Batocera Package Manager (pacman)

This feature is introduced with Batocera 5.27.

What is pacman?

Batocera now integrates a package manager, pacman, so that you can easily install/upgrade/remove your own packages, or packaged provided and hosted by the Batocera community.

Obviously, packages hosted by Batocera **will never include copyrighted material** – only freeware or shareware ROMs, homebrews, free games, themes, free musical themes and so on. It's a way to easily customize your Batocera to fit better with your needs.

The advantages of using a package manager are:

- one package contains all the files needed: ROM, scraped metadata, update in the gamelist.xml…
- you can install packages directly from the Internet
- you can update them when a new release of the package is available
- you can easily remove packages that you don't want any longer
- dependencies are automatically resolved

Pacman was chosen because it is lightweight, very portable, and with Batocera's version, you can package your own pacman packages on your Batocera box **without a full dev environment**. Batocera provides its specific tools (slimmed down from ArchLinux's full pacman distribution).

From a technical perspective, a pacman package is simply a `tar.zstd` or `.tar.xz` archive – exact same format as Batocera's `boot.tar.xz`. Newer versions of the packages use zstd rather than xz as the compression algorithm, as it's much faster to decompress, especially for larger packages.

Batocera-store

Batocera provides a full UI in EmulationStation to manage your pacman packages from **UPDATES & DOWNLOADS → CONTENT DOWNLOADER** and a series of command lines to help you install, update and remove packages through the `batocera-store` tool:

```
/usr/bin/batocera-store install <package>
/usr/bin/batocera-store remove  <package>
/usr/bin/batocera-store list
/usr/bin/batocera-store list-repositories
/usr/bin/batocera-store clean
/usr/bin/batocera-store clean-all
/usr/bin/batocera-store refresh
/usr/bin/batocera-store update
```

The first two commands are self-explanatory; `list` lists all the packages you can install, and `list-repositories` lists all the repositories from where packages can be installed.

Whenever a package is downloaded and installed, pacman keeps a local cache of the package into `/userdata/system/pacman/pkg/`. The command `clean` will clean up this repository and keep only the last version installed, while `clean-all` will remove all files from the cache.

Finally `refresh` will refresh the packages available from all the repositories, and `update` will update all your installed packages to their latest version.

How can I use pacman as a user?

Users usually need only `batocera-store` commands. But here are some details on how it works under the hood.

The first thing to do is to synchronize your local pacman DB with Batocera's repository (kind of an “app store” for packages).

```
 # pacman -Sy
 :: Synchronizing package databases...
 batocera                                768.0   B  0.00   B/s 00:00 [----------
-----------------------------------] 100%
```

Then you can search for all packages available

```
 # pacman -Ss
 batocera/bezels-default-glazed 1.0.0-1
     Batocera bezels with a CRT effect
 batocera/nes-alter-ego 1.0.0-1
     NES freeware puzzle game - manual provided as a pdf
 batocera/ports-quake-shareware 1.0.0-1
     Quake game files (shareware version)
```

You can search from string, like `pacman -Ss bezels`.

In order to install/upgrade a package you can type:

```
 # pacman -S nes-alter-ego
 resolving dependencies...
 looking for conflicting packages...
 Packages (1) nes-alter-ego-1.0.0-1
 Total Download Size:   0.40 MiB
 Total Installed Size:  0.49 MiB
 :: Proceed with installation? [Y/n]
 (1/1) checking keys in keyring
[-------------------------------------------] 100%
 (1/1) checking package integrity
[-------------------------------------------] 100%
 (1/1) loading package files
[-------------------------------------------] 100%
 (1/1) checking for file conflicts
[-------------------------------------------] 100%
 (1/1) checking available disk space
```

```
[-----------------------------------------------] 100%
 (1/1) reinstalling nes-alter-ego
[-----------------------------------------------] 100%
 :: Running post-transaction hooks...
 (1/1) batocera-install.hook
 Entry for Alter_Ego.nes added in /userdata/roms/nes/gamelist.xml
```

You game will be added to EmulationStation, and you can play this newly installed game!

To remove a package:

```
 # pacman -R nes-alter-ego
 checking dependencies...
 Packages (1) nes-alter-ego-1.0.0-1
 Total Removed Size:  0.47 MiB
 :: Do you want to remove these packages? [Y/n]
 :: Processing package changes...
 (1/1) removing nes-alter-ego                                  [------------------
----------------------------------------------------------] 100%
```

How can I create my own pacman packages?

It's **very easy** and you don't need real development skills - basic scripting is more than enough!

For this example, we'll use the NES free homebrew ROM Alter Ego available from https://www.romhacking.net/homebrew/1/

First, create a new directory where replicate the directories of the final installed package, once it is installed on a Batocera box (relative to the root directory /). Put all the directories and files needed, and an two additional description files for the package called `.PKGINFO` and `.BATOEXEC`.

```
 # find ./
 ./
 ./userdata
 ./userdata/roms
 ./userdata/roms/nes
 ./userdata/roms/nes/Alter_Ego.nes
 ./userdata/roms/nes/images
 ./userdata/roms/nes/images/Alter_Ego.png
 ./userdata/roms/nes/manuals
 ./userdata/roms/nes/manuals/Alter_Ego-manual.pdf
 ./userdata/roms/nes/videos
 ./userdata/roms/nes/videos/Alter_Ego-video.mp4
 ./.PKGINFO
 ./.BATOEXEC
```

Here is the content of such a `.PKGINFO` - the only mandatory lines are the **first 5 lines** in the following example (**pkgname**, **pkgver**, **pkgdesc**, **arch** and **group**), the rest will be taken care of by the script `batocera-makepkg` that will actually generate the package.

```
# cat .PKGINFO
pkgname = nes-alter-ego
pkgver = 1.0.0-1
pkgdesc = NES freeware puzzle game - manual provided as Alter_Ego.pdf
arch = any
group = sys-nes
packager = your_name
url = http://where_is_this_coming.from/
```

group: this will define how your package will be presented in the content downloader. If you are creating a package for a system, you need to prefix it with `sys-` before the name of the system, as seen by Batocera. Like `sys-megadrive` for Megadrive/Genesis. If you package a bezel, selection of music tunes or any other type of content, don't use the sys- prefix. When you browse the content downloader, only systems compatible with your architecture will show up. For example, you won't see wine packages for Windows games if you are not on a x86 or x86_64 PC.

The other file is `.BATOEXEC` and it gives you the ability add metadata or execute commands. Typically, for a package that adds a new ROM, you want to add it into the `gamelist.xml`. It's syntax is very simple: the first line defines the action you want to do, and the rest of the file describes what is to be done. To add a <game> node in gamelist.xml, here is a typical `.BATOEXEC` file:

```
 # cat .BATOEXEC
 gamelist = nes
 <game>
   <path>./Alter_Ego.nes</path>
   <name>Alter Ego</name>
   <desc>Freeware puzzle game for NES, a port of the original (by Denis
Grachev for the ZX Spectrum). Swap positions with your 'alter ego' to move
about the level and obtain all the bouncing pixels.</desc>
   <rating>0.6</rating>
   <releasedate>20110827T000000</releasedate>
   <developer>Shiru</developer>
   <publisher></publisher>
   <genre>Puzzle</genre>
   <players>1</players>
   <image>./images/Alter_Ego.png</image>
   <manual>./manuals/Alter_Ego-manual.pdf</manual>
   <video>./videos/Alter_Ego-video.mp4</video>
 </game>
```

We won't use the full makepkg package from ArchLinux, as it requires Perl and other dependencies that are not available on Batocera. I wrote a mini-version of it, `batocera-makepkg` that is only depending on Bash and Buildroot/Busybox for Batocera:

```
 # batocera-makepkg
 Creating package ../nes-alter-ego-1.0.0-1-any.pkg.tar.xz ...
 SUCCESS: package ../nes-alter-ego-1.0.0-1-any.pkg.tar.xz correctly
generated
```

You pacman package has been generated in the upper directory. The script won't overwrite an existing package (use new revisions of the package like `1.0.0-2` if needed - or, of course, you can

also remove the previous package before recreating it).

BATOEXEC files

Do you remember back in the MS-DOS days when you had to write an AUTOEXEC.BAT file to describe what to do? That's the inspiration for the name `.BATOEXEC`: it describes what you can do when a package is installed/upgraded or removed.

The first line describes the action you want to do. Currently, BATOEXEC accepts two commands: `gamelist` and `exec`.

gamelist

On the first line, you tell which **system** will you be adding a segment in its `gamelist.xml`, and then starting from the second line, you can put the <game> subtree that will be inserted there.

Example:

```
 # cat .BATOEXEC
 gamelist = nes
 <game>
   <path>./Alter_Ego.nes</path>
   <name>Alter Ego</name>
   <desc>Freeware puzzle game for NES, a port of the original (by Denis
Grachev for the ZX Spectrum). Swap positions with your 'alter ego' to move
about the level and obtain all the bouncing pixels.</desc>
   <rating>0.6</rating>
   <releasedate>20110827T000000</releasedate>
   <developer>Shiru</developer>
   <publisher></publisher>
   <genre>Puzzle</genre>
   <players>1</players>
   <image>./images/Alter_Ego.png</image>
   <manual>./manuals/Alter_Ego-manual.pdf</manual>
   <video>./videos/Alter_Ego-video.mp4</video>
 </game>
```

exec

On the first line, you tell which command interpreter you want to use, and then the commands you want to execute. Only interpreters that takes a `-c` argument for commands are available. You can use Bash or Python for example, those are tools heavily used by Batocera.

If you define sections between `.INSTALL_START` and `.INSTALL_END` or `.UNINSTALL_START` and `.UNINSTALL_END`, those commands will be executed only on install/upgrade or removal of the package. Commands outside those sections will be executed in any case (ex: `date` in the example below)

```
# cat .BATOEXEC
exec = /bin/bash
date
.INSTALL_START
echo "installing bezels with glazed effect..."
.INSTALL_END
.UNINSTALL_START
echo "Uninstalling bezels with glazed effect..."
.UNINSTALL_END
```

A `.BATOEXEC` file is not mandatory to use `batocera-makepkg` (if you don't have to do anything specific... for example if you package an EmulationStation theme, or if you package mod tracker musics...).

Install, remove and manage packages

The easiest is to install a package from a repository. For example, to install `nes-alter-ego` from the Batocera repository, just enter:

```
pacman -S nes-alter-ego
```

It is not necessary to setup a repository to test your own packages (see below about setting up your own repositories). You can simply install or upgrade a local package in .tar.xz format by typing:

```
pacman -U mypackage-1.0.0-1.tar.xz
```

To remove a package, whether it's been installed from a repository or from an individual package, you need to use the `-R` switch, without providing the version of the package that is intalled. For example:

```
pacman -R mypackage
```

Also, you can see that every package that gets installed (whether from an individual package or from a repository) is copied locally to `/userdata/system/pacman/pkg/` on your Batocera system. This local cache directory of packages can be cleaned up, i.e. remove all the package files, by entering:

```
pacman -Scc
```

Manage your own repository

You can use pacman to manage your own library of packages (if you want to add your own ROMs to the free ones distributed by the Batocera game store for example).

In order to do so, you can create a local repository by adding a new section in `/userdata/system/pacman/pacman.conf`:

```
[private_repo]
Server            = file:///userdata/local_repo/
```

Server can be a local directory like `/userdata/local_repo/` or you can host that on your own web server.

To add a package to your local repository, copy your new package `mypackage-1.0.0-1.tar.xz` to where your local repository stands and type:

```
 # repo-add /userdata/local_repo/private_repo.db.tar.gz mypackage-1.0.0-1-
any.pkg.tar.xz
 ==> Extracting private_repo.db.tar.gz to a temporary location...
 ==> Extracting private_repo.files.tar.gz to a temporary location...
 ==> Adding package 'mypackage-1.0.0-1-any.pkg.tar.xz'
   -> Computing checksums...
   -> Creating 'desc' db entry...
   -> Creating 'files' db entry...
 ==> Creating updated database file
'/userdata/local_repo/private_repo.db.tar.gz'
```

Once the package is added to your repo, synchronize your pacman with the latest information by typing `pacman -Sy` and you can now install your package from your repository.

To remove a package from your repository, you can use `repo-remove /userdata/local_repo/private_repo.db.tar.gz mypackage` (it's the name of the package, not the file name with the version).