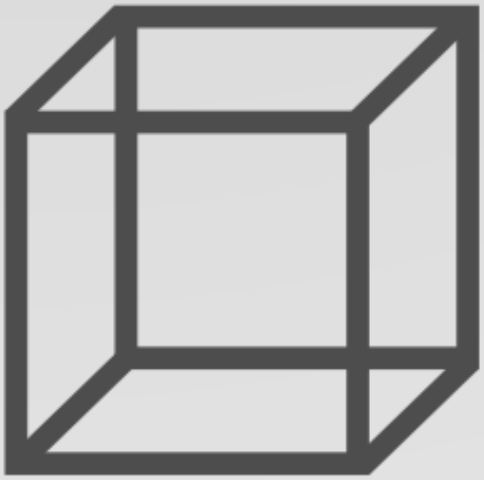


Chakra Project

Packaging Guide

Version 2013.07

Starting. Meet us!

Welcome and thanks for your interest!

http://chakra-project.org/

Chakra is a free, user-friendly and extremely powerful live and installable distribution based on the award winning KDE Software Compilation. Chakra is by default a GTK free distribution specially made for run Qt based applications and frameworks at full performance.

Index

Some Things About Chakra's Philosophy

Three reasons for Chakra focusing on being as pure a KDE distro as possible:

> True belief in the power of KDE. Like some distros believe in simply providing "foss" packages, Chakra believes it is time to provide as clean as possible, fully KDE focused environment. This means virtually no GTK-depended applications in the official repositories.

> Many of those "must have" GTK applications depend on a good part of a complete GNOME desktop; examples are apps such as GIMP or Filezilla. Why install 50-60 GNOME dependencies for only one or two apps?

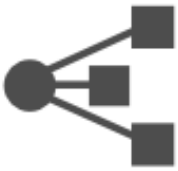
> KDE uses the /usr directory almost exclusively to install applications. This makes for a very transparent system. Many GTK apps are installed in quite a few different directories.

The target audience for Chakra is officially "KISS-minded users who aren't afraid to get their hands dirty." Although Chakra is very stable and easy to use, it occasionally requires a certain level of competency and does not shy away from the command-line. That being said, the simpleness and clarity in code and system architecture makes Chakra a great distro for users who want to learn the ins-and-outs of Linux. If you are a beginner and don't mind reading wiki's and forums and putting in a little effort, Chakra and its community is a great place to start!

Chakra follows a half-rolling release model. New Chakra images will only improve the system installation. If you already have it installed, you can upgrade all your software through Pacman (sudo pacman -Syu).

Arch was the origin of the project, but Chakra has matured into an independent distribution of its own. Chakra is still heavily influenced by the virtues of simplicity and elegance learned while with Arch but the underlying architecture is no longer compatible.

Chakra's Repositories



Main Repositories

[core]

Base of the system.

[platform]

Drivers, firmware, X.org and additional packages and dependencies related with desktop, apps, games and extra repos.

[desktop]

KDE Software Compilation packages, their dependencies and Chakra tools.

[apps]

Additional KDE/Qt applications.

[games]

Games and other gaming-related software.

[lib32]

x86_64 packages compiled against i686 libraries.

[extra]

An optional repo with a few essential applications GTK dependent.

[unstable]

An optional repo dedicated to developers, not to common use.

Packaging Standards

When building packages or build packages for Chakra, you should adhere to the package guidelines below, especially if you would like to contribute your new package or build package to Chakra.

File: PKGBUILD

```
pkgname=NAME
pkgver=VERSION
pkgrel=1
pkgdesc=""
arch=('x86_64')
url="http://ADDRESS/"
screenshot="http://address"
license=('GPL')
groups=()
depends=()
makedepends=()
optdepends=()
provides=()
conflicts=()
replaces=()
backup=()
options=()
hooks=()
install=
changelog=
source=($pkgname-$pkgver.tar.gz)
noextract=()
md5sums=() #generate with 'makepkg -g'

build() {
  cd $srcdir/$pkgname-$pkgver
  ./configure --prefix=/usr
  make
}

package() {
  cd $srcdir/$pkgname-$pkgver
  make DESTDIR=$pkgdir install
}
```

Between the body and the header there should be one line of separation, uncommented. The header content will depend on the place where the PKGBUILD will be used in official repos.

When a variable is not needed, remove its line from the PKGBUILD (provides, replaces, etc.). It is common practice to preserve the order of the PKGBUILD fields as shown above. Also, try to keep the line length in the PKGBUILD below ~100 characters.

Packaging Details

Package Name

Package names should consist of alphanumeric characters only; all letters should be lowercase. Development versions (alphas, betas, etc.) should be named like `$pkgname-dev`, and treated like an alternate version of the stable package.

Package Version

Package versions should be the same as the version released by the author. Versions can include letters if need be. Version tags may not include hyphens! Letters, numbers, and periods only. When original version number includes an hyphen, it's a common practice to replace it with an underscore. Example: `1.0-beta3` becomes `1.0_beta3`.

Package Release

Package releases are specific to Chakra packages. These allow users to differentiate between newer and older package builds. When a new package version is first released, the release count starts at `1`. Then as fixes and optimizations are made, the package will be re-released to the public and the release number will increment. When a new version comes out, the release count resets to `1`. Package release tags follow the same naming restrictions as version tags.

Package Description

When writing a description for a package, do not include the package name in a self-referencing way. For example, "Nedit is a text editor for X11" could be simplified to "Text editor for X11". Also try to keep the descriptions to ~80 characters or less.

Architectures

The arch array should contain 'x86_64' (Chakra is focused only in 64 bits). You can also use 'any' for architecture independent packages.

Licenses

The `license` should be always used. You can use it as follows:

> If the license is already in `/usr/share/licenses/common`, you can refer to it using it's directory name.

  Example: `license=('GPL3')`

> If the appropriate license is not included in the official licenses package, several things must be done:

  The license file(s) should be included in `/usr/share/licenses/$pkgname/`.

  If the source archive does not contain the license details and the license is only displayed on a website for example, then you need to copy the license to a file and include it. Remember to call it something appropriate too.

  Add `'custom'` to the licenses array. Optionally, you can replace `'custom'` with `'custom:<name of license>'`.

  Once a license is used in two or more packages in an official repository, it becomes common.

  The MIT, BSD, zlib/libpng and Python licenses are special cases and cannot be included in the 'common' licenses pkg. For the sake of the license variable, it's treated like a common license (license=('BSD'), license=('MIT'), license=('ZLIB') or license=('Python')) but for the sake of the file-system, it's a custom license, because each one has its own copyright line. Each MIT, BSD, zlib/libpng or Python licensed package should have its unique license stored in `/usr/share/licenses/$pkgname/`.

Some packages may not be covered by a single license. In these cases multiple entries may be made in the license array e.g. license=("GPL" "custom:some commercial license"). For the majority of packages these licenses apply in different cases, as opposed to applying at the same time.

The (L)GPL has many versions and permutations of those versions. For (L)GPL software, the convention is:
(L)GPL - (L)GPLv2 or any later version
(L)GPL2 - (L)GPL2 only
(L)GPL3 - (L)GPL3 or any later version

It's preferred to set license to the exact version when it applies. For example, you should better set the license to GPL2 instead of just GPL. This is because just writing GPL as license can be understood as "Not sure which version of the license".

Dependencies

Dependencies are the most common packaging error. Verify dependencies by looking at source documentation and the program website.

Optional Dependencies

Any optional dependencies that aren't needed to run the package or have it generally function shouldn't be included; instead the information should be added to the optdepends array like this:

```
optdepends=('package: functionality it provides'
            'package2: functionality it provides')
```

The optdepends information will automatically be printed out on installation or upgrade from Pacman.

Alternate Versions of Packages

If you just provide an alternate version of an already existing package, use conflicts array to reference to the original package, and any other alternate packages. Use also `provides=<original package name>` if that package is required by others.

Renaming packages

If you want to rename your package, change the name in the PKGBUILD and define the old one in the replaces array.

Important messages

All important messages should be echoed during install using an `.install` file. For example, if a package needs extra setup to work, directions should be included.

New Variables

In general, no new variables should be introduced in the `PKGBUILD` file. Nevertheless, there are some accepted variables which might be used in certain cases:

$_pkgname
When the `$pkgname` value is not the real name of the package (used i.e. in the source URLs), this alternative variable can be added right after the `$pkgname` and set to the real name. Example:

```
pkgname=rekonq-git
_pkgname=rekonq
```

$_pkgver
When the `$pkgver` value is different from the one used in source URLs or paths, this alternative variable can be added right after the `$pkgver`. Example:

```
pkgver=1.0_rc2
_pkgver=1.0-rc2
```

Any other new variable, which should be added only if absolutely required to build the package, has to be prefixed with an underscore (_). Example: `_customvariable=`.

Moving Files

When moving files from the `$srcdir` to the `$pkgdir`, the preferred way is to do it by using the install command. It's really easy to use, just have a look at any `PKGBUILD` using it and also run this:

```
install --help
```

Building Packages Without Compilation

With certain software there's no compilation process, just a packaging process (downloaded files are just moved to the right path). In those cases, `build()` function should be deleted, and all the process should be defined inside the `package()` function.

Directories

Configuration files should be placed in the `/etc` directory. If there's more than one configuration file, it's customary to use a subdirectory in order to keep the `/etc` area as clean as possible. Use `/etc/$pkgname/`, where `$pkgname` is the name of your package (or a suitable alternative, e.g., apache uses `/etc/httpd/`).

Package files should follow these general directory guidelines:

/etc

System-essential configuration files.

/etc/$pkgname

Configuration files for $pkgname.

/usr/bin

Application binaries.

/usr/include

Header files.

/usr/lib

Libraries.

/usr/lib/$pkgname

Modules, plug-ins, etc.

/usr/sbin

System binaries.

/usr/share/doc/$pkgname

Application documentation.

/usr/share/info

GNU Info system files.

/usr/share/man

Manpages.

/usr/share/$pkgname

Application data.

/var/lib/$pkgname

Persistent application storage.

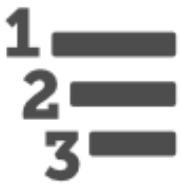
/opt/$pkgname

Large self-contained packages such as Java, etc

Package should not contain following directories:

```
/dev
/home
/srv
/media
/mnt
/proc
/root
/selinux
/sys
/tmp
/var/tmp
```

Repository-specific Standards



Chakra Official Repositories PKGBUILD Header

The PKGBUILD should have the following header:

File: PKGBUILD

```
#
# <Repo> Packages for Chakra, part of chakra-project.org
#
# Maintainer: <Maintainer contact information>

# Only for DESKTOP:
# include global config
source ../_buildscripts/${current_repo}-${_arch}-cfg.conf
```

<Repo>

This should be replaced according to the repository: Core, Platform, Desktop, Apps.

<Maintainer contact information>

Maintainer contact information should include a name and an e-mail address. These are some valid examples:

```
Complete Name <account[at]doma[dot]in>
Complete Name (Nick) <account[at]doma[dot]in>
```

If there were several maintainers, there would be one maintainer line per maintainer, and the difference between them should be explained this way:

```
# Maintainer (difference): Complete Name <account[at]doma[dot]in>
```

As an example:

```
# Maintainer (x86_64): Complete Name <account[at]doma[dot]in>
```

Right after maintainers lines, there should be a list of previous contributors.

Software-specific Standards

CMake PKGBUILDs

When using CMake, `DCMAKE_BUILD_TYPE=RelWithDebInfo` option should be added. This way debug information is included, so Chakra users can properly report bugs to help upstream developers to fix them.

This is an example of the build() and package() funtions for CMake-based PKGBUILDs:

```
build() {
  cd $srcdir/$pkgname-$pkgver
  [[ -d build ]] && rm -r build
  mkdir build && cd build
  cmake \
    -DCMAKE_INSTALL_PREFIX=/usr \
    -DCMAKE_BUILD_TYPE=RelWithDebInfo \
    ..
  make
}

package() {
  cd $srcdir/$pkgname-$pkgver/build
  make DESTDIR=$pkgdir install
}
```

IRC Channels

You can find us on the following IRC channels on the Freenode network (irc.freenode.net)

#chakra The Chakra Project general discussion channel
#chakra-support Our support channel
#chakra-devel Our development channel
#chakra-es Our spanish support and discussion channel
#chakra-fr Our french support and discussion channel
#chakra-gl Our galician support and discussion channel
#chakra-it Our italian support and discussion channel

License and Credits

Made by Malcer for the Chakra Project.
malcer.deviantart.com

Thanks to Manuel Tortosa and Adrián Chaves for the info help
(and the Chakra wiki, of course!)

And thanks to you, possible new Chakra packager.