Assaf Morami

Hosting your own PPA repository on GitHub

May 2, 2019 • Assaf Morami

Publishing your own Debian packages and hosting it on a GitHub repo is pretty easy. This is a quick HowTo.

A PPA repo can be as simple as one directory

```
.
└── my_ppa
    ├── my_list_file.list
    ├── InRelease
    ├── KEY.gpg
    ├── Packages
    ├── Packages.gz
    ├── Release
    ├── Release.gpg
    ├── package-a_0.0.1_amd64.deb
    ├── package-a_0.0.2_amd64.deb
    ├── package-b_0.1.0_amd64.deb
    ├── package-b_0.1.1_amd64.deb
    ├── ...
    └── package-z_1.0.0_amd64.deb
```

A working example can be found in https://github.com/assafmo/ppa.

You can name `my_ppa` and `my_list_file.list` whatever you like. I used those names because it's hard to name things.

Also don't forget to replace `${GITHUB_USERNAME}` with your GitHub user name and `${EMAIL}` with your email address.

0. Creating a GitHub repo with your deb packages

Create a GitHub repo. We'll call it `my_ppa`. Then go to `https://github.com/${GITHUB_USERNAME}/my_ppa/settings`, and under `GitHub Pages` select `Source` to be `master branch`.

Any HTTP server will work just fine, but GitHub pages is free, easy and fast.

Now clone the repo and put all of your debian packages inside:

```
git clone "git@github.com:${GITHUB_USERNAME}/my_ppa.git"
cd my_ppa
cp /path/to/my/package-a_0.0.1_amd64.deb .
```

1. Creating a GPG key

Install `gpg` and create a new key:

```
sudo apt install gnupg
gpg --full-gen-key
```

Use RSA:

```
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
```

RSA with 4096 bits:

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096
```

Key should be valid forever:

```
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
```

Enter your name and email:

```
Real name: My Name
```

```
Email address: ${EMAIL}
Comment:
You selected this USER-ID:
"My Name <my.name@email.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
```

At this point the `gpg` command will start to create your key and will ask for a passphrase for extra protection. I like to leave it blank so when I sign things with my key it won't promp for the passphrase each time.

```
We need to generate a lot of random bytes. It is a good idea to perfo
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.
We need to generate a lot of random bytes. It is a good idea to perfo
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.
gpg: key B58FBB4C23247554 marked as ultimately trusted
gpg: directory '/home/assafmo/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/assafmo/.gnupg/openpgp-r
public and secret key created and signed.

pub rsa4096 2019-05-01 [SC]
31EE74534094184D9964EF82B58FBB4C23247554
uid My Name <my.name@email.com>
sub rsa4096 2019-05-01 [E]
```

You can backup your private key using:

```
gpg --export-secret-keys "${EMAIL}" > my-private-key.asc
```

And import it using:

```
gpg --import my-private-key.asc
```

2. Creating the `KEY.gpg` file

Create the ASCII public key file `KEY.gpg` inside the git repo `my_ppa`:

```
gpg --armor --export "${EMAIL}" > /path/to/my_ppa/KEY.gpg
```

Note: The private key is referenced by the email address you entered in the previous step.

3. Creating the `Packages` and `Packages.gz` files

Inside the git repo `my_ppa`:

```
dpkg-scanpackages --multiversion . > Packages
gzip -k -f Packages
```

4. Creating the `Release`, `Release.gpg` and `InRelease` files

Inside the git repo `my_ppa`:

```
apt-ftparchive release . > Release
gpg --default-key "${EMAIL}" -abs -o - Release > Release.gpg
gpg --default-key "${EMAIL}" --clearsign -o - Release > InRelease
```

5. Creating the `my_list_file.list` file

Inside the git repo `my_ppa`:

```
echo "deb [signed-by=/etc/apt/trusted.gpg.d/my_ppa.gpg] https://${GIT
```

This file will be installed later on in the user's `/etc/apt/sources.list.d/` directory. This tells `apt` to look for updates from your PPA in `https://${GITHUB_USERNAME}.github.io/my_ppa`.

That's it!

Commit and push to GitHub and your PPA is ready to go:

```
git add -A
git commit -m "my ppa repo is now hosted on github"
git push -u origin master
```

Now you can tell all your friends and users to install your PPA this way:

```
curl -s --compressed "https://${GITHUB_USERNAME}.github.io/my_ppa/KEY
sudo curl -s --compressed -o /etc/apt/sources.list.d/my_list_file.lis
sudo apt update
```

Then they can install your packages:

```
sudo apt install package-a package-b package-z
```

Whenever you publish a new version for an existing package your users will get it just like any other update.

How to add new packages

Just put your new `.deb` files inside the git repo `my_ppa` and execute:

```
# Packages & Packages.gz
dpkg-scanpackages --multiversion . > Packages
gzip -k -f Packages

# Release, Release.gpg & InRelease
apt-ftparchive release . > Release
gpg --default-key "${EMAIL}" -abs -o - Release > Release.gpg
gpg --default-key "${EMAIL}" --clearsign -o - Release > InRelease

# Commit & push
git add -A
git commit -m update
git push
```

Sources

- Export and import a GPG key
- Creating your own Signed APT Repository and Debian Packages
- Create your own custom and authenticated APT repository
- A vscode ppa example by @tagplus5
- What is the simplest Debian Packaging Guide?

Assaf Morami

Assaf Morami
assaf.morami@gmail.com

assafmo

Things I have learned over time. :-)