The Falcon repository builder
Version 2.0, beta 1

Dennis Kaarsemaker dennis@kaarsemaker.net

July 4 2007

Contents

1 Introduction

This documents describes the usage of falcon. I wrote falcon because none of the existing repository maintenance tools fitted my needs. Falcon is very easy to setup, extremely easy to use and has a few features I need a lot:

No hassle with `incoming/` Most repository maintenance tools work with an `incoming/` system and override files, I don't like that. I work on my packages right inside the `pool/` directory. You can however still use an `incoming/` dir and install one package at the time. It is still flexible though and you can easily install packages from eg pbuilder's result dir into falcon.

Components I divide my repository into logical components (not main and non-free, but backports, extras, drivers etc...). Falcon handles this transparently and without override files.

HTML indices Falcon can create HTML pages for your repository, listing all releases, components, packages and changelogs. These HTML pages are customized per mirror listing sources.list lines to add and other repository information.

Multi-release support Falcon supporats multiple releases just like it supports multiple components.

Mirror synchronization I maintain my repository on my adsl-line hosted server which is not online 24/7. When it became more popular, people offered mirroring. Falcon supports dead-easy synchronization with mirrors via rsync or ssh, without disabling working directly in the pool dir.

Creating .iso images Falcon can very easily create .iso images from a directory of .deb files (Currently limited to 1 image max).

Autobuilding Falcon can soon autobuild your packages after uploading, turning it into a full-fledged repository management system.

2 Setting up a repository

2.1 Prerequisites

The best way to install Falcon is via a .deb archive. You can also pull it from .bzr however and not work via a .deb file. In that case you will need to manually install the following:

- python 2.4
- dpkg-dev
- python-apt
- gzip and bzip2
- rsync

- django (python-django)
- python-newt
- mkisofs (If you want the image creation function)
- ssh-agent can be very useful during syncing

For hosting the repository you need an http server such as apache 2 and/or an ftp server such as vsftpd.

Falcon can cryptographically sign your repository. For this you need to have gnupg installed and you need to have a signing key in your gpg keyring. If you don't know how to create a key, read appendix A for an example.

2.2 Creating the directory tree

You need a few directories for Falcon to function. First you need your repositories root, in this example I'll use `/var/www/falcon`.

```
dennis@mirage:~$ mkdir -p /var/www/falcon
dennis@mirage:~$ cd /var/www/falcon
```

Packages are located in a pool, which you have to create.

```
dennis@mirage:/var/www/falcon$ mkdir pool
dennis@mirage:/var/www/falcon$ cd pool
```

Packages usually are released in distribution versions. You need to name your initial version now and create a directory for it. I started with breezy-seveas since my first packages were aimed at Ubuntu Breezy.

```
dennis@mirage:/var/www/falcon/pool$ mkdir breezy-seveas
```

Then you need to divide your packages into components. For each of the components you need to make a directory. breezy-seveas contains the components extras, backports and drivers

```
dennis@mirage:/var/www/falcon/pool$ mkdir breezy-seveas/extras
dennis@mirage:/var/www/falcon/pool$ mkdir breezy-seveas/backports
dennis@mirage:/var/www/falcon/pool$ mkdir breezy-seveas/drivers
```

After that you place your .deb files in the directories of your components. If a source or package needs to appear in multiple sections, you can use symlinks so you don't wast diskspace and bandwith to mirrors. Symlinks to files outside the basedir will not work properly though.

2.3 Creating the configuration

After creating the directory infrastructure, you need to configure falcon. You do this by running `falcon configure`. This will launch an interactive configuration editor.

You can specify your repository root with the `-r` flag, but it's easier to set a default repository root. You can do this by creating a symlink in `/.falcon` to your repository root

```
dennis@mirage:~$ mkdir -p ~/.falcon
dennis@mirage:~$ ln -s /var/www/falcon ~/.falcon/rootdir
```

Every time you add a component or a section you should configure it with `falcon configure`. . For now, you should only look at the *General Configuration* and *Pockets and Components* sections. Configuration is very straightforward and all options are explained.

For some items, the configuration engine will launch an external editor, you can select which editor to use with the `-e` flag or the `$EDITOR` environment variable.

If you plan to serve your .deb files on a publicly accessible server, you should seriously consider keeping the searchbot out of your `pool/` directory by adding the following to the robots.txt in your web root (that what corresponds to http://yourdomain.com, which is not neccessarily your repository root).

```
User-agent: *
Disallow: /falcon/pool/
```

Make sure your mirrors do the same thing. Replace `/falcon/pool` with the path to your pool directory.

2.4 Metacomponents

One of the features of Falcon is the creation of metacomponents. Metacomponents contain all packages in several components and mak it easy to add components to your repository without annoying your users by making them change their `sources.list`. You can create/modify/delete metacomponents with the interactive configuration editor.

3 Invoking falcon

Having configured the beast, you are now ready to actually use falcon and turn your pile of .deb files into a proper repository.

3.1 Updating the repository

After the initial hurdle of creating the directory tree and configuring Falcon, updating your repository is surprisingly easy. After adding/moving/removing a file you simpy run this command:

```
dennis@mirage:~$ falcon scan
```

That's it! Falcon will re-scan your pool directory and update its internal database. To update the Packages/Sources lists as well as the html files, run the following command:

```
dennis@mirage:~$ falcon export
```

To scan/export only a single pocket or a single component, use the `-P` and the `-C` flags to falcon. To avoid scanning altogether, you can install single sources with the install command:

```
dennis@mirage:~$ falcon install -P feisty-seveas -C freenx freenx_0.6.0-0~seveas1.ds
```

To install only packages that have been built completely, use the `-c` flag:

```
dennis@mirage:~$ falcon install -c -P feisty-seveas -C freenx freenx_0.6.0-0~seveas
```

Often you will include new versions of packages in your repository. To prevent your repository being cluttered by lots of old packages, falcon can clean out older versions of your packages. It will prompt you if it finds older packages and can then move them to a designated morgue directory. To make the scanning noninteractive use the `-y` (force yes) or `-n` (force no) flags.

3.2 Creating .iso images

If you want to create an .iso image, forget everything you just read. Creating .iso images requires NO configuration at all. You simply need to run falcon and give it the directory with .deb files as rootdir

```
dennis@mirage:~$ falcon iso -r ~/collcted_debs/
```

The .iso image will be created in `/tmp`.

3.3 Creating app-install data

To write

3.4 Creating HTML indices

To set up the HTML indexing you need to set the settings for webbase and template in the configuration editor. Templates can live in the following locations:

- `/usr/share/falcon/templates`
- `/.falcon/templates`
- `.falcon/templates` in your repository root

You can easily modify or create new templates. For more information about that, see appendix **??**.

3.5 Synchronizing with mirrors

Synchronizing with mirrors takes a bit more configuration. But after that, synchronizing with mirrors is just as easy as updating your repository.

You can manage your mirror settings with the interactive configuration editor. It's fairly straightforward, but beware of the rsync path:

The rsync path of a mirror can have several forms:

An rsync url like rsync://example.com/your_mirror is an rsyncd mirror, usually rigged to let you in based on your IP address

An ssh path like example.com:/var/www/mirror will mean uploading via ssh. Using ssh-keys and ssh-agent (falcon will automatically start ssh-agent if it finds ssh-keys) will drastically reduce the number of times you have to enter your password.

A local path like /path/to/file A simple local file mirror, useful for mirrors that can only pull (ie, you can't push to them so you'll have to setup an rsyncd/sshd/ftpd yourself to allow pulling)

A blank entry indicates that no syncing should take place

VERY IMPORTANT: the remote mirrors must at least have rsync version 2.5; older versions contain a bug where sometimes rsync deletes far too much which it subsequently uploads again.

Now that you configured it, the actual synchronizing is so simple, it's almost an anticlimax. All synchronization is done with the following command:

```
dennis@mirage:~$ falcon sync
```

To sync to only a specific mirror:

```
dennis@mirage:~$ falcon sync mirrorname
```

You're now all set to create, manage and maintain a repository of software, have fun!

4 Importing packages

If you want to import source packages from another repository into your own (eg for backports or customization), you can use the falcon-import tool. With this tool, it takes one command to import a source package into your repository. If, for example, you want to import zsnes from Ubuntu Dapper multiverse, you issue this command:

```
dennis@mirage:~$ falcon-import -P dapper-seveas -C backports \
http://archive.ubuntu.com/ubuntu edgy multiverse zsnes
```

If you don't want to import into a repository, just don't specify a pocket and component and it will download to the current directory.

For more information about falcon-import, see its manpage.

5 Bugs, updates and contact information

Falcon is developed and maintained by Dennis Kaarsemaker ¡dennis@kaarsemaker.net¿. Bugs and feature requests can be filed at https://bugs.launchpad.net/falcon/+filebug

When new versions introduce backwards-incompatible changes, they will be added to the documentation. Always read the `upgrading` file before upgrading to a new release.

A Creating a GPG key

```
dennis@mirage:~$ gpg --gen-key
Please select what kind of key you want:
   (1) DSA and Elgamal (default)
   (2) DSA (sign only)
   (5) RSA (sign only)
Your selection? 1
DSA keypair will have 1024 bits.
ELG-E keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y
```

```
You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Dennis Kaarsemaker
Email address: dennis@kaarsemaker.net
Comment: Packages Key
You selected this USER-ID:
    "Dennis Kaarsemaker (Packages Key) <dennis@kaarsemaker.net>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
+++ [cut some text ]
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
+++ [cut some text]
key 5E1E9DA9 marked as ultimately trusted
public and secret key created and signed.

pub   1024D/5E1E9DA9 2006-01-10
      Key fingerprint = FF60 6941 DAC6 AEC0 67FE  D647 614F E7C0 5E1E 9DA9
uid                  Dennis Kaarsemaker (Packages Key) <dennis@kaarsemaker.net>
sub   2048g/9F4B4632 2006-01-10
```

Don't forget to uplad your key to a keyserver:

```
dennis@mirage:~$ gpg --server hkp://subkeys.pgp.net --send-keys 5E1E9DA9
```

B Template API

The templating system uses django, for more information about django's template system, visit http://www.djangoproject.com/documentation/templates/. Templates should be in their own directory, the name of this directory is the name of the template. Three templates should be crated: base.html, pocket.html and component.html.

Each template should use the {% api %} tag so it'll be used only with compatible versions of falcon. The following variables are available in the templates:

base.html gets the following variables

conf	The falcon configuration
dots	A number of concatenated '../' strings to use in relative links that point to the rootdir
pockets	The available pockets
mirrors	The available mirrors
mirror	The mirror for which html is currently being generated (if any)

pocket.html gets the following variables

conf	The falcon configuration
dots	A number of concatenated '../' strings to use in relative links that point to the rootdir
p	The current pocket
components	The available components in this pocket

component.html gets the following variables

conf	The falcon configuration
dots	A number of concatenated '../' strings to use in relative links that point to the rootdir
c	The current component
sources	The source packages in the current component

The default template is a rather simple template which is usable as learning material if you want to crate your own template.

C Plugin API

Needs to be written