Setup Debian Repository (Apache)

This tutorial will cover how to create a debian (`.deb`) package, which just prints one line when executed. This package is then hostet on an apache2-based repository.

Prerequisites: Install following packages:

```
sudo apt-get install -y gcc dpkg-dev gpg curl apt-utils
```

Build custom package

Create executeable package

Firstly create directory `~/build/`:

```
mkdir -p ~/build/firstpkg/
```

Enter the directory and create a file called `main.c` with the following content. You can use any editor therefore.

```
#include <stdio.h>
int main() {
    printf("That worked!\n");
    return 0;
}
```

Then create an executeable:

```
gcc -o firstpkg main.c
```

The default naming scheme for `.deb` packages looks like this:

`<package-name>_<version>-<release-number>_<architecture>`

We will replace these variables with the following:

Varibale	**Value**
package-name	`firstpkg`
version	`0.0.1`
release-number	1
architecture	`amd64/arm64`

The release number is usually set to 1, only in case there was an error when packaging, this number would be changed.

Next create a directory with replaced variables and the subdirectories `usr/bin` and `DEBIAN`:

```
mkdir -p ~/build/firstpkg_0.0.1-1_amd64
```

```
mkdir -p ~/build/firstpkg_0.0.1-1_amd64/usr/bin/
```

```
mkdir -p ~/build/firstpkg_0.0.1-1_amd64/DEBIAN
```

Now copy the executeable to the binary directory:

```
cp ~/build/firstpkg/firstpkg ~/build/firstpkg_0.0.1-1_amd64/usr/bin/
```

To clearly identify packages, each package requires a `control` file under `DEBIAN/`:

```
touch ~/build/firstpkg_0.0.1-1_amd64/DEBIAN/control
```

Add the following lines to this file:

```
Package: firstpkg
Version: 0.0.1
Maintainer: example <example@example.com>
Depends: libc6
Architecture: amd64
Homepage: http://example.com
Description: just a test package
```

> Remember to select the correct architecture (`arm64`/`amd64`/`armhf`/ or whatever you are creating the package for.)

Build a .deb package

Now build the package:

```
dpkg --build ~/build/firstpkg_0.0.1-1_amd64
```

If everything worked correctly there should be one file named `firstpkg_0.0.1-1_amd64.deb` as output in the current directory.

Finally you can view package information with:

```
dpkg-deb --info ~/build/firstpkg_0.0.1-1_amd64.deb
```

```
dpkg-deb --contents ~/build/firstpkg_0.0.1-1_amd64.deb
```

The output of the last command should contain the executeable.

Try to install this package with:

```
sudo apt install ./firstpkg_0.0.1-1_amd64.deb
```

And run it with:

```
firstpkg
```

If the output equals `That worked!` everything worked :).

Remove it again with:

```
sudo apt remove firstpkg
```

This part was sourced from earthly.dev - Creating and hosting your own deb packages.

Setup repository server

Therefore this guide uses the `apache` webserver.

Create directories

The root path should be `/srv/repo/`. Within this directory two subdirectories are required: `pool/main` and `dists/stable`. The the first one will contain all binaries, the second one the Release files.

```
mkdir -p /srv/repo/
```

```
mkdir -p /srv/repo/pool/main
```

```
mkdir -p /srv/repo/dists/stable
```

Setup apache2

This guide uses the `apache2` server, therefore install `apache2`:

```
apt install apache2
```

Next remove the default sites:

```
rm /etc/apache2/sites-available/000-default.conf
```

```
rm /etc/apache2/sites-available/default-ssl.conf
```

And create the new repository site::

```
sudo nano /etc/apache2/sites-available/repository.conf
```

The <servername> and <serveralias> variables are replaced with your servername, e.g. `repository.yourdomain.com`. And the `DocumentRoot` should be set to the point of your filesystem, where your repository should start, e.g. `/srv/repo`.

Paste this content with changed variables to the site config:

Show/Hide repository.conf

```
<VirtualHost *:80>
        # The ServerName directive sets the request scheme, hostname and
port that
        # the server uses to identify itself. This is used when creating
        # redirection URLs. In the context of virtual hosts, the ServerName
        # specifies what hostname must appear in the request's Host: header
to
        # match this virtual host. For the default virtual host (this file)
this
        # value is not decisive as it is used as a last resort host
regardless.
        # However, you must set it for any further virtual host explicitly.

        #ServerAdmin webmaster@localhost
        DocumentRoot </path/to/root>
        ServerName <servername>
        ServerAlias <serveralias>

        # Available loglevels: trace8, ..., trace1, debug, info, notice,
warn,
        # error, crit, alert, emerg.
        # It is also possible to configure the loglevel for particular
        # modules, e.g.
        #LogLevel info ssl:warn

        # HTTP Strict Transport Security (63072000 seconds)
        Header always set Strict-Transport-Security "max-age=63072000"

        # Setting this header will prevent MSIE from interpreting files as
something
        # else than declared by the content type in the HTTP headers.
        Header setifempty X-Content-Type-Options: "nosniff"

        # Setting this header will prevent location disclosure to third
party sites,
        # e.g. if a user follows a link outside of our SLD.
        Header setifempty Referrer-Policy: "strict-origin"
```

```
        # Block pages from loading when they detect reflected XSS attacks
        Header setifempty X-XSS-Protection: "1; mode=block"

        <Directory "/srv/repo">
                Allow from all
                AllowOverride All
                Options Indexes FollowSymlinks
                Require all granted

                <Files packages-amd64.db>
                        Require all denied
                </Files>

                <Files packages-arm64.db>
                        Require all denied
                </Files>

                <Files packages-armhf.db>
                        Require all denied
                </Files>
        </Directory>
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Create GPG Keys

To securely update from the repository the Releases file will be signed with a GPG - GNU Privacy Guard-key.

These keys will be generated in a safe environment, therefore create a temporary directory:

```
mktemp -d XXXX
```

The new four XXXX will automatically be replaced with random letters.

Next set the directory for GPG:

```
export GNUPGHOME=/path/to/tmp/dir
```

Finally generate the keypair with the following settings:

```
gpg --full-generate-key
```

Request	Selection
Kind of Key	1 (RSA and RSA)
Keysize	3072
Expiring	0 (never)

A name must be entered, email address and comment can be left empty. You *can* also enter a password or just skip the prompts, it's your decision.

To view all created keys use this command:

```
gpg --list-secret-keys --keyid-format=long
```

Now create the public key file:

```
gpg --armor --export <ID> > public.key
```

Replace <ID> with the value behind `sec  rsaXXXX/` when running the command above. To make the public key accessible so that it can be installed, copy it to the root-path of the repository, e.g.:

```
cp public.key public.key /srv/repo/PublicReleaseKey.gpg
```

Also export the secret/private key file:

```
gpg --armor --export-secret-keys <ID> > private.key
```

Remind to store this file carefully!

The `ID` of the just created key is later required to sign the Release files properly, so maybe save this ID somewhere.

If getting the error `No such file and directory` when running `gpg –full-generate-key` try restarting the gpg agent:

```
gpgconf --kill gpg-agent
```

Sourced from unix.stackexchange.org

Create Release files

To create the Packages, Contents and Release files we use the command `apt-ftparchive` that is part of the `apt-utils` package.

`apt-ftparchive` requires two configuration files:

Show/Hide files

`aptgenerate.conf`

```
Dir::ArchiveDir "/srv/repo/";
Dir::CacheDir "/srv/repo/";
TreeDefault::Directory "pool/";
TreeDefault::SrcDirectory "pool/";
Default::Packages::Extensions ".deb";
Default::Packages::Compress ". gzip bzip2";
```

```
Default::Sources::Compress ". gzip bzip2";
Default::Contents::Compress "gzip bzip2";

Tree "dists/stable" {
  Sections "main";
  Architectures "armhf amd64 arm64";
};
```

aptrelease.conf

```
APT::FTPArchive::Release {
  Origin "repository.yourdomain.com";
  Label "repository.yourdomain.com";
  Suite "stable";
  Codename "stable";
  Architectures "amd64 i386 arm64 armhf";
  Components "main";
  Description "My first repository!";
  Version "1.0";
};
```

> Remind to adjust values like `Dir::ArchiveDir`, `Dir::CacheDir`, `Origin`, `Label`, `Architectures` and `Description`.

Now create the `Packages` and `Contents` file:

```
apt-ftparchive -c=/path/to/aptrelease.conf generate
/path/to/aptgenerate.conf
```

Next create the unsigned `Release` file:

```
apt-ftparchive release -c=/path/to/aptrelease.conf <path-to-basedir> >
<path-to-basedir>/Release
```

In this case `path-to-basedir` would be `/srv/repo/dists/stable`.

More information about the Release files can be found here: wiki.debian.org

Sign Release file

Firstly create the `Release.gpg` file:

```
gpg --yes --pinentry-mode loopback --default-key <ID> -abs -o <path-to-
basedir>/Release.gpg <path-to-basedir>/Release
```

`path-to-basedir` would again be `/srv/repo/dists/stable`, and `ID` the ID of the created GPG key: Create GPG Keys.

And finally create the `InRelease` file:

```
gpg --yes --pinentry-mode loopback --default-key <ID> --clearsign -o <path-to-basedir>/InRelease <path-to-basedir>/Release
```

This file will later be sourced from APT to index the repository.

Add custom repository

Finally add your local repository. There are several ways:

Proper way

Firstly install the key:

```
curl -sS http://repository.yourdomain.com/PublicReleaseKey.gpg | gpg --dearmor | sudo tee /usr/share/keyrings/repository.yourdomain.com.gpg > /dev/null
```

The command gets the file, dearmors the downloaded file and saves the content (key) to a file within the keysrings directory. The `> /dev/null` suppresses the unreadable output of the –`dearmor` command.

Then add the repository to `/etc/apt/sources.list.d/`:

```
echo "deb [signed-by=/usr/share/keyrings/repository.yourdomain.com.gpg] http://repository.yourdomain.com/ stable main" | sudo tee /etc/apt/sources.list.d/repository.yourdomain.com.list
```

Now `apt` can source your local repsitory securely, try it by installing the test-package:

```
sudo apt update && sudo apt install firstpkg
```

Deprecated ways

The most simple way is to 'install' the key:

```
curl -sS http://repository.yourdomain.com/PublicReleaseKey.gpg | sudo apt-key add -
```

But this will show warnings (! not errors), everytime you update., because it's a legacy method to store keys for apt-repositories.

There's another deprecated way:

```
curl -sS http://repository.yourdomain.com/PublicReleaseKey.gpg | gpg --
dearmor | sudo tee /etc/apt/trusted.gpg.d/repository.gpg
```

It works also but isn't expected from the APT developers, consider using the method described above in this guide.

For both methods the repository has to be added this way:

```
sudo nano /etc/apt/sources.list.d/repository.yourdomain.com.list
```

With content:

```
# local - repository
deb http://repository.yourdomain.com/ stable main
```

Repository extensions

There is a little helper when self-hosting a repository, more information can be found here: Extensions for self hosted Repository

---

Additional settings

When using a standard dir as GNUPGHOME, set correct permissions on this directory:

```
find ~/.gnupg -type f -exec chmod 600 {} \;
find ~/.gnupg -type d -exec chmod 700 {} \;
```

Explanation for 600, 700:

Lets start from the back: 00 mean NO rights AT ALL for everybody who is not the owner of the files/directories.

That means, that the process reading these (gnupg) must run as the owner of these files/directories.

~/.gnupg/ is a folder, the process reading the contents must be able to “enter” (=execute) this folder. This is the “x” Bit. It has the value 1. 7 - 6 = 1

Both ~/.gnupg/ and ~/.gnupg/* you want to be able to read and write, thats 4 + 2 = 6.

⇒ Only the owner of the files can read/write them now (=600). Only he can enter into the directory as well (=700)

⇒ These file rights don't “need” to be documented, they are derivable from the intended usage.

More info about permission notation: wikipedia.org - FS permissions

Sourced from superuser.com - Correct permissions for .gnupg file dir

Sign `Release` file automatically with passphrase:

```
gpg --yes --pinentry-mode loopback --passphrase-file /path/to/passphrase-file --default-key <ID> --clearsign -o /path/to/InRelease /path/to/Release
```

Sources

This guide was sourced from several pages:

- earthly.dev
- unix.stackexchange.com - Generating GPG Key Error
- unix.stackexchange.com - Generate Release file
- unix.stackexchange.com - Avoid prompts when signing file
- superuser.com - Correct permissions for .gnupg file dir
- medium.com/
- itsfoss.com

Other helpful commands

Create Packages file with `dpkg-scanpackages`: (run in base directory of repository e.g. `/srv/repo/`)

```
dpkg-scanpackages --arch amd64 pool/ > dists/stable/main/binary-amd64/Packages
```

Create compressed Packages file:

```
cat dists/stable/main/binary-amd64/Packages | gzip -9 > dists/stable/main/binary-amd64/Packages.gz
```

Generate Package files:

```
apt-ftparchive generate /path/to/aptgenerate.conf
```

Create Release file:

```
apt-ftparchive -c /root/apt-ftp-files/aptrelease.conf release /srv/ext-stor/repo/ > /srv/ext-stor/repo/dists/stable/Release
```

Signing Release file:

```
gpg --default-key <ID> -abs -o Release.gpg Release
```

Signing InRelease file:

```
gpg --default-key <ID> --clearsign -o InRelease Release
```

Apache config to hide .db files from apt-ftparchive, add this to site `repository.conf`:

```
<Files packages-amd64.db>
    Require all denied
</Files>
```

Automated file signing of password protected keys: unix.stackexchange.com - Sign files with gpg automatically with password-protected keys