pacman

The **pacman (https://pacman.archlinux.page/) package manager** is one of the major distinguishing features of **Arch Linux**. It combines a simple binary package format with an easy-to-use **Arch build system**. The goal of *pacman* is to make it possible to easily manage packages, whether they are from the **official repositories** or the user's own builds.

Pacman keeps the system up-to-date by synchronizing package lists with the master server. This server/client model also allows the user to download/install packages with a simple command, complete with all required dependencies.

Pacman is written in the **C** programming language and uses the **bsdtar(1) (https://man.archlinux.org/man/bsdtar.1)** **tar** format for packaging.

> **Tip**
>
> The **pacman (https://archlinux.org/packages/?name=pacman)** package contains tools such as **makepkg** and **vercmp(8) (https://man.archlinux.org/man/vercmp.8)**. Other useful tools such as **pactree** and **checkupdates** are found in **pacman-contrib (https://archlinux.org/packages/?name=pacman-contrib)** (**formerly (https://gitlab.archlinux.org/pacman/pacman/commit/0c99eabd50752310f42ec808c8734a338122ec86)** part of *pacman*). Run
> ```
> pacman -Ql pacman pacman-contrib | grep -E 'bin/.+'
> ```
> to see the full list.

Related articles

- /Package signing
- /Pacnew and Pacsave
- /Restore local database
- /Rosetta
- /Tips and tricks
- Arch User Repository
- Creating packages
- Downgrading packages
- namcap
- FAQ#Package management
- System maintenance

1 Usage

What follows is just a small sample of the operations that *pacman* can perform. To read more examples, refer to **pacman(8) (https://man.archlinux.org/man/pacman.8)**.

> **Tip**
>
> For those who have used other Linux distributions before, there is a helpful **Pacman/Rosetta** article.

1.1 Installing packages

A package is an archive containing:

- all of the (compiled) files of an application

- metadata about the application, such as application name, version, dependencies, etc.
- installation files and directives for *pacman*

Arch's package manager *pacman* can install, update, and remove those packages. Using packages instead of compiling and installing programs yourself has various benefits:

- easily updatable: *pacman* will update existing packages as soon as updates are available
- dependency checks: *pacman* handles dependencies for you, you only need to specify the program and *pacman* installs it together with every other program it needs
- clean removal: *pacman* has a list of every file in a package; this way, no files are unintentionally left behind when you decide to remove a package.

Note

- Packages often have **optional dependencies** which are packages that provide additional functionality to the application but not strictly required for running it. When installing a package, *pacman* will list a package's optional dependencies, but they will not be found in `pacman.log`. Use the **#Querying package databases** command to view the optional dependencies of a package.
- When installing a package which you require only as a (optional) dependency of some other package (i.e. not required by you explicitly), it is recommended to use the `--asdeps` option. For details, see the **#Installation reason** section.

Warning

When installing packages in Arch, avoid refreshing the package list without **upgrading the system** (for example, when a **package is no longer found** in the official repositories). In practice, do **not** run `pacman -Sy package_name` instead of `pacman -Syu package_name`, as this could lead to dependency issues. See **System maintenance#Partial upgrades are unsupported** and **BBS#89328 (https://bbs.archlinux.org/viewtopic.php?id=89328)**.

1.1.1 Installing specific packages

To install a single package or list of packages, including dependencies, issue the following command:

```
# pacman -S package_name1 package_name2 ...
```

To install a list of packages with regex (see **this forum thread (https://bbs.archlinux.org/viewtopic.php?id=7179)**):

```
# pacman -S $(pacman -Ssq package_regex)
```

Sometimes there are multiple versions of a package in different repositories (e.g. *extra* and *extra-testing*). To install the version from the *extra* repository in this example, the repository needs to be defined in front of the package name:

```
# pacman -S extra/package_name
```

To install a number of packages sharing similar patterns in their names, one can use curly brace expansion. For example:

```
# pacman -S plasma-{desktop,mediacenter,nm}
```

This can be expanded to however many levels needed:

```
# pacman -S plasma-{workspace{,-wallpapers},pa}
```

1.1.1.1 Virtual packages

A virtual package is a special package which does not exist by itself, but is **provided** by one or more other packages. Virtual packages allow other packages to not name a specific package as a dependency, in case there are several candidates. Virtual packages cannot be installed by their name, instead they become installed in your system when you have installed a package *providing* the virtual package. An example is the **dbus-units** package.

> **Tip**
>
> When there are multiple candidates, the list of choices presented will sort first by **repositories** in the order they appear in `pacman.conf`, then alphabetically when multiple results exist from the same repository.

1.1.2 Installing package groups

Some packages belong to a **group of packages** that can all be installed simultaneously. For example, issuing the command:

```
# pacman -S gnome
```

will prompt you to select the packages from the **gnome (https://archlinux.org/groups/x86_64/gnome/)** group that you wish to install.

Sometimes a package group will contain a large amount of packages, and there may be only a few that you do or do not want to install. Instead of having to enter all the numbers except the ones you do not want, it is sometimes more convenient to select or exclude packages or ranges of packages with the following syntax:

```
Enter a selection (default=all): 1-10 15
```

which will select packages 1 through 10 and 15 for installation, or:

```
Enter a selection (default=all): ^5-8 ^2
```

which will select all packages except 5 through 8 and 2 for installation.

To see what packages belong to the gnome group, run:

```
$ pacman -Sg gnome
```

Also visit **https://archlinux.org/groups/** to see what package groups are available.

> **Note**
>
> If a package in the list is already installed on the system, it will be reinstalled even if it is already up-to-date. This behavior can be overridden with the `--needed` option.

1.2 Removing packages

To remove a single package, leaving all of its dependencies installed:

```
# pacman -R package_name
```

To remove a package and its dependencies which are not required by any other installed package:

```
# pacman -Rs package_name
```

> **Warning**
>
> When removing a group, such as *gnome*, this ignores the install reason of the packages in the group, because it acts as though each package in the group is listed separately. Install reason of dependencies is still respected.

The above may sometimes refuse to run when removing a group which contains otherwise needed packages. In this case try:

```
# pacman -Rsu package_name
```

To remove a package, its dependencies and all the packages that depend on the target package:

> **Warning**
>
> This operation is recursive, and must be used with care since it can remove many potentially needed packages.

```
# pacman -Rsc package_name
```

To remove a package, which is required by another package, without removing the dependent package:

> **Warning**
>
> The following operation can break a system and should be avoided. See **System maintenance#Avoid certain pacman commands**.

```
# pacman -Rdd package_name
```

Pacman saves important configuration files when removing certain applications and names them with the extension: *.pacsave*. To prevent the creation of these backup files use the `-n` option:

```
# pacman -Rn package_name
```

Note

Pacman will not remove configurations that the application itself creates (for example "dotfiles" in the home directory).

1.3 Upgrading packages

Warning

- Users are expected to follow the guidance in the **System maintenance#Upgrading the system** section to upgrade their systems regularly and not blindly run the following command.
- Arch only supports full system upgrades. See **System maintenance#Partial upgrades are unsupported** and **#Installing packages** for details.

Pacman can update all packages on the system with just one command. This could take quite a while depending on how up-to-date the system is. The following command synchronizes the repository databases *and* updates the system's packages, excluding "local" packages that are not in the configured repositories:

```
# pacman -Syu
```

1.4 Querying package databases

Pacman queries the local package database with the `-Q` flag, the sync database with the `-S` flag and the files database with the `-F` flag. See `pacman -Q --help`, `pacman -S --help` and `pacman -F --help` for the respective suboptions of each flag.

Note

Sync the files database before querying it to get up-to-date results:

```
# pacman -Fy
```

Tip

You can **enable/start** the `pacman-filesdb-refresh.timer` (provided within the **pacman-contrib (https://archlinux.org/packages/?name=pacman-contrib)** package) to

> refresh *pacman* files database weekly.

Pacman can search for packages in the database, searching both in packages' names and descriptions:

```
$ pacman -Ss string1 string2 ...
```

Sometimes, `-s` 's builtin ERE (Extended Regular Expressions) can cause a lot of unwanted results, so it has to be limited to match the package name only; not the description nor any other field:

```
$ pacman -Ss '^vim-'
```

To search for already installed packages:

```
$ pacman -Qs string1 string2 ...
```

To search for package file names in remote packages:

```
$ pacman -F string1 string2 ...
```

To display extensive information about a given package (e.g. its dependencies):

```
$ pacman -Si package_name
```

For locally installed packages:

```
$ pacman -Qi package_name
```

Passing two `-i` flags will also display the list of backup files and their modification states:

```
$ pacman -Qii package_name
```

To retrieve a list of the files installed by a package:

```
$ pacman -Ql package_name
```

To retrieve a list of the files installed by a remote package:

```
$ pacman -Fl package_name
```

To verify the presence of the files installed by a package:

```
$ pacman -Qk package_name
```

Passing the `k` flag twice will perform a more thorough check.

To query the database to know which package a file in the file system belongs to:

```
$ pacman -Qo /path/to/file_name
```

To query the database to know which remote package a file belongs to:

```
$ pacman -F /path/to/file_name
```

To list all packages no longer required as dependencies (orphans):

```
$ pacman -Qdt
```

To list all packages explicitly installed and not required as dependencies:

```
$ pacman -Qet
```

See **pacman/Tips and tricks** for more examples.

For advanced functionality, install **pkgfile**, which uses a separate database with all files and their associated packages.

1.4.1 Pactree

> **Note**
>
> `pactree(8)` (https://man.archlinux.org/man/pactree.8) is not part of the `pacman` (https://archlinux.org/packages/?name=pacman) package anymore. Instead it can be found in `pacman-contrib` (https://archlinux.org/packages/?name=pacman-contrib).

To view the dependency tree of a package:

```
$ pactree package_name
```

To view the dependent tree of a package, pass the reverse flag `-r` to *pactree*.

1.4.2 Database structure

The *pacman* databases are normally located at `/var/lib/pacman/sync`. For each repository specified in `/etc/pacman.conf`, there will be a corresponding database file located there. Database files are gzipped tar archives containing one directory for each package, for example for the `which` (https://archlinux.org/packages/?name=which) package:

```
$ tree which-2.21-5
```

```
which-2.21-5
|-- desc
```

The `desc` file contains meta data such as the package description, dependencies, file size and MD5 hash.

1.5 Cleaning the package cache

Pacman stores its downloaded packages in `/var/cache/pacman/pkg/` and does not remove the old or uninstalled versions automatically. This has some advantages:

1. It allows to **downgrade** a package without the need to retrieve the previous version through other means, such as the **Arch Linux Archive**.
2. A package that has been uninstalled can easily be reinstalled directly from the cache directory, not requiring a new download from the repository.

However, it is necessary to deliberately clean up the cache periodically to prevent the directory to grow indefinitely in size.

The **paccache(8) (https://man.archlinux.org/man/paccache.8)** script, provided within the **pacman-contrib (https://archlinux.org/packages/?name=pacman-contrib)** package, deletes all cached versions of installed and uninstalled packages, except for the most recent three, by default:

```
# paccache -r
```

Enable and **start** `paccache.timer` to discard unused packages weekly. You can configure the arguments for the service in `/etc/conf.d/pacman-contrib`, e.g with `PACCACHE_ARGS='-k1'` or `PACCACHE_ARGS='-uk0'` for the two examples below.

> **Tip**
>
> You can create a **hook** to run this automatically after every *pacman* transaction, **install** **paccache-hook (https://aur.archlinux.org/packages/paccache-hook/)**[AUR] and see **other examples (https://bbs.archlinux.org/viewtopic.php?pid=1694743#p1694743)**

You can also define how many recent versions you want to keep. To retain only one past version use:

```
# paccache -rk1
```

Add the `-u` / `--uninstalled` switch to limit the action of *paccache* to uninstalled packages. For example to remove all cached versions of uninstalled packages, use the following:

```
# paccache -ruk0
```

See `paccache -h` for more options.

Pacman also has some built-in options to clean the cache and the leftover database files from repositories which are no longer listed in the configuration file `/etc/pacman.conf`. However *pacman* does not offer the possibility to keep a number of past versions and is therefore more aggressive than *paccache* default options.

To remove all the cached packages that are not currently installed, and the unused sync databases, execute:

```
# pacman -Sc
```

To remove all files from the cache, use the clean switch twice, this is the most aggressive approach and will leave nothing in the cache directory:

```
# pacman -Scc
```

> **Warning**
>
> One should avoid deleting from the cache all past versions of installed packages and all uninstalled packages unless one desperately needs to free some disk space. This will prevent downgrading or reinstalling packages without downloading them again.

pkgcacheclean (https://aur.archlinux.org/packages/pkgcacheclean/)[AUR] and **pacleaner (https://aur.archlinux.org/packages/pacleaner/)**[AUR] are two further alternatives to clean the cache.

1.6 Additional commands

Download a package without installing it:

```
# pacman -Sw package_name
```

Install a 'local' package that is not from a remote repository (e.g. the package is from the **AUR**):

```
# pacman -U /path/to/package/package_name-version.pkg.tar.zst
```

To keep a copy of the local package in *pacman*'s cache, use:

```
# pacman -U file:///path/to/package/package_name-version.pkg.tar.zst
```

Install a 'remote' package (not from a repository stated in *pacman*'s configuration files):

```
# pacman -U http://www.example.com/repo/example.pkg.tar.zst
```

1.6.1 dry run

Pacman **always** lists packages to be installed or removed, and asks for permission before taking any action.

To get a list in a processable format, and to prevent the actions of `-S`, `-U` and `-R`, you can use `-p`, short for `--print`.

`--print-format` can be added to format this list in various ways. `--print-format %n` will return a list without package versions.

1.7 Installation reason

The *pacman* database organizes installed packages into two groups, according to installation reason:

- **explicitly-installed**: packages that were literally passed to a generic *pacman* `-S` or `-U` command;
- **dependencies**: packages that, despite never (in general) having been passed to a *pacman* installation command, were *implicitly* installed because they were **required** by packages explicitly installed.

When installing a package, it is possible to force its installation reason to *dependency* with:

```
# pacman -S --asdeps package_name
```

The command is normally used because explicitly-installed packages may offer **optional packages**, usually for non-essential features for which the user has discretion.

> **Tip**
>
> Installing optional dependencies with `--asdeps` will ensure that, if you **remove orphans**, *pacman* will also remove optional packages set this way.

When **re**installing a package, though, the current installation reason is preserved by default.

The list of explicitly-installed packages can be shown with `pacman -Qe`, while the complementary list of dependencies can be shown with `pacman -Qd`.

To change the installation reason of an already installed package, execute:

```
# pacman -D --asdeps package_name
```

Use `--asexplicit` to do the opposite operation.

> **Note**
>
> Using `--asdeps` and `--asexplicit` options when upgrading, such as with `pacman -Syu package_name --asdeps`, is discouraged. This would change the installation reason of not only the package being installed, but also the packages being upgraded.

1.8 What happens during package install/upgrade/removal

When successful, the workflow of a transaction follows five high-level steps plus pre/post transaction hooks:

1. Initialize the transaction if there is not a database lock.
2. Choose which packages will be added or removed in the transaction.
3. Prepare the transaction, based on flags, by performing sanity checks on the sync databases, packages, and their dependencies.
4. Commit the transaction:
    1. When applicable, download packages (`_alpm_sync_load`).
    2. If pre-existing *pacman* `PreTransaction` hooks apply, they are executed.
    3. Packages are removed that are to-be-replaced, conflicting, or explicitly targeted to be removed.

4. If there are packages to add, then each package is committed:
    1. If the package has an install script, its `pre_install` function is executed (or `pre_upgrade` or `pre_remove` in the case of an upgraded or removed package).
    2. *Pacman* deletes all the files from a pre-existing version of the package (in the case of an upgraded or removed package). However, files that were marked as configuration files in the package are kept (see **/Pacnew and Pacsave**).
    3. *Pacman* untars the package and dumps its files into the file system (in the case of an installed or upgraded package). Files that would overwrite kept, and manually modified, configuration files (see previous step), are stored with a new name (*.pacnew*).
    4. If the package has an install script, its `post_install` function is executed (or `post_upgrade` or `post_remove` in the case of an upgraded or removed package).
5. If *pacman* `PostTransaction` hooks that exist at the end of the transaction apply, they are executed.
5. Release the transaction and transaction resource (i.e. database lock).

2 Configuration

Pacman settings are located in `/etc/pacman.conf`: this is the place where the user configures the program to work in the desired manner. In-depth information about the configuration file can be found in **pacman.conf(5) (https://man.archlinux.org/man/pacman.conf.5)**.

2.1 General options

General options are in the `[options]` section. Read **pacman.conf(5) (https://man.archlinux.org/man/pacman.conf.5)** or look in the default `pacman.conf` for information on what can be done here.

2.1.1 Comparing versions before updating

To see old and new versions of available packages, uncomment the "VerbosePkgLists" line in `/etc/pacman.conf`. The output of `pacman -Syu` will be like this:

```
Package (6)             Old Version  New Version  Net Change  Download Size

extra/libmariadbclient  10.1.9-4     10.1.10-1      0.03 MiB       4.35 MiB
extra/libpng            1.6.19-1     1.6.20-1       0.00 MiB       0.23 MiB
extra/mariadb           10.1.9-4     10.1.10-1      0.26 MiB      13.80 MiB
```

2.1.2 Parallel downloads

The number of packages being downloaded in parallel (at the same time) are configured in `/etc/pacman.conf` with the `ParallelDownloads` option under `[options]`. The `/etc/pacman.conf` shipped with the **pacman (https://archlinux.org/packages/?name=pacman)** package sets it to `5`. If the option is unset, packages will be downloaded sequentially.

2.1.3 Skip package from being upgraded

> **Warning**
>
> Be careful in skipping packages, since **partial upgrades** are unsupported.

To have a specific package skipped when **upgrading** the system, add this line in the `[options]` section:

```
IgnorePkg=linux
```

For multiple packages use a space-separated list, or use additional `IgnorePkg` lines. Also, **glob** patterns can be used. If you want to skip packages just once, you can also use the `--ignore` option on the command-line - this time with a comma-separated list.

It will still be possible to upgrade the ignored packages using `pacman -S` : in this case *pacman* will remind you that the packages have been included in an `IgnorePkg` statement.

2.1.4 Skip package group from being upgraded

> **Warning**
>
> Be careful in skipping package groups, since **partial upgrades** are unsupported.

As with packages, skipping a whole package group is also possible:

```
IgnoreGroup=gnome
```

2.1.5 Skip file from being upgraded

All files listed with a `NoUpgrade` directive will never be touched during a package install/upgrade, and the new files will be installed with a *.pacnew* extension.

```
NoUpgrade=path/to/file
```

Multiple files can be specified like this:

```
NoUpgrade=path/to/file1 path/to/file2
```

> **Note**
>
> The path refers to files in the package archive. Therefore, do not include the leading slash.

2.1.6 Skip files from being installed to system

To always skip installation of specific files or directories list them under `NoExtract` . For example, to avoid installing bash completion scripts, use:

```
NoExtract=usr/share/bash-completion/completions/*
```

Later rules override previous ones, and you can negate a rule by prepending `!`.

2.1.7 Maintain several configuration files

If you have several configuration files (e.g. main configuration and configuration with **testing** repository enabled) and would have to share options between configurations you may use `Include` option declared in the configuration files, e.g.:

```
Include = /path/to/common/settings
```

where */path/to/common/settings* file contains the same options for both configurations.

2.1.8 Hooks

Pacman can run pre- and post-transaction hooks from the `/usr/share/libalpm/hooks/` directory; more directories can be specified with the `HookDir` option in `pacman.conf`, which defaults to `/etc/pacman.d/hooks`. Hook file names must be suffixed with *.hook*. Pacman hooks are not interactive.

Pacman hooks are used, for example, in combination with `systemd-sysusers` and `systemd-tmpfiles` to automatically create system users and files during the installation of packages. For example, **tomcat8 (https://archlinux.org/packages/?name=tomcat8)** specifies that it wants a system user called `tomcat8` and certain directories owned by this user. The *pacman* hooks `systemd-sysusers.hook` and `systemd-tmpfiles.hook` invoke `systemd-sysusers` and `systemd-tmpfiles` when *pacman* determines that **tomcat8 (https://archlinux.org/packages/?name=tomcat8)** contains files specifying users and tmp files.

For more information on alpm hooks, see **alpm-hooks(5) (https://man.archlinux.org/man/alpm-hooks.5)**.

2.2 Repositories and mirrors

Besides the special **[options]** section, each other `[section]` in `pacman.conf` defines a package repository to be used. A *repository* is a *logical* collection of packages, which are *physically* stored on one or more servers: for this reason each server is called a *mirror* for the repository.

Repositories are distinguished between **official** and **unofficial**. The order of repositories in the configuration file matters; repositories listed first will take precedence over those listed later in the file when packages in two repositories have identical names, regardless of version number. In order to use a repository after adding it, you will need to **upgrade** the whole system first.

Each repository section allows defining the list of its mirrors directly or in a dedicated external file through the `Include` directive; for example, the mirrors for the official repositories are included from `/etc/pacman.d/mirrorlist`. See the **Mirrors** article for mirror configuration.

2.2.1 Package cache directory

Pacman stores downloaded package files in cache, in a directory denoted by `CacheDir` in **[options]** section of `pacman.conf` (defaults to `/var/cache/pacman/pkg/` if not set).

Cache directory may grow over time, even if keeping just the freshest versions of installed packages.

If you want to move that directory to some more convenient place, do one of the following:

- Set the `CacheDir` option in `pacman.conf` to new directory. Remember to retain the trailing slash. **This is the recommended solution**.
- Mount a dedicated partition or e.g. **Btrfs subvolume** in `/var/cache/pacman/pkg/` .
- Bind-mount selected directory in `/var/cache/pacman/pkg/` .

> **Warning**
>
> **Do not symlink** the `/var/cache/pacman/pkg/` directory to some other location. It **will** cause pacman to misbehave, especially when pacman attempts to update itself.

2.2.2 Package security

Pacman supports package signatures, which add an extra layer of security to the packages. The default configuration, `SigLevel = Required DatabaseOptional` , enables signature verification for all the packages on a global level. This can be overridden by per-repository `SigLevel` lines. For more details on package signing and signature verification, take a look at **pacman-key**.

3 Troubleshooting

3.1 "Failed to commit transaction (conflicting files)" error

If you see the following error: **[1] (https://bbs.archlinux.org/viewtopic.php?id=56373)**

```
error: could not prepare transaction
error: failed to commit transaction (conflicting files)
package: /path/to/file exists in filesystem
Errors occurred, no packages were upgraded.
```

This is happening because *pacman* has detected a file conflict, and by design, will not overwrite files for you. This is by design, not a flaw.

The problem is usually trivial to solve (although to be sure, you should try to find out how these files got there in the first place). A safe way is to first check if another package owns the file (`pacman -Qo /path/to/file`). If the file is owned by another package, **file a bug report**. If the file is not owned by another package, rename the file which "exists in filesystem" and re-issue the update command. If all goes well, the file may then be removed.

If you had installed a program manually without using *pacman*, for example through `make install`, you have to remove/uninstall this program with all of its files. See also **Pacman tips#Identify files not owned by any package**.

Every installed package provides a `/var/lib/pacman/local/package-version/files` file that contains metadata about this package. If this file gets corrupted, is empty or goes missing, it results in `file exists in filesystem` errors when trying to update the package. Such an error usually concerns only one package. Instead of manually renaming and later removing all the files that belong to the package in question, you may explicitly run `pacman -S --overwrite glob package` to force *pacman* to overwrite files that match `glob`.

> **Warning**
>
> Generally avoid using the `--overwrite` switch. See **System maintenance#Avoid certain pacman commands**.

3.2 "Failed to commit transaction (invalid or corrupted package)" error

Look for *.part* files (partially downloaded packages) in `/var/cache/pacman/pkg/` and remove them (often caused by usage of a custom `XferCommand` in `pacman.conf`).

```
# find /var/cache/pacman/pkg/ -iname "*.part" -delete
```

That same error may also appear if *archlinux-keyring* is out-of-date, preventing *pacman* from verifying signatures. See **Pacman/Package signing#Upgrade system regularly** for the fix and how to avoid it in the future.

3.3 "Failed to init transaction (unable to lock database)" error

When *pacman* is about to alter the package database, for example installing a package, it creates a lock file at `/var/lib/pacman/db.lck`. This prevents another instance of *pacman* from trying to alter the package database at the same time.

If *pacman* is interrupted while changing the database, this stale lock file can remain. If you are certain that no instances of *pacman* are running then delete the lock file:

```
# rm /var/lib/pacman/db.lck
```

> **Tip**
>
> You can run `fuser /var/lib/pacman/db.lck` as root to verify if there is any process still using it.

3.4 Packages cannot be retrieved on installation

This error manifests as `Not found in sync db`, `Target not found` or `Failed retrieving file`.

Firstly, ensure the package actually exists. If certain the package exists, your package list may be out-of-date. Try running `pacman -Syu` to force a refresh of all package lists and upgrade. Also make sure the selected **mirrors** are up-to-date and **repositories** are correctly configured. You can also use **Reflector** to keep the mirrors up-to-date.

If *pacman* reports there is nothing to update, but the `Failed retrieving file` error continues to be printed, consider **forcing a database download** with `pacman -Syyu`. This is never needed under normal circumstances, so inspect more closely the **status and consistency of the mirror**.

It could also be that the repository containing the package is not enabled on your system, e.g. the package could be in the **multilib** repository, but *multilib* is not enabled in your `pacman.conf`.

See also **FAQ#Why is there only a single version of each shared library in the official repositories?**.

3.5 Fixing an unbootable system caused by an interrupted upgrade

Whether due to power loss, kernel panic or hardware failure an update may be interrupted. In most cases, there will not be much damage but the system will likely be unbootable.

1. Ready a **USB flash installation medium** and boot it.
2. **Mount** the root filesystem as well as your **ESP**.
3. `arch-chroot` into the mounted root filesystem.
4. Check `/var/log/pacman.log` and replicate the exact update by supplying the *entire* list of packages that was upgraded during the failed transaction to `pacman -Syu`, allowing it to reinstall while resuming the original update:

```
# pacman -Syu $(grep "\[2025-07-27T22.*\] \[ALPM\] upgraded" /var/log/pacman.log | cut -d " " -f4 | tr "\n"
" ")
```

Replicating the *exact* upgrade is needed to ensure the right scriptlets and hooks will run.

3.6 Pacman crashes during an upgrade

In the case that *pacman* crashes with a "database write" error while removing packages, and reinstalling or upgrading packages fails thereafter, do the following:

1. Boot using the Arch **USB flash installation medium**. Preferably use a recent media so that the *pacman* version matches/is newer than the system.
2. Mount the system's root filesystem, e.g., `mount /dev/sdaX /mnt` as root, and check the mount has sufficient space with `df -h`
3. Mount the proc, sys and dev filesystems as well:
   `mount -t proc proc /mnt/proc; mount --rbind /sys /mnt/sys; mount --rbind /dev /mnt/dev`

4. If the system uses default database and directory locations, you can now update the system's *pacman* database and upgrade it via `pacman --root=/mnt --cachedir=/mnt/var/cache/pacman/pkg -Syu` as root.
   - Alternatively, if you cannot update/upgrade, refer to **Pacman/Tips and tricks#Reinstalling all packages**.
5. After the upgrade, one way to double-check for not upgraded but still broken packages: `find /mnt/usr/lib -size 0`
6. Followed by a re-install of any still broken package via `pacman --root /mnt --cachedir=/mnt/var/cache/pacman/pkg -S package` .

3.6.1 pacman: command not found

If `/var/cache/pacman/pkg` is a symlink, *pacman* will try to make a directory instead and thus remove this symlink during self-upgrade. This will cause the update to fail. As a result, `/usr/bin/pacman` and other contents of the **pacman (https://archlinux.org/packages/?name=pacman)** package will be missing.

Never symlink `/var/cache/pacman/pkg` because it is controlled by *pacman*. Use the `CacheDir` option or a bind mount instead; see **#Package cache directory**.

If you have already encountered this problem and broke your system, you can manually extract `/usr` contents from the package to restore *pacman* and then reinstall it properly; see **FS#73306 (https://bugs.archlinux.org/task/73306)** and **related forum thread (https://bbs.archlinux.org/viewtopic.php?id=241213)** for details.

3.7 Manually reinstalling pacman

3.7.1 Using pacman-static

pacman-static (https://aur.archlinux.org/packages/pacman-static/)[AUR] is a statically compiled version of *pacman*, so it will be able to run even when the libraries on the system are not working. This can also come in handy when a **partial upgrade** was performed and *pacman* can not run anymore.

The pinned comment and the PKGBUILD provides a way to directly download the binary, which can be used to reinstall *pacman* or to upgrade the entire system in case of partial upgrades.

3.7.1.1 Using a precompiled pacman-static binary when PKGBUILD build fails

In some situations, your system may be too broken (e.g., due to missing or incompatible libraries) to run `makepkg` or build the `pacman-static` package from the AUR successfully.

If building from the PKGBUILD fails or `makepkg` cannot be run, you can download a precompiled `pacman-static` binary from a trusted source. This static binary does not depend on system libraries and can be used to restore a working `pacman` on your system.

A reliable source for the binary is:

```
# https://pkgbuild.com/~morganamilo/pacman-static/x86_64/bin/pacman-static
```

To use it, run:

```
$ curl -L -o pacman-static https://pkgbuild.com/~morganamilo/pacman-static/x86_64/bin/pacman-static
$ chmod +x pacman-static
$ sudo ./pacman-static -Syu pacman
```

This will update your system and reinstall `pacman`, fixing broken dependencies related to missing shared libraries.

3.7.2 Using an external pacman

If even `pacman-static` does not work, it is possible to recover using an external *pacman*. One of the easiest methods to do so is by using the **archiso** and simply using `--sysroot` or `--root` to specify the mount point of the system to perform the operation on. See **Chroot#Using chroot** on how to mount the necessary filesystems required by `--sysroot`.

3.7.3 By manually extracting

> **Warning**
>
> It is extremely easy to break your system even worse using this approach. Use this only as a last resort if the method from **#Pacman crashes during an upgrade** is not an option.

Even if *pacman* is terribly broken, you can fix it manually by downloading the latest packages and extracting them to the correct locations. The rough steps to perform are:

1. Determine the `pacman (https://archlinux.org/packages/?name=pacman)` dependencies to install
2. Download each package from a **mirror** of your choice
3. Extract each package to root
4. Reinstall these packages with `pacman -S --overwrite` to update the package database accordingly
5. Do a full system upgrade

If you have a healthy Arch system on hand, you can see the full list of dependencies with:

```
$ pacman -Q $(pactree -u pacman)
```

But you may only need to update a few of them depending on your issue. An example of extracting a package is

```
# tar -xvpwf package.tar.zst -C / --exclude .PKGINFO --exclude .INSTALL --exclude .MTREE --exclude .BUILDINF
O
```

Note the use of the `w` flag for interactive mode. Running non-interactively is very risky since you might end up overwriting an important file. Also take care to extract packages in the correct order (i.e. dependencies first). **This forum post (https://bbs.archlinux.org/viewtopic.php?id=95007)** contains an example of this process

where only a couple *pacman* dependencies are broken.

3.8 "Unable to find root device" error after rebooting

Most likely the **initramfs** became corrupted during a **kernel** update (improper use of *pacman*'s `--overwrite` option can be a cause). There are two options; first, try the *Fallback* entry.

> **Tip**
>
> In case you removed the *Fallback* entry, you can always press the `Tab` key when the boot loader menu shows up (for Syslinux) or `e` (for GRUB or systemd-boot), rename it `initramfs-linux-fallback.img` and press `Enter` or `b` (depending on your **boot loader**) to boot with the new parameters.

Once the system starts, run this command (for the stock **linux (https://archlinux.org/packages/?name=linux)** kernel) either from the console or from a terminal to rebuild the initramfs image:

```
# mkinitcpio -p linux
```

If that does not work, from a current Arch release (CD/DVD or USB stick), **mount** your root and boot partitions to `/mnt` and `/mnt/boot`, respectively. Then **chroot** using *arch-chroot*:

```
# arch-chroot /mnt
# pacman -Syu mkinitcpio systemd linux
```

> **Note**
>
> - If you do not have a current release or if you only have some other "live" Linux distribution laying around, you can **chroot** using the old fashioned way. Obviously, there will be more typing than simply running the `arch-chroot` script.
> - If *pacman* fails with `Could not resolve host`, please **check your internet connection**.
> - If you cannot enter the arch-chroot or chroot environment but need to re-install packages, you can use the command `pacman --sysroot /mnt -Syu foo bar` to use *pacman* on your root partition.

Reinstalling the kernel (the **linux (https://archlinux.org/packages/?name=linux)** package) will automatically re-generate the initramfs image with `mkinitcpio -p linux`. There is no need to do this separately.

Afterwards, it is recommended that you run `exit`, `umount /mnt/{boot,}` and `reboot`.

3.9 "Warning: current locale is invalid; using default "C" locale" error

As the error message says, your locale is not correctly configured. See **Locale**.

3.10 Missing Locales Warning Messages

When locale files are intentionally removed by tools such as **bleachbit (https://archlinux.org/packages/?name=bleachbit)** or **localepurge (https://aur.archlinux.org/packages/localepurge/)**[AUR], pacman may issue warnings about missing locales during package updates.

To suppress these warnings, you can comment out the `CheckSpace` option in `pacman.conf`. Keep in mind that disabling CheckSpace turns off the space-checking functionality for all package installations, so use this workaround only when you have alternative means to monitor disk space.

3.11 Pacman does not honor proxy settings

Make sure that the relevant environment variables (`$http_proxy`, `$ftp_proxy` etc.) are set up. If you use *pacman* with **sudo**, you need to configure sudo to **pass these environment variables to pacman**. Also, ensure the configuration of **dirmngr** has `honor-http-proxy` in `/etc/pacman.d/gnupg/dirmngr.conf` to honor the proxy when refreshing the keys.

3.12 How do I reinstall all packages, retaining information on whether something was explicitly installed or as a dependency?

To reinstall all the native packages: `pacman -Qnq | pacman -S -` or `pacman -S $(pacman -Qnq)` (the `-S` option preserves the installation reason by default).

You will then need to reinstall all the foreign packages, which can be listed with `pacman -Qmq`.

3.13 "Cannot open shared object file" error

It looks like previous *pacman* transaction removed or corrupted shared libraries needed for *pacman* itself.

To recover from this situation, you need to unpack required libraries to your filesystem manually. First find what package contains the missed library and then locate it in the *pacman* cache (`/var/cache/pacman/pkg/`). Unpack required shared library to the filesystem. This will allow to run *pacman*.

Now you need to **reinstall** the broken package. Note that you need to use `--overwrite` flag as you just unpacked system files and *pacman* does not know about it. Pacman will correctly replace our shared library file with one from package.

That's it. Update the rest of the system.

3.14 Freeze of package downloads

Some issues have been reported regarding network problems that prevent *pacman* from updating/synchronizing repositories. **[2] (https://bbs.archlinux.org/viewtopic.php?id=68944)** **[3] (https://bbs.archlinux.org/viewtopic.php?id=65728)** When installing Arch Linux natively, these issues have been resolved by replacing the

default *pacman* file downloader with an alternative (see **Improve pacman performance** for more details). When installing Arch Linux as a guest OS in **VirtualBox**, this issue has also been addressed by using *Host interface* instead of *NAT* in the machine properties.

3.15 Failed retrieving file 'core.db' from mirror

If you receive this error message with correct **mirrors**, try setting a different **name server**.

3.16 error: 'local-package.pkg.tar': permission denied

If you want to install a package on an sshfs mount using `pacman -U` and receive this error, move the package to a local directory and try to install again.

3.17 error: could not determine cachedir mount point /var/cache/pacman/pkg

Upon executing, e.g., `pacman -Syu` inside a chroot environment an error is encountered:

```
error: could not determine cachedir mount point /var/cache/pacman/pkg
error: failed to commit transaction (not enough free disk space)
```

This is frequently caused by the chroot directory not being a mountpoint when the chroot is entered. See the note at **Install Arch Linux from existing Linux#Downloading basic tools** for a solution, and **`arch-chroot(8)` (https://man.archlinux.org/man/arch-chroot.8)** for an explanation and an example of using bind mounting to make the chroot directory a mountpoint.

3.18 error: GPGME error: No data

If you are unable to update packages and receive this error, then try `rm -r /var/lib/pacman/sync/` before attempting to update.

If removing sync files doesn't help, check that the sync files are `gzip compressed data` using `file /var/lib/pacman/sync/*` before attempting to update. A router or proxy might corrupt the downloads. Corruption could possibly be HTML type.

If sync files are of the correct type, there might be an issue with the mirror server. Look up the mirror server(s) in use with `pacman-conf -r core` and `pacman-conf -r extra`. Paste the first returned url in a browser and check that a file listing is returned. In case the mirror returns an error, comment it in `/etc/pacman.d/mirrorlist`. You may try updating or re-ranking **mirrors**.

3.19 error: GPGME error: General error and ":: File /var/cache/pacman/pkg/<package>.pkg.tar.zst is corrupted (invalid or corrupted package (PGP signature)).

If this error occurs and you're for instance unable to update your system or any package at all, it is possible that you have `DISPLAY` set to a blank value, which seems to break the GPG-Flow.

In this case, `unset DISPLAY` or setting it to a arbitrary value will most likely allow to update again, in case any other option above didn't do the trick yet. See **this (https://bbs.archlinux.org/viewtopic.php?pid=2204786)** post for further details.

3.20 Reinstall broken or out-of-sync packages

One may use the `pacman -Qk $pkg` to check if the installed files of the `$pkg` package match the files from its database version. For several packages, one may use the following loop to reinstall all packages which have missing file(s):

```
# LC_ALL=C.UTF-8 pacman -Qk 2>/dev/null | grep -v ' 0 missing files' | cut -d: -f1 |
    while read -r package; do
        pacman -S "$package" --noconfirm
    done
```

Suppose that your local database located in `/var/lib/pacman` is more up-to-date compared to installed packages in the `/` filesystem (*e.g.*, because of a partial rollback), then this method is the appropriate one to re-synchronize the root filesystem with the local database.

4 See also

- **libalpm(3) (https://man.archlinux.org/man/libalpm.3)**
- **pacman(8) (https://man.archlinux.org/man/pacman.8)**
- **pacman.conf(5) (https://man.archlinux.org/man/pacman.conf.5)**
- **repo-add(8) (https://man.archlinux.org/man/repo-add.8)**