pacman

The **pacman (https://www.archlinux.org/pacman/) package manager** is one of the major distinguishing features of Arch Linux. It combines a simple binary package format with an easy-to-use **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 **tar** format for packaging.

Tip: The `pacman (https://www.archlinux.org/packages/?name=pacman)` package contains other useful tools such as **makepkg**, **pactree**, **vercmp**, and **checkupdates**. Run `pacman -Qlq pacman | grep bin` to see the full list.

Contents

Related articles

Creating packages

Downgrading packages

pacman/Package signing

pacman/Pacnew and Pacsave

pacman/Restore local database

pacman/Rosetta

pacman/Tips and tricks

FAQ#Package management

System maintenance

Arch Build System

Official repositories

Arch User Repository

Usage

What follows is just a small sample of the operations that *pacman* can perform. To read more examples, refer to **pacman(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/pacman.8)**.

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

Installing packages

Note: Packages often have a series of **optional dependencies** which are packages that provide additional functionality to the application, albeit not strictly required for running it. When installing a package, *pacman* will list its optional dependencies among the output messages, but they will not be found in `pacman.log` : use the **pacman -Si** command to view the optional dependencies of a package, together with short descriptions of their functionality.

Note: When installing a package which you require only as (optional) dependency of some other package (i.e. not required by you explicitly otherwise), it is recommended to use `--asdeps` option. For details see **Installation reason**.

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)**.

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 *testing*. To install the former version, the repository needs to be defined in front:

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

To install a number of packages sharing similar patterns in their names -- not the entire group nor all matching packages; eg. **plasma (https://www.archlinux.org/groups/x86_64/plasma/)**:

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

Of course, that is not limited and can be expanded to however many levels needed:

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

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://www.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://www.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.

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
```

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:

```
# 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 folder).

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
```

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.

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 -Fs string1 string2 ...
```

To display extensive information about a given package:

```
$ 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 -Fo /path/to/file_name
```

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

```
$ pacman -Qdt
```

Tip: Add the above command to a pacman post-transaction **hook** to be notified if a transaction orphaned a package. This can be useful for being notified when a package has been dropped from a repository, since any dropped package will also be orphaned on a local installation (unless it was explicitly installed). To avoid any "failed to execute command" errors when no orphans are found, use the following command for `Exec` in your hook: `/usr/bin/bash -c "/usr/bin/pacman -Qtd || /usr/bin/echo '=> None found.'"`

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

```
$ pacman -Qet
```

To list a dependency tree of a package:

```
$ pactree package_name
```

To list all the packages recursively depending on an *installed* package, use *whoneeds* from **pkgtools (https://aur.archlinux.org/packages/pkgtools/)**[AUR]:

```
$ whoneeds package_name
```

or the reverse flag to *pactree*:

```
$ pactree -r package_name
```

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

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 tar-gzipped archives containing one directory for each package, for example for the **which (https://www.archlinux.org/packages/?name=which)** package:

```
% tree which-2.20-6
which-2.20-6
|-- depends
`-- desc
```

The `depends` file lists the packages this package depends on, while `desc` has a description of the package such as the file size and the MD5 hash.

Cleaning the package cache

Pacman stores its downloaded packages in `/var/cache/pacman/pkg/` and does not remove the old or uninstalled versions automatically. Therefore, it is necessary to deliberately clean up that folder periodically to prevent such folder to grow indefinitely in size.

The built-in option to remove all the cached packages that are not currently installed is:

```
# pacman -Sc
```

Warning:

- Only do this when certain that previous package versions are not required, for example for a later **downgrade**. `pacman -Sc` only leaves the versions of packages which are *currently installed* available, older versions would have to be retrieved through other means, such as the **Archive**.
- It is possible to empty the cache folder fully with `pacman -Scc`. In addition to the above, this also prevents from reinstalling a package directly *from* the cache folder in case of need, thus requiring a new download. It should be avoided unless there is an immediate need for disk space.

Because of the above limitations, consider an alternative for more control over which packages, and how many, are deleted from the cache:

The *paccache* script, provided by the **pacman (https://www.archlinux.org/packages/?name=pacman)** package itself, deletes all cached versions of each package regardless of whether they are installed or not, except for the most recent 3, by default:

```
# paccache -r
```

Tip: You can create **#Hooks** to run this automatically after every pacman transaction. See **this thread (https://bbs.archlinux.org/viewtopic.php?pid=1694743#p1694743)** for examples.

You can also define how many recent versions you want to keep:

```
# paccache -rk 1
```

To remove all cached versions of uninstalled packages, re-run *paccache* with:

```
# paccache -ruk0
```

See `paccache -h` for more options.

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

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.xz
```

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

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

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

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

To inhibit the `-S`, `-U` and `-R` actions, `-p` can be used.

Pacman always lists packages to be installed or removed and asks for permission before it takes action.

Installation reason

The *pacman* database distinguishes the installed packages in two groups according to the reason why they were installed:

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

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

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

Tip: Installing optional dependencies with `--asdeps` will cause it such that if you **remove orphans**, *pacman* will also remove leftover optional dependencies.

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.

Search for a package that contains a specific file

Sync the files database:

```
# pacman -Fy
```

Search for a package containing a file, e.g.:

```
# pacman -Fs pacman
core/pacman 5.0.1-4
    usr/bin/pacman
    usr/share/bash-completion/completions/pacman
extra/xscreensaver 5.36-1
    usr/lib/xscreensaver/pacman
```

Tip: You can set a cron job or a systemd timer to sync the files database regularly.

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

Configuration

Pacman's 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://jlk.fjfi.cvut.cz/arch/manpages/man/pacman.conf.5)**.

General options

General options are in the `[options]` section. Read **pacman(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/pacman.8)** or look in the default `pacman.conf` for information on what can be done here.

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
```

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, specify it as such:

```
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.

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
```

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
```

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

Skip files from being installed to system

To always skip installation of specific directories list them under `NoExtract`. For example, to avoid installation of **systemd** units use this:

```
NoExtract=usr/lib/systemd/system/*
```

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

Tip: *Pacman* issues warning messages about missing locales when updating a package for which locales have been cleared by *localepurge* or *bleachbit*. Commenting the `CheckSpace` option in `pacman.conf` suppresses such warnings, but consider that the space-checking functionality will be disabled for all packages.

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.

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 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, package `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 package `tomcat8` contains files specifying users and tmp files.

For more information on alpm hooks, see **alpm-hooks(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/alpm-hooks.5)**.

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.

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**.

Troubleshooting

"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.
```

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

The problem is usually trivial to solve. 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* or a frontend, for example through `make install`, you have to remove it and all its files and reinstall properly using *pacman*. 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 exceptionally run `pacman -S --force $package` to force *pacman* to overwrite these files.

Warning: Take care when using the `--force` switch (for example `pacman -Syu --force`) as it can cause major problems if used improperly. It is highly recommended to only use this option when the Arch news instructs the user to do so.

"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" -exec rm {} \;
```

"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
```

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 (and watch out for typos!). If certain the package exists, your package list may be out-of-date or your repositories may be incorrectly configured. Try running `pacman -Syyu` to force a refresh of all package lists and upgrade.

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?**.

Manually reinstalling pacman

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 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 --force` 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.xz -C / --exclude .PKGINFO --exclude .INSTALL --exclude .MTREE --excl
ude .BUILDINFO
```

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.

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 installation media. 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 -Syyu` as root.
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` .

"Unable to find root device" error after rebooting

Most likely your initramfs got broken during a kernel update (improper use of *pacman*'s `--force` option can be a cause). You have two options; first, try the *Fallback* entry.

Tip: In case you removed the *Fallback* entry, you can always press the `Tab` key when the bootloader 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 bootloader) to boot with the new parameters.

Once the system starts, run this command (for the stock **linux (https://www.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. 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 -r /mnt -Syu foo bar` to use *pacman* on your root partition.

Reinstalling the kernel (the **linux (https://www.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`.

Signature from "User <email@example.org>" is unknown trust, installation failed

You can try to either:

- update the known keys, i.e. `pacman-key --refresh-keys`
- manually upgrade **archlinux-keyring (https://www.archlinux.org/packages/?name=archlinux-keyring)** package first, i.e. `pacman -Sy archlinux-keyring && pacman -Su`
- follow **pacman-key#Resetting all the keys**

Request on importing PGP keys

If **installing** Arch with an outdated ISO, you are likely prompted to import PGP keys. Agree to download the key to proceed. If you are unable to add the PGP key successfully, update the keyring or upgrade **archlinux-keyring (https://www.archlinux.org/packages/?name=archlinux-keyring)** (see **above**).

Error: key "0123456789ABCDEF" could not be looked up remotely

If packages are signed with new keys, which were only recently added to **archlinux-keyring (https://www.archlinux.org/packages/?name=archlinux-keyring)**, these keys are not locally available during update (chicken-egg-problem). The installed **archlinux-keyring (https://www.archlinux.org/packages/?name=archlinux-keyring)** does not contain the key, until it is updated.

Pacman tries to bypass this by a lookup through a key-server, which might not be possible e.g. behind proxys or firewalls and results in the stated error. Upgrade **archlinux-keyring (https://www.archlinux.org/packages/?name=archlinux-keyring)** first as shown **above**.

Signature from "User <email@archlinux.org>" is invalid, installation failed

When the system time is faulty, signing keys are considered expired (or invalid) and signature checks on packages will fail with the following error:

```
error: package: signature from "User <email@archlinux.org>" is invalid
error: failed to commit transaction (invalid or corrupted package (PGP signature))
Errors occured, no packages were upgraded.
```

Make sure to correct the **time**, for example with `ntpd -qg` run as root, and run `hwclock -w` as root before subsequent installations or upgrades.

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

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

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**.

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 -` (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` .

"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 `--force` 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.

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.

Failed retrieving file 'core.db' from mirror

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

Understanding

What happens during package install/upgrade/removal

When successfully completing a package transaction, pacman performs for the following high-level steps:

1. pacman obtains the to-be installed package file for all packages queued in a transaction
2. pacman performs various checks that the packages can likely be installed
3. if pre-existing pacman `PreTransaction` hooks apply, they are executed
4. Each package is installed/upgraded/removed in turn
    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)
    3. pacman untars the package and dumps its files into the file system (in the case of an installed or upgraded package)
    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

See also

- **Pacman Home Page (https://www.archlinux.org/pacman/)**
- **libalpm(3) (https://jlk.fjfi.cvut.cz/arch/manpages/man/libalpm.3)**
- **pacman(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/pacman.8)**
- **pacman.conf(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/pacman.conf.5)**
- **repo-add(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/repo-add.8)**