guida per lo stile

1.5.1 Screenshot ed esempi di terminale

Per guide che si riferiscono a programmi grafici, è opportuno aggiungere screenshot.

Per quanto riguarda esempi tratti dal terminale, meglio riportare comandi ed output come testo, per maggiore accessibilità, usando dei blocchi di codice, ad esempio:

```
testo introduttivo::

    $ comando
    output del comando
    # /sbin/comando
    output del comando lanciato da root
```

Nota: I doppi due punti del codice reStructuredText vengono convertiti in un punto unico nelle versioni pubblicate, a meno che non sia preceduto da spazi (o in un paragrafo da solo), nel qual caso viene rimosso.

1.5.2 Livelli di struttura

reStructuredText permette di usare caratteri diversi per i vari livelli di struttura di un documento, purché siano coerenti all'interno del singolo file.

È però meglio attenersi alla convenzione della Python's Style Guide for documenting, che prevede:

- # con doppia riga per le parti (non usate in questi manuali);
- * con doppia riga per i capitoli (corrisponenti per noi ai file);
- = per le sezioni;
- – per le sottosezioni;
- ^ per le sotto-sottosezioni;
- " per i paragrafi.

1.6 Tips e tricks

1.6.1 Migrazione dalla wiki di Redmine

I documenti presenti sulla wiki di redmine usano il formato (sorgente) textile; per convertirlo in reStructuredText può essere utile il programma Pandoc

Una volta copiato il sorgente della pagina in un file locale `articolo.txt` si può usare il seguente comando per ottenerne una versione in reStructuredText nel file `articolo.rst`:

```
pandoc -f textile -t rst -o articolo.rst articolo.txt
```

Tale file è un buon punto di partenza, ma non è pronto per l'inclusione diretta nelle guide FUSS senza prima verificare manualmente i contenuti; in particolare sarà sicuramente necessario sistemare manualmente alcune caratteristiche.

- I livelli di struttura andranno corretti per adeguarsi allo standard specificato qui sopra, facendo attenzione alla posizione in cui viene inserito il documento (ad esempio se si sta convertendo una pagina della wiki in una sezione o sottosezione di una pagina della guida).
- Le immagini vengono caricate dal file originale sulla wiki; per la mantenibilità futura è invece opportuno caricarle localmente all'interno della guida.

  Dopo aver scaricato le immagini, ad esempio nella directory `images/articolo/`, e spostati i riferimenti generati da pandoc dalla fine del documento alla posizione dove dovrà apparire l'immagine, li si può convertire in direttive `figure` col seguente comando di vim:

  ```
  :%s/|image.*| image:: https:\/\/work.fuss.bz.it\/attachments\/download\/.*\//
  ↪figure:: images\/articolo\//
  ```

- pandoc genera dei blocchi di codice introdotti da una riga contenente solo `::`, anche quando il paragrafo precedente termina con `:`; per migliore eleganza e leggibilità del sorgente questi si possono convertire mettendo `::` solo sul paragrafo precedente.

  Entrambi i casi sono comunque costruzioni reST valide che vengono compilate in una presentazione simile.

- Alcuni articoli della wiki contengono sezioni con indicazioni tipo "attenzione" o "nota": in questi casi può valere la pena convertirli nella relativa direttiva specifica reStructuredText

1.6.2 Screenshot

Redimensionare una finestra con precisione

Per fare screenshot della finestra di un programma è utile ridimensionarla alla dimensione precisa che si vuole dare allo screenshot; per farlo si può usare il comando seguente:

```
sleep 3 ; xdotool getactivewindow windowsize 1024 768
```

che aspetta 3 secondi, nel corso del quale si può cambiare la finestra attiva dal terminale alla finestra desiderata, e quindi effettua il resize.

CAPITOLO 2

Pacchetti e Repository

La distribuzione FUSS comprende un repository di pacchetti aggiuntivi rispetto alla base (Debian), disponibile all'indirizzo https://archive.fuss.bz.it/ ed ospitato su `isolda.fuss.bz.it` nella directory `/iso/repo`.

2.1 Build dei pacchetti

Nei repository del software sviluppato per FUSS è presente la directory `debian` contenente i file necessari per la generazione dei pacchetti `.deb`.

In alcuni progetti, tale directory è presente solo in un branch dedicato, con un nome tipo `fuss/<versione>`; per questi casi vedere anche la sezione *Pacchettizzazione gestita con git* .

2.1.1 Setup

Per effettuare build locali dei pacchetti è necessario installare alcuni strumenti di sviluppo:

```
# apt install devscripts dput-ng dh-systemd
```

Nota: Assicurarsi di aver installato anche i Recommends dei pacchetti (questo è il comportamento di default di `apt`, a meno che non lo si sia disabilitato manualmente), in particolare nel caso di `dput-ng`.

Inoltre è necessario impostare le variabili di ambiente `DEBEMAIL` e `DEBFULLNAME`, contenenti rispettivamente il nome completo e l'email dello sviluppatore, che verranno usate per aggiornare alcuni metadati.

Per usare `dput-ng` per effettuare gli upload serve configurarlo creando il file `~/.dput.d/profiles/fuss-<versione>.json` contenente:

```
{
    "method": "sftp",
    "fqdn": "archive.fuss.bz.it",
    "incoming": "/iso/incoming/<versione>",
    "allow_dcut": false,
    "allowed-distribution": {},
    "codenames": null,
    "post_upload_command": "ssh -S none isolda.fuss.bz.it 'sudo /iso/bin/post-
↪upload'",
```

(continues on next page)

(continua dalla pagina precedente)

```
    "hooks": [
        "allowed-distribution",
        "checksum",
        "suite-mismatch"
    ]
}
```

dove <versione> è jessie per FUSS server, stretch per FUSS client, e buster e buster-proposed-updates per la prossima versione di FUSS sia server che client.

Assicurarsi inoltre di poter accedere via ssh ad archive.fuss.bz.it senza ulteriori opzioni; ad esempio potrebbe essere necessario aggiungere quanto segue a ~/.ssh/config:

```
Host archive.fuss.bz.it
    User root
```

Nota: dput-ng usa paramiko per effettuare le connessioni ssh; questo implica che le opzioni impostate direttamente in ~/.ssh/config vengono lette correttamente, ma non c'è supporto per l'uso di Include per suddividere la configurazione su più file.

Inoltre non verrà salvato il fingerprint dei server, ma verrà chiesto ogni volta di verificarlo.

A marzo 2019i fingerprint di isolda sono:

```
256 SHA256:aLTgA+Trj5iYo0dl0i8Q82aigs3K/dPwDbazrvG95YY root@isolda (ECDSA)
256 SHA256:7i6j0jXPWRrW6LXDGbR+HWr3AFJi6gGSmdW41uBRJV4 root@isolda (ED25519)
2048 SHA256:OkP1maDf0pSIGCdq1mph8oI8CTADMrFXfe3aty608SA root@isolda (RSA)

256 MD5:b1:a1:ec:cb:a5:39:c8:8d:39:f1:dd:ba:aa:be:38:11 root@isolda (ECDSA)
256 MD5:21:41:8b:19:1b:25:b5:9c:f2:5c:e8:b9:8b:08:07:f8 root@isolda (ED25519)
2048 MD5:bd:88:bd:5f:bc:52:03:0b:88:d9:0c:2b:86:59:dc:92 root@isolda (RSA)
```

Cowbuilder

Nota: cowbuilder e pbuilder sono degli strumenti per gestire delle chroot all'interno delle quali effettuare build di pacchetti in un ambiente pulito e abbastanza isolato dal sistema base.

Buildare pacchetti all'interno di un sistema isolato è utile per evitare influenze da parte del proprio sistema (con librerie ed altre dipendenze già installate, magari in versioni non standard), ma è anche comodo nel caso si vogliano generare pacchetti per distribuzioni diverse da quelle in uso (ad esempio buildare per jessie o stretch su un sistema buster)

Oltre a quanto indicato sopra, installare cowbuilder e pbuilder:

```
# apt install pbuilder cowbuilder
```

ed assicurarsi che l'utente che si vuole usare per lanciare le build faccia parte del gruppo sudo.

Quindi creare le chroot base per le distribuzioni attualmente (giugno 2019) in uso buster, stretch e jessie:

```
# cowbuilder --create --distribution buster --debootstrap debootstrap \
  --basepath /var/cache/pbuilder/base-fuss-buster.cow
# cowbuilder --create --distribution stretch --debootstrap debootstrap \
  --basepath /var/cache/pbuilder/base-fuss-stretch.cow
# cowbuilder --create --distribution jessie --debootstrap debootstrap \
  --basepath /var/cache/pbuilder/base-fuss-jessie.cow
```

Suggerimento: cowbuilder può essere usato anche sotto distribuzioni derivate da debian, come ubuntu; in questo caso è necessario però specificare esplicitamente l'uso di un mirror debian aggiungendo ai comandi sopra le opzioni:

```
--mirror <un mirror debian valido> --components main
```

Aggiungere i repository di backports e fuss alle chroot base appena create: fare login nella chroot:

```
# cowbuilder --login --save-after-login \
  --basepath /var/cache/pbuilder/base-fuss-buster.cow
```

ed effettuare le modifiche a `/etc/apt/sources.list` e l'aggiunta della chiave (sostituendo `<mirror>` con un mirror debian opportuno, ad esempio quello già presente in `/etc/apt/sources.list`:

```
# echo 'deb <mirror> buster-backports main' >> /etc/apt/sources.list
# echo 'deb http://archive.fuss.bz.it/ buster main contrib' \
  >> /etc/apt/sources.list
# apt install gnupg
# apt-key add - # incollare i contenuti di
                # https://archive.fuss.bz.it/apt.key seguiti da ctrl-d
# apt remove gnupg
# apt autoremove
# apt update
# exit
```

Nota: nella chroot minimale da stretch in poi non è presente gnupg, che è necessario per l'uso di apt-key add: lo installiamo per l'operazione e rimuoviamo subito dopo per essere certi che l'immagine sia sempre minimale e continuare ad accorgersi di eventuali dipendenze non esplicitate nei pacchetti che generiamo.

Ripetere la stessa cosa per la chroot `jessie`.

Nota: dato che jessie-backports non è più supportato, quando si configura la chroot per jessie con `cowbuilder --login ...` è necessario configurarlo puntando agli archivi e disabilitare il controllo della data di validità della firma di tutti i repository.:

```
# echo 'deb http://archive.debian.org/debian jessie-backports main' >> /etc/apt/
↪sources.list
# echo 'Acquire::Check-Valid-Until no;' >> /etc/apt/apt.conf.d/99no-check-valid-
↪until
```

Purtroppo in jessie non era ancora disponibile la possibilità di disattivare tale controllo per una sola fonte.

Nel caso in cui le chroot siano state create da un po" di tempo è opportuno aggiornarle, coi seguenti comandi:

```
# cowbuilder --update --basepath /var/cache/pbuilder/base-fuss-buster.cow/
# cowbuilder --update --basepath /var/cache/pbuilder/base-fuss-stretch.cow/
# cowbuilder --update --basepath /var/cache/pbuilder/base-fuss-jessie.cow/
```

2.1.2 Clone del repository

Clonare il repository del progetto desiderato:

```
$ git clone https://work.fuss.bz.it/git/<progetto>
```

Nei vecchi progetti il branch da buildare per l'upload è `master`, in quelli recenti `fuss/master`, entrambi da aggiornare nel caso in cui si abbia già un clone locale del repository:

```
$ git checkout [fuss/]master
$ git pull
```

2.1.3 Versionamento

Per poter pubblicare il pacchetto, è necessario incrementare il numero di versione nel file `debian/changelog`.

Il numero di versione da dare dipende dal tipo di pacchetto, come descritto nella sezione *Policy di versionamento* e nelle guide di sviluppo degli specifici pacchetti, ma nella maggior parte dei casi sarà da incrementare il patch level (es. da 9.0.5-1 a 9.0.6-1).

Nota: Nei pacchetti contenenti programmi in python è generalmente necessario mantenere aggiornato il numero di versione anche in `setup.py`; come per debsrc sopra questo dovrebbe essere citato nel README dei pacchetti.

Il programma `dch`, permette di automatizzare l'editing del file `debian/changelog` che contiene la versione del pacchetto.

- Quando si iniziano a fare modifiche usare il comando `dch -v <nuova_versione>` per creare una nuova stanza ed aprire il changelog nell'editor di default.

  Verrà impostato il numero di versione richiesto e la release speciale `UNRELEASED` che indica che le modifiche sono ancora in lavorazione.

  Si può anche usare `dch` senza opzioni: in questo modo se l'ultima stanza risulta `UNRELEASED` il file verrà aperto così com'è, mentre se l'ultima stanza riporta una release come `unstable` ne viene creata una nuova incrementando il numero di versione.

  Attenzione che in quest'ultimo caso dch potrebbe non essere in grado di indovinare la versione corretta: verificare e nel caso correggere. Inoltre, nel caso in cui non si sia elencati tra i Maintainer e Uploaders in `debian/control` verrà aggiunta una riga `Non Maintainer Upload` che per noi non è rilevante e va tolta.

  Nel caso in cui più persone facciano modifiche, dch provvederà a suddividerle in sezioni intestate con il nome della persona che ha effettuato la modifica.

- Descrivere le modifiche effettuate, possibilmente indicando i ticket di riferimento da cui nascono le richieste di modifica.
- Man mano che si fanno modifiche, descriverle se necessario nel changelog, usando `dch` senza opzioni, come descritto sopra.
- Quando si è pronti a pubblicare il pacchetto, modificare `UNRELEASED` con `unstable` nella prima riga; questo si può fare anche con il comando `dch -r`.

2.1.4 Verifica dello stato del repository e push

Prima di effettuare la build, accertarsi di aver committato tutte le modifiche effettuate, di non avere file spuri e di essere sul branch corretto (`master` o `fuss/master` a seconda dell'età del progetto), ad esempio con il comando:

```
$ git status
```

Committare quindi eventuali modifiche rimanenti, indicando se possibile nel commit log il numero di ticket associato alla modifica, con la dicitura «refs #NUMERO»:

```
$ git add -p <file modificati>
$ git add <file aggiunti>
$ git commit -m "<modifiche effettuate>. refs #NumeroTicket"
```

Inoltre o subito prima o subito dopo la build, ma prima dell'upload, è importante pushare tali commit, in modo da essere sicuri che nel frattempo non avvengano conflitti con commit altrui:

```
$ git push
```

2.1.5 Build

Alcuni pacchetti, come octofussd necessitano della preventiva creazione del tar dei sorgenti originali, come specificato nel README dei rispettivi repository; in tal caso prima di eseguire il comando precedente è necessario eseguire, nella directory principale del repository:

```
$ debian/rules debsrc
```

A questo punto si può usare `pdebuild` per eseguire la build del pacchetto all'interno di una chroot opportuna gestita da cowbuilder (sostituendo `buster` con `stretch` o `jessie` nei casi opportuni):

```
$ DIST=buster pdebuild --use-pdebuild-internal --pbuilder cowbuilder -- --basepath␣
↪/var/cache/pbuilder/base-fuss-buster.cow/
```

`pdebuild` provvederà autonomamente ad installare le dipendenze necessarie all'interno della chroot e ad effettuare la build.

Generalmente, l'infrastruttura di build[1] è in grado di capire dal numero di versione ed altri indizi se sia necessario o meno includere la tarball `.orig` tra ciò che va uploadato. Nel caso in cui però si stia effettuando un backport questo non è automatico: per il primo backport di una certa versione upstream è necessario prevedere l'inclusione della tarball sorgente con l'opzione `--debbuildopts "-sa"`, ovvero:

```
$ DIST=buster pdebuild --buildresult ../build/ --use-pdebuild-internal --pbuilder␣
↪cowbuilder --debbuildopts "-sa" -- --basepath /var/cache/pbuilder/base-fuss-
↪buster.cow/
```

Nota: Alla fine della build si riceverà un warning `cannot create regular file /var/cache/pbuilder/result/<nomepacchetto>_<versione>.dsc`; questo è irrilevante e i file necessari che sono stati generati si trovano nella directory superiore a quella da cui è stato lanciato `pdebuild`.

2.1.6 Test

Tramite `dpkg -i <nomefile>.deb` si puo" installare e testare il pacchetto. Si ricorda che `dpkg` non risolve le dipendenze, quindi darà un errore nel caso si sia aggiunta una nuova dipendenza.

Per installare le dipendenze mancanti si può usare il comando:

```
# apt -f install
```

Un altro comando utile è `dpkg -c <nomefile>.deb` per verificare i file presenti nel pacchetto.

[1] In particolare, l'inclusione o meno della tarball sorgente è decisa ed effettuata da `dpkg-genchanges`, richiamato da `dpkg-buildpackage` al quale `pdebuild` passa le opzioni specificate con il parametro `--debbuildopts`.
Generalmente questo avviene in automatico, senza bisogno di preoccuparsi di chi faccia cosa.

lintian

Uno strumento di diagnostica molto dettagliato è `lintian`, che analizza i pacchetti generati alla ricerca di problemi di vario tipo e si lancia con:

```
lintian --pedantic -Iiv <pacchetto>.changes
```

Un limite di questo strumento è che è basato sugli standard di Debian e in alcuni casi gli errori potrebbero essere falsi positivi per gli standard fuss.

In particolare si possono ignorare i seguenti tag.

- `changelog-should-mention-nmu`
- `source-nmu-has-incorrect-version-number`

ed altri che verranno successivamente aggiunti a questo elenco.

2.1.7 Upload

Per uploadare il pacchetto buildato con `dput-ng` è sufficiente usare il comando:

```
$ dput fuss-<versione> nomepacchetto_versione_arch.changes
```

Nel caso si voglia procedere manualmente invece si possono copiare i file generati su `isolda` nella directory `/iso/incoming/<versione>` ed aggiornare il repository con il comando:

```
# /iso/bin/post-upload
```

Verificare poi che in `/iso/incoming/<versione>` non siano rimasti file spuri, e nel caso cancellarli a mano.

2.1.8 Tagging

Nel momento in cui tutto è pronto per un upload, taggare il commit corrispondente a quanto verrà uploadato con il comando:

```
$ git tag -s -m 'Fuss release <versione>' fuss/<versione>
```

in questo modo il tag verrà firmato con la propria chiave gpg di default; per non firmare il tag:

```
$ git tag -a -m 'Fuss release <versione>' fuss/<versione>
```

Ricordarsi di effettuare il push dei tag verso il server:

```
$ git push --tags
```

2.2 Build dei pacchetti in chroot

Nel caso ci siano problemi con l'uso di cowbuilder, è anche possibile usare una semplice chroot all'interno della quale installare gli strumenti di build e clonare il pacchetto.

2.2.1 Setup

Per creare una chroot ed installare gli strumenti di base:

```
# mkdir <versione>_build
# debootstrap <versione> <versione>_build (<mirror>)
# chroot <versione>_build
# apt install debhelper devscripts dpkg-dev
```

dove <versione> è al momento (marzo 2019) jessie per il FUSS server, stretch per il FUSS client e buster per le versioni future di server e client.

2.2.2 Build

Una volta clonato il repository (dentro la chroot), incrementato il numero di versione come sopra ed eventualmente generato il tar sorgente, per eseguire il build del pacchetto eseguire, nella directory principale del repository:

```
# dpkg-buildpackage -us -uc
```

Se la procedura va a buon fine, nella directory superiore si troveranno i pacchetti .deb generati, e anche i file .changes, .dsc e .tar.gz con il sorgente del pacchetto.

La procedura potrebbe fallire con un errore contenente:

```
Unmet build dependencies: <pacchetto1> <pacchetto2>
```

in tal caso installare semplicemente i pacchetti e riprovare. Tali dipendenze sono elencate nel campo Build-Depends del file debian/control, nel caso ci si voglia assicurare di averle già installate prima di buildare.

A questo punto si può procedere con test, commit+push ed upload come nel caso generale.

2.3 Policy di versionamento

2.3.1 Software sviluppato per FUSS

Per i pacchetti sviluppati specificatamente per FUSS possono esserci policy specifiche indicate nella relativa guida sviluppatori e/o nei README dei progetti.

In generale, lo schema usato prevede che la major version corrisponda alla versione di fuss per cui è rilasciato il pacchetto (che a sua volta corrisponde alla versione di debian su cui è basta). Un pacchetto per FUSS 8 avrà quindi versione tipo 8.X.Y, uno per FUSS 9 9.X.Y eccetera.

I pacchetti possono essere nativi o meno: nel primo caso il numero di versione è del tipo 9.X.Y sia per il pacchetto che in setup.py, mentre nel secondo si aggiunge un numero di revisione, es. 9.X.Y-Z; quest'ultimo va incrementato quando la nuova versione presenta modifiche nella pacchettizzazione (ovvero nella directory debian), ma non nel codice.

I pacchetti nativi devono anche avere 3.0 (native) nel file debian/source/format, mentre i pacchetti non-nativi devono avere 3.0 (quilt) e per buildarli è necessario generare una tarball sorgente (<nome>_<9.X.Z>.orig.tar.gz), ad esempio tramite debsrc.

2.3.2 Rebuild di pacchetti di debian

Per i pacchetti presi da debian e ribuildati da noi seguiamo una convenzione simile a quella usata dai backports aggiungendo ~fussN-X al numero di versione, dove N è la versione di FUSS per la quale stiamo preparando il pacchetto e X la revisione del backport.

Se si fa una rebuild di un pacchetto che ad esempio ha versione 1.2.3-4 la nostra versione sarà 1.2.3-4~fuss9+1 (+2 per una rebuild successiva con modifiche alla sola pacchettizzazione, eccetera).

2.4 Configurazione del repository

Il file `/iso/repo/conf/distributions` definisce le distribuzioni utilizzate nel repository, con snippet di configurazione come:

```
Origin: FUSS
Label: FUSS
Suite: jessie
Codename: jessie
Version: 8.0
Architectures: i386 amd64 source
Components: main contrib
Description: FUSS 8.0
SignWith: C00D47EF47AA6DE72DFE1033229CF7A871C7C823
```

inoltre nello stesso file sono definite le versioni precedenti e future della distribuzione. Al momento attuale la configurazione riguarda fino alla versione 10 di Debian (codename `buster`).

Le varie distribuzioni sono raggiungibili da apt usando, in `/etc/apt/sources.list`:

```
deb http://archive.fuss.bz.it CODENAME_DISTRIBUZIONE main contrib
```

e la chiave con la quale viene firmato il repository si può installare su una macchina debian o derivate eseguendo, da root:

```
# wget -qO - https://archive.fuss.bz.it/apt.key | apt-key add -
```

2.4.1 Aggiunta di nuova distribuzione e/o nuovo repository

Oltre al file `/iso/repo/conf/distributions` per indicare la nuova distribuzione e/o nuovo repository, è necessario:

- Creare una cartella per lo spool di incoming dei pacchetti in `/iso/incoming/<nuova distribuzione>`
- Aggiungere la descrizione e il path relativo al punto precedente nel file `/iso/repo/conf/incoming`
- Aggiungere allo script `/iso/bin/post-upload` l'esecuzione del processing del nuovo path di incoming. In questo script vanno tolte quelle non più usate quando è certo che non ci saranno più nuovi pacchetti per una specifica distribuzione.

2.5 Pacchettizzazione gestita con git

Nei progetti più recenti si è adottata una delle convenzioni in uso in Debian per la pacchettizzazione basata su git.

- I branch di sviluppo del progetto, incluso `master` non contengono la directory `debian`, come da raccomandazione della UpstreamGuide di Debian.
- I branch il cui nome inizia per `fuss/` contengono la directory debian; generalmente il branch usato per gli upload della versione corrente sarà `fuss/master`.
- Ad ogni rilascio, il branch `master` viene mergiato in `fuss/master` (ma *mai* il contrario) e il pacchetto può essere generato con i metodi descritti sopra.

Nel caso si voglia effettuare la build con `gbp` (pacchetto `git-buildpackage` il comando da usare sarà:

```
gbp buildpackage \
--git-pbuilder \
--git-no-pristine-tar \
```

(continues on next page)

(continua dalla pagina precedente)

```
--git-debian-branch=fuss/<versione> \
--git-dist=fuss-buster
```

aggiungendo `--git-export=WC` per fare build di prova dello stato attuale della working directory (anziché dello stato all'ultimo commit) oppure `--git-ignore-new` per fare una build corrispondente all'ultimo commit, ignorando le modifiche eventualmente presenti.

2.6 Pacchetti particolari

2.6.1 coova-chilli

Il pacchetto coova-chilli presente su archive.fuss.bz.it è generato da un nostro repository https://work.fuss.bz.it/git/coova-chilli copia del repository upstream https://github.com/coova/coova-chilli.git alla quale abbiamo aggiunto alcune modifiche di pacchettizzazione.

In particolare, per la release 1.4 è presente un branch `1.4-patched` con dei bugfix alla pacchettizzazione che sono stati nel frattempo accettati upstream per le versioni successive.

Per effettuare nuove build della versione 1.4 è quindi necessario usare il branch `1.4-patched`, mergiandovi eventuali modifiche upstream desiderate; usando `git-buildpackage` si dovrà usare:

```
$ gbp buildpackage --git-debian-branch=1.4-patched [--git-pbuilder]
```

Per versioni successive si possono invece usare i tag pubblicati da upstream.

Altri branch presenti sul nostro repository contengono la pacchettizzazione per versioni precedenti di FUSS, di utilità solo storica.

CAPITOLO 3

Metapacchetti

La distribuzione FUSS comprende alcuni metapacchetti per semplificare l'installazione di programmi didattici o comunque utili in ambito scolastico, gestiti nel progetto fuss-software

3.1 Repository

Il repository dei metapacchetti si può clonare con:

```
$ git clone https://work.fuss.bz.it/git/fuss-software
```

Il branch `master` si riferisce all'ultima release di FUSS, le versioni precedenti sono nei branch chiamati con il codename della distribuzione relativa.

3.2 Modifica dei metapacchetti

I file della cartella `metapackages` si riferiscono al metapacchetto con lo stesso nome e contengono le dipendenze, una per riga e preferibilmente in ordine alfabetico.

Avvertimento: Le dipendenze devono essere presenti all'interno dei repository configurati, altrimenti il pacchetto non sarà più installabile.

A luglio 2018 questo significa che i pacchetti che si desidera installare devono essere presenti in Debian 9 "stretch" nelle sezioni `main` e `contrib`.

3.2.1 Numero di versione

La major version dei metapacchetti deve rispecchiare la versione della distribuzione Debian utilizzata; ad esempio un pacchetto per la versione "stretch" avrà come major version 9.

Il patch level va aumentato di uno ad ogni versione, come di consueto.

I metapacchetti sono nativi, quindi non deve essere presente una versione debian, ma solo le tre componenti MAJOR.MINOR.PATCH.

Per build del pacchetto e upload delle modifiche vedere *Pacchetti e Repository*

CAPITOLO 4

FUSS Server

`fuss-server` è uno script python che lancia un playbook ansible che configura una macchina per poter funzionare come server in una rete FUSS.

4.1 fuss-server

Lo script `fuss-server` è scritto per essere compatibile con python 2 e 3; nel momento in cui ansible passerà ad usare python 3 si potrà eliminare la compatibilità python 2.

I vari sottocomandi corrispondono alle funzioni con lo stesso nome e generalmente si concludono con l'`os.execvp` di un comando di shell per lanciare ansible; notare che questo termina l'esecuzione del programma python, eventuale codice successivo non viene eseguito.

4.2 Playbook

Ansible viene chiamato con uno dei seguenti playbook, a seconda del sottocomando usato:

`create.yml` per configurare da zero un fuss-server.

`upgrade.yml` per aggiornare la configurazione di un fuss-server.

`captive_portal.yml` per applicare la configurazione aggiuntiva necessaria sui captive portal.

`purge.yml` per eliminare la configurazione del fuss-server.

Quest'ultimo ripristina alcuni file di configurazione dai backup, gli altri non compiono direttamente azioni, ma richiamano ruoli dalla directory `roles`, in modo da poter condividere il codice, in particolare tra `create` e `upgrade`.

4.3 Pacchetti Debian

Il repository prevede la generazione di due pacchetti .deb, `fuss-server` e `fuss-server-dependencies`; il primo contiene il fuss-server vero e proprio, mentre il secondo è un metapacchetto che dipende da tutti i pacchetti installati dal playbook ansible.

`fuss-server-dependencies` non è necessario per l'uso di fuss-server, ma è aggiunto per comodità per pre-installare (e soprattutto pre-scaricare) tutti i pacchetti necessari.

Per le istruzioni su come buildare i pacchetti e caricarli su archive.fuss.bz.it si può vedere l'articolo *Pacchetti e Repository*

4.3.1 Numeri di versione

Il pacchetto `fuss-server` è nativo, quindi il numero di versione è del tipo X.Y.Z dove X è il numero di versione debian corrispondente (ad esempio 8 per jessie, 9 per stretch, 10 per buster).

4.3.2 Dipendenze

Alcune delle dipendenze del pacchetto fuss-server, in particolare ansible (nella versione richiesta) e python-ruamel.yaml sono disponibili solo in `jessie-backports`; per installare il pacchetto è necessario aver abilitato quel repository, e per usarlo è necessario avere anche il repository di FUSS.

CAPITOLO 5

FUSS Client

`fuss-client` è uno script python che lancia un playbook ansible che configura una macchina come client in una rete FUSS.

5.1 fuss-client

Lo script `fuss-client` è scritto per python 3.

Le opzioni `-a | -U | -r | -l` sono mutualmente esclusive e corrispondono rispettivamente ai metodi `add`, `upgrade`, `remove` e `listavail`; ad eccezione di quest'ultimo si concludono con l'`os.execvp` di un comando di shell per lanciare ansible; notare che questo termina l'esecuzione del programma python, eventuale codice successivo non viene eseguito.

Prima della configurazione, l'opzione `-a` ricerca e contatta un fuss-server (metodi `_test_connection` e `_get_cluster`) per aggiungere la macchina corrente ad un cluster, tramite l'api di octofuss.

Notare che non esiste un'api corrispondente per rimuovere una macchina da un cluster, operazione che va svolta lato server.

Il passo successivo è la generazione di una chiave kerberos per il client: questa operazione viene svolta sul server dallo script `add_client_principal`, richiamato tramite ssh, quindi la chiave viene copiata localmente tramite scp. Per l'autenticazione sul server, sono supportati vari casi: accesso come root, accesso come utente con permessi sudo, oppure accesso con chiave con permessi limitati alle sole operazioni necessarie per lo script.

5.2 Playbook

Ansible viene chiamato con uno dei seguenti playbook, a seconda del sottocomando usato:

`connect.yml` per configurare un fuss-client

`remove.yml` per eliminare la configurazione del fuss-client.

Quest'ultimo ripristina alcuni file di configurazione dai backup, il primo non compie direttamente azioni, ma richiama ruoli dalla directory `roles`.

5.2.1 Compatibilità raspbian

Alcuni task, ed in particolare quelli relativi a lightdm, non vanno eseguiti quando la distribuzione base non è fuss-client (o una normale Debian), ma Raspbian, che richiede alcune personalizzazioni specifiche; per questi si usa la condizione `when: ansible_lsb.id != "Raspbian"`.

5.3 Pacchetti Debian

Il repository prevede la generazione di due pacchetti .deb, `fuss-client` e `fuss-client-dependencies`; il primo contiene il fuss-client vero e proprio, mentre il secondo è un metapacchetto che dipende da tutti i pacchetti installati dal playbook ansible.

`fuss-client-dependencies` non è necessario per l'uso di fuss-client, ma è aggiunto per comodità per pre-installare (e soprattutto pre-scaricare) tutti i pacchetti necessari.

5.3.1 Numeri di versione

Il pacchetto `fuss-client` è nativo, quindi il numero di versione è del tipo X.Y.Z dove X è il numero di versione debian corrispondente (ad esempio 8 per jessie, 9 per stretch, 10 per buster).

CAPITOLO 6

ISO

A partire da FUSS 9 (Debian stretch) si generano delle ISO live (complete di installer testuale e grafico) usando `live-wrapper`.

Vengono usate le versioni presenti in Debian buster, ovvero ad agosto 2018:

```
live-wrapper (0.7)
vmdebootstrap (1.11-1)
```

6.1 Build

6.1.1 Setup

- Su un'installazione di Debian buster o successive, installare live-wrapper:

```
# apt install live-wrapper
```

- Copiare il file `/usr/share/live-wrapper/customise.sh`:

```
# cp /usr/share/live-wrapper/customise.sh fuss-customise.sh
```

sotto alla riga:

```
. /usr/share/vmdebootstrap/common/customise.lib
```

aggiungere:

```
# overriden from the above for FUSS
prepare_apt_source() {
    # handle the apt source
    mv ${rootdir}/etc/apt/sources.list.d/base.list ${rootdir}/etc/apt/
    echo "deb $1 $2 main contrib non-free" > ${rootdir}/etc/apt/sources.list
    echo "deb-src $1 $2 main contrib non-free" >> ${rootdir}/etc/apt/sources.
↪list
    echo "deb http://archive.fuss.bz.it/ stretch main" >> ${rootdir}/etc/apt/
↪sources.list
    wget -qO ${rootdir}/tmp/fuss-apt.key https://archive.fuss.bz.it/apt.key
```

(continues on next page)

(continua dalla pagina precedente)

```
    chroot ${rootdir} apt-key add /tmp/fuss-apt.key
    chroot ${rootdir} apt -qq -y update > /dev/null 2>&1
}
```

6.1.2 Build

Per generare la ISO di FUSS 9 per architettura amd64, lanciare il seguente comando:

```
lwr -o fuss9-live-amd64.iso -d stretch --architecture=amd64 --customise=./fuss-
↪customise.sh -m http://ftp.de.debian.org/debian/ -e "fuss-client fuss-kids fuss-
↪children fuss-education fuss-graphics fuss-language-support fuss-multimedia fuss-
↪extra-multimedia fuss-net fuss-office fuss-various"
```

Per generare la ISO di FUSS 9 per architettura i386, lanciare il seguente comando:

```
lwr -o fuss9-live-i386.iso -d stretch --architecture=i386 --customise=./fuss-
↪customise.sh -m http://ftp.de.debian.org/debian/ -e "fuss-client fuss-kids fuss-
↪children fuss-education fuss-graphics fuss-language-support fuss-multimedia fuss-
↪extra-multimedia fuss-net fuss-office fuss-various"
```

6.2 Vedi anche

- https://live-wrapper.readthedocs.io/en/latest/
- https://wiki.debian.org/vmdebootstrap

CAPITOLO 7

Nuove versioni di Debian

Questa procedura è la procedura per realizzare l'aggiornamento di una versione di FUSS basata su una nuova release Debian stable.

7.1 Procedura di aggiornamento

7.1.1 Repository software

I repository git contenenti il software custom di FUSS dovranno ospitare le nuove versione specifiche per la versione upstream.

Il workflow attuale è il seguente:

branch `master`: attuale distribuzione debian stable

branch `codename_distribuzione`: contiene il codice rispetto ad una specifica versione di Debian, per backporting e altre correzioni

Quando viene rilasciata una nuova Debian, è necessario quindi creare il branch relativo alla nuova oldstable, e proseguire su master per l'attuale versione stabile.

Ad esempio, per passare da stretch a buster, su tutti i repository che contengono software pubblicato nell'archivio di FUSS:

```
(master) $ git pull
(master) $ git checkout -b stretch
(stretch) $ git push -u origin stretch
(stretch) $ git checkout master
(master) $ # proseguire con le modifiche...
```

7.1.2 Aggiornamento, build e test pacchetti

Ogni nuovo pacchetto dovrà come minimo:

- Essere aggiornato allo standard di riferimento Debian per la versione in uso (vedi https://www.debian.org/doc/debian-policy/).

- Avere una versione (e relativo changelog) che riporti, come major version number, quello della distribuzione Debian di riferimento. Ad esempio un pacchetto per Debian Buster avrà come versione 10.x.x.
- Essere aggiornato rispetto alle nuove versioni delle dipendenze presenti nella nuova versione della distribuzioone.
- Dovrà essere buildato *e testato* su una installazione della versione Debian di riferimento.

Per le istruzioni di build e di successivo upload si veda *Pacchetti e Repository*.

fuss-server e fuss-client

FUSS Server e *FUSS Client* intervengono sui file di configurazione di numerosi pacchetti: per il loro aggiornamento è opportuno fare delle verifiche specifiche sul loro funzionamento.

- Per i file in cui vengono inserite sezioni nei file di configurazione (task `lineinfile` e `blockinfile`) che le sezioni siano ancora inserite nel posto opportuno.
- Per i file che vengono completamente sovrascritti (task `copy` e `template`) è generalmente il caso di ripartire dal file di configurazione di default della nuova versione di debian e riapplicare le modifiche necessarie (in modo da ricevere le nuove eventuali impostazioni di default).

7.1.3 Immagini ISO

TBD

7.2 Informazioni sullo sviluppo upstream

7.2.1 Date di release

Contrariamente ad altre distribuzioni, Debian non fissa date di rilascio, dando maggiore priorità alla qualità della distribuzione. Vengono però fissate le date di *freeze*, il che permette di stimare con approssimazione di qualche mese quando avverrà il prossimo rilascio.

La freeze, avviene ogni due anni d'inverno; le date precise sono annunciate con largo anticipo e pubblicate su https://release.debian.org/; i rilasci avvengono generalmente[1] durante le estati degli anni dispari.

A partire dalla data della freeze non vengono più effettuati cambiamenti delle versioni di pacchetti presenti nella distribuzione, ma vengono solo accettati fix mirati per bug sufficientemente importanti; questo per FUSS vuol dire che eventuali test di aggiornamento di versione saranno già corrispondenti al comportamento della versione rilasciata.

[1] Ulteriori statistiche sono presenti su https://wiki.debian.org/DebianReleases#Release_statistics Notare che la data della freeze è stata spostata in avanti di due mesi tra jessie e stretch.

CAPITOLO 8

Macchine virtuali con libvirt+qemy/kvm per fuss-server e fuss-client

Per fare test di fuss-server e fuss-client è utile avere a disposizione delle macchine virtuali; specialmente se si lavora su debian stretch o successive (dove VirtualBox non è più disponibile) è comodo usare qemu/kvm tramite libvirt.

8.1 Installazione e configurazione

8.1.1 Installazione di libvirt

- Installare i seguenti pacchetti:

```
apt install qemu libvirt-clients libvirt-daemon virtinst \
libvirt-daemon-system virt-viewer virt-manager dnsmasq-base
```

- Aggiungere il proprio utente ai gruppi `libvirt` e `kvm`:

```
adduser $UTENTE libvirt
adduser $UTENTE kvm
```

Una volta che l'utente fa parte dei gruppi (ad esempio previo logout/ri-login) si può usare `virt-manager` per gestire macchine virtuali e reti tramite un'interfaccia grafica simile a quella di VirtualBox.

Volendo invece usare la riga di comando, si può proseguire con questa guida.

8.1.2 Configurazione della rete

fuss-server richiede la configurazione di almeno due, in alcuni casi tre, schede di rete: una con accesso ad internet e due su rete isolata.

Per creare queste interfacce di rete da riga di comando se ne deve scrivere il file di configurazione e passarlo al comando `virsh net-define`

Per la rete nattata, creare il file `natted.xml` con i seguenti contenuti, sostituendo a `eth0` il nome della propria scheda di rete e a `8.8.8.8` quello di un server dns opportuno:

```
<network>
  <name>natted</name>
  <forward dev='eth0' mode='nat'>
    <interface dev='eth0'/>
  </forward>
  <bridge name='virbr7' stp='on' delay='0'/>
  <dns>
    <forwarder addr='8.8.8.8'/>
  </dns>
  <ip address='192.168.7.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.7.128' end='192.168.7.254'/>
    </dhcp>
  </ip>
</network>
```

quindi lanciare, come root:

```
# virsh net-define natted.xml
# virsh net-start natted
```

Similmente, per le interfacce isolate si può usare quanto segue, nel file `isolated.xml`:

```
<network>
  <name>isolated</name>
  <bridge name='virbr6' stp='on' delay='0'/>
  <ip address='192.168.6.253' netmask='255.255.255.0'>
  </ip>
</network>
```

e nel file `isolated2.xml`:

```
<network>
  <name>isolated2</name>
  <bridge name='virbr8' stp='on' delay='0'/>
  <ip address='192.168.8.253' netmask='255.255.255.0'>
  </ip>
</network>
```

E come prima, sempre con utente root:

```
# virsh net-define isolated.xml
# virsh net-start isolated
# virsh net-define isolated2.xml
# virsh net-start isolated2
```

In questo modo le interfacce sono definite, ma non verranno avviate automaticamente; per farlo usare i seguenti comandi:

```
# virsh net-autostart natted
# virsh net-autostart isolated
# virsh net-autostart isolated2
```

oppure usare `net-start` per ciascuna interfaccia quando se ne ha bisogno.

8.1.3 Creazione delle macchine virtuali

Per creare la macchina che ospiterà il server, dopo aver abilitato le interfacce di rete e scaricato l'iso di fuss-server lanciare il seguente comando:

```
$ virt-install --connect qemu:///system --name fuss_server --memory 1024 \
  --cdrom $PATH_ISO_FUSS-SERVER --network network=natted \
  --network network=isolated --network network=isolated2 \
  --disk size=16,format=qcow2 --os-variant debian8
```

questo creerà una macchina virtuale che fa il boot dall'iso dell'installer e aprirà una finestra di virt-viewer per controllarla. Alla fine dell'installazione si può fare login e procedere con l'Installazione di Fuss Server

Una volta che il server è installato e configurato si può fare la stessa cosa per una (o più) macchine client:

```
$ virt-install --connect qemu:///system --name fuss_client --memory 1024 \
  --cdrom $PATH_ISO_FUSS-CLIENT --network network=isolated \
  --disk size=24,format=qcow2 --os-variant debian9
```

8.1.4 Boot delle macchine

Per avviare le volte successive le macchine è necessario:

- Se l'host è stato spento, e non è stato configurato l'autoavvio delle interfacce di rete, abilitarle:

```
# virsh net-start natted
# virsh net-start isolated
# virsh net-start isolated2
```

- Avviare la macchina di cui si ha bisogno:

```
$ virsh --connect qemu:///system start fuss-server
```

- Se necessario, avviare una sessione grafica sulla macchina:

```
$ virt-viewer --connect qemu:///system fuss-server
```

8.2 Configurazioni del sistema

8.2.1 Configurazione della rete

All'interno del fuss-server, le interfacce di rete come definite in questa pagina possono essere configurate aggiungendo a `/etc/network/interfaces` le seguenti righe:

```
allow-hotplug eth0
iface eth0 inet dhcp

allow-hotplug eth1
iface eth1 inet static
    address 192.168.6.1
    netmask 255.255.255.0
```

In fuss-client non è invece necessaria nessuna configurazione, dato che viene usato dhcp come da default.

8.3 Vedi anche

In caso di problemi o per approfondire l'uso da riga di comando di libvirt è molto utile la pagina su Libvirt della wiki di Arch Linux , la maggior parte della quale si applica anche a sistemi Debian o Debian-based.

CAPITOLO 9

Contribuisci

Chiunque può contribuire a migliorare questa documentazione che è scritta in reStructuredText.

CAPITOLO 10

Supporto

Se ti serve aiuto, scrivi una mail ad info@fuss.bz.it

CAPITOLO 11

Licenze