# Arch User Repository

The Arch User Repository (AUR) is a community-driven repository for Arch users. It contains package descriptions (**PKGBUILDs**) that allow you to compile a package from source with **makepkg** and then install it via **pacman**. The AUR was created to organize and share new packages from the community and to help expedite popular packages' inclusion into the **community repository**. This document explains how users can access and utilize the AUR.

A good number of new packages that enter the official repositories start in the AUR. In the AUR, users are able to contribute their own package builds (`PKGBUILD` and related files). The AUR community has the ability to vote for packages in the AUR. If a package becomes popular enough — provided it has a compatible license and good packaging technique — it may be entered into the *community* repository (directly accessible by *pacman* or **abs**).

**Warning:** AUR packages are user-produced content. These `PKGBUILD`s are completely unofficial and have not been thoroughly vetted. Any use of the provided files is at your own risk.

**Related articles**

- makepkg
- pacman
- PKGBUILD
- .SRCINFO
- Aurweb RPC interface
- AUR submission guidelines
- AUR Trusted User Guidelines
- Official repositories
- Arch Build System
- Creating packages
- AUR helpers
- AUR Cleanup Day

## Contents

## Getting started

Users can search and download **PKGBUILDs** from the **AUR Web Interface (https://aur.archlinux.org)**. These `PKGBUILD` s can be built into installable packages using *makepkg*, then installed using *pacman*.

- Ensure the **`base-devel (https://archlinux.org/groups/x86_64/base-devel/)`** package group is installed in full ( `pacman -S --needed base-devel` ).
- Glance over the **#FAQ** for answers to the most common questions.
- You may wish to adjust `/etc/makepkg.conf` to optimize the build process to your system prior to building packages from the AUR. A significant improvement in package build times can be realized on systems with multi-core processors by adjusting the `MAKEFLAGS` variable, by using multiple cores for compression, or by using different compression algorithm. Users can also enable hardware-specific compiler optimizations via the `CFLAGS` variable. See **makepkg#Tips and tricks** for more information.

It is also possible to interact with the AUR through SSH: type `ssh aur@aur.archlinux.org help` for a list of available commands.

## History

In the beginning, there was `ftp://ftp.archlinux.org/incoming` , and people contributed by simply uploading the **PKGBUILD**, the needed supplementary files, and the built package itself to the server. The package and associated files remained there until a **Package Maintainer** saw the program and adopted it.

Then the Trusted User Repositories were born. Certain individuals in the community were allowed to host their own repositories for anyone to use. The AUR expanded on this basis, with the aim of making it both more flexible and more usable. In fact, the AUR maintainers are still referred to as TUs (Trusted Users).

Between 2015-06-08 and 2015-08-08 the AUR transitioned from version 3.5.1 to 4.0.0, introducing the use of Git repositories for publishing the `PKGBUILD` s. Existing packages were dropped unless manually migrated to the new infrastructure by their maintainers.

### Git repositories for AUR3 packages

The **AUR Archive (https://github.com/aur-archive)** on GitHub has a repository for every package that was in AUR 3 at the time of the migration. Alternatively, there is the **aur3-mirror (https://github.com/felixonmars/aur3-mirror/)** repository which provides the same.

## Installing and upgrading packages

Installing packages from the AUR is a relatively simple process. Essentially:

1. Acquire the build files, including the **PKGBUILD** and possibly other required files, like **systemd** units and patches (often not the actual code).
2. Verify that the `PKGBUILD` and accompanying files are not malicious or untrustworthy.
3. Run `makepkg` in the directory where the files are saved. This will download the code, compile it, and package it.
4. Run `pacman -U package_file` to install the package onto your system.

**Note:** The AUR is unsupported, so any packages you install are *your responsibility* to update, not pacman's. If packages in the official repositories are updated, you will need to rebuild any AUR packages that depend on those libraries.

### Prerequisites

First ensure that the necessary tools are installed by **installing** the `base-devel (https://archlinux.org/groups/x86_64/base-devel/)` group in full which includes `make (https://archlinux.org/packages/?name=make)` and other tools needed for compiling from source.

**Tip:** Use the `--needed` flag when installing the `base-devel (https://archlinux.org/groups/x86_64/base-devel/)` group to skip packages you already have instead of reinstalling them.

**Note:** Packages in the AUR assume that the `base-devel (https://archlinux.org/groups/x86_64/base-devel/)` group is installed, i.e. they do not list the group's members as build dependencies explicitly.

Next choose an appropriate build directory. A build directory is simply a directory where the package will be made or "built" and can be any directory. The examples in the following sections will use `~/builds` as the build directory.

### Acquire build files

Locate the package in the AUR. This is done using the search field at the top of the **AUR home page (https://aur.archlinux.org/)**. Clicking the application's name in the search list brings up an information page on the package. Read through the description to confirm that this is the desired package, note when the package was last updated, and read any comments.

There are several methods for acquiring the build files for a package:

- Clone its **git** repository, labeled "Git Clone URL" in the "Package Details" on its AUR page. This is the preferred method, an advantage of which is that you can easily get updates to the package via `git pull`.

```
$ git clone https://aur.archlinux.org/package_name.git
```

- Download a snapshot, either by clicking the "Download snapshot" link under "Package Actions" on the right hand side of its AUR page, or in a terminal:

```
$ curl -L -O https://aur.archlinux.org/cgit/aur.git/snapshot/package_name.tar.gz
```

**Note:** The snapshot file is compressed, and must be extracted (preferably in a directory set aside for AUR builds): `tar -xvf package_name.tar.gz`

## Acquire a PGP public key if needed

Check if a signature file in the form of *.sig* or *.asc* is part of the **PKGBUILD** source array, if that is the case, then acquire one of the public keys listed in the PKGBUILD **validpgpkeys** array. Refer to **makepkg#Signature checking** for more information.

## Build the package

Change directories to the directory containing the package's **PKGBUILD**.

```
$ cd package_name
```

**Warning:** Carefully check the `PKGBUILD`, any *.install* files, and any other files in the package's git repository for malicious or dangerous commands. If in doubt, do not build the package, and **seek advice** on the forums or mailing list. Malicious code has been found in packages before. **[1] (https://lists.archlinux.org/pipermail/aur-general/2018-July/034151.html)**

View the contents of all provided files. For example, to use the pager *less* to view `PKGBUILD` do:

```
$ less PKGBUILD
```

**Tip:** If you are updating a package, you may want to look at the changes since the last commit.

- To view changes since the last git commit you can use `git show`.
- To view changes since the last commit using *vimdiff*, do `git difftool @~..@ vimdiff`. The advantage of *vimdiff* is that you view the entire contents of each file along with indicators on what has changed.

Make the package. After manually confirming the contents of the files, run **makepkg** as a normal user. Some helpful flags:

- `-s` / `--syncdeps` automatically resolves and installs any dependencies with **pacman** before building. If the package depends on other AUR packages, you will need to manually install them first.
- `-i` / `--install` installs the package if it is built successfully. This lets you skip the next step that is usually done manually.

- `-r` / `--rmdeps` removes build-time dependencies after the build, as they are no longer needed. However these dependencies may need to be reinstalled the next time the package is updated.
- `-c` / `--clean` cleans up temporary build files after the build, as they are no longer needed. These files are usually needed only when debugging the build process.

**Tip:** Use `git clean -dfX` to delete all files that are ignored by git, thus deleting all previously built package files.

### Install the package

The package can now be installed with pacman:

```
# pacman -U package_name-version-architecture.pkg.tar.zst
```

**Note:**

- If you have changed your `PKGEXT` in `makepkg.conf`, the name of the package file may be slightly different.
- The above example is only a brief summary of the build process. It is **highly** recommended to read the **makepkg** and **ABS** articles for more details.

### Upgrading packages

In the directory containing the package's **PKGBUILD** you must first update the files and changes by using the command

```
$ git pull
```

then follow the previous build and install instructions.

## Feedback

### Commenting on packages

The **AUR Web Interface (https://aur.archlinux.org)** has a comments facility that allows users to provide suggestions and feedback on improvements to the **PKGBUILD** contributor.

**Tip:** Avoid pasting patches or `PKGBUILD`s into the comments section: they quickly become obsolete and just end up needlessly taking up lots of space. Instead email those files to the maintainer, or even use a **pastebin**.

**Python-Markdown (https://python-markdown.github.io/)** provides basic **Markdown** syntax to format comments.

**Note:**

- This implementation has some occasional **differences (https://python-markdown.github.io/#differences)** with the official **syntax rules (https://daringfireball.net/projects/markdown/syntax)**.

- Commit hashes to the Git repository of the package and references to Flyspray tickets are converted to links automatically.
- Long comments are collapsed and can be expanded on demand.

### Voting for packages

One of the easiest activities for **all** Arch users is to browse the AUR and **vote** for their favourite packages using the online interface. All packages are eligible for adoption by a TU for inclusion in the **community repository**, and the vote count is one of the considerations in that process; it is in everyone's interest to vote!

Sign up on the **AUR website (https://aur.archlinux.org/)** to get a "Vote for this package" option while browsing packages. After signing up it is also possible to vote from the commandline with `aurvote (https://aur.archlinux.org/packages/aurvote/)`AUR, `aurvote-git (https://aur.archlinux.org/packages/aurvote-git/)`AUR or `aur-auto-vote-git (https://aur.archlinux.org/packages/aur-auto-vote-git/)`AUR.

Alternatively, if you have set up **ssh authentication**, you can directly vote from the command line using your ssh key. This means that you will not need to save or type in your AUR password.

```
$ ssh aur@aur.archlinux.org vote package_name
```

### Flagging packages out-of-date

First, you should flag the package *out-of-date* indicating details on why the package is outdated, preferably including links to the release announcement or the new release **tarball**.

You should also try to reach out to the maintainer directly by email. If there is no response from the maintainer after *two weeks*, you can file an *orphan* request. See **AUR submission guidelines#Requests** for details.

**Note:** **VCS packages** are not considered out of date when the `pkgver` changes, do not flag them as the maintainer will merely unflag the package and ignore you. AUR maintainers should not commit mere `pkgver` bumps.

## Debugging the package build process

1. Ensure your build environment is up-to-date by **upgrading** before building anything.
2. Ensure you have the `base-devel (https://archlinux.org/groups/x86_64/base-devel/)` group installed.
3. Use the `-s` option with `makepkg` to check and install all dependencies needed before starting the build process.
4. Try the default **makepkg configuration (https://github.com/archlinux/svntogit-packages/blob/packages/pacman/trunk/makepkg.conf)**.
5. See **Makepkg#Troubleshooting** for common issues.

If you are having trouble building a package, first read its **PKGBUILD** and the comments on its AUR page.

It is possible that a `PKGBUILD` is broken for everyone. If you cannot figure it out on your own, report it to the maintainer (e.g. by **posting the errors you are getting in the comments on the AUR page**). You may also seek help in the **AUR Issues, Discussion & PKGBUILD Requests forum (https://bbs.archlinux.org/viewforum.php?id=38)**.

The reason might not be trivial after all. Custom `CFLAGS`, `LDFLAGS` and `MAKEFLAGS` can cause failures. To avoid problems caused by your particular system configuration, build packages in a **clean chroot**. If the build process still fails in a clean chroot, the issue is probably with the `PKGBUILD`.

See **Creating packages#Checking package sanity** about using `namcap`. If you would like to have a `PKGBUILD` reviewed, post it on the **aur-general mailing list (https://mailman.archlinux.org/mailman/listinfo/aur-general)** to get feedback from the TUs and fellow AUR members, or the **Creating & Modifying Packages forum (https://bbs.archlinux.org/viewforum.php?id=4)**. You could also seek help in the **IRC channel #archlinux-aur (ircs://irc.libera.chat/archlinux-aur)** on the **Libera Chat (https://libera.chat)** network.

## Submitting packages

Users can share **PKGBUILDs** using the Arch User Repository. See **AUR submission guidelines** for details.

## Web interface translation

See **i18n.txt (https://gitlab.archlinux.org/archlinux/aurweb/-/blob/master/doc/i18n.txt)** in the AUR source tree for information about creating and maintaining translation of the **AUR Web Interface (https://aur.archlinux.org)**.

## FAQ

### What kind of packages are permitted on the AUR?

The packages on the AUR are merely "build scripts", i.e. recipes to build binaries for **pacman**. For most cases, everything is permitted, subject to **usefulness and scope guidelines**, as long as you are in compliance with the licensing terms of the content. For other cases, where it is mentioned that "you may not link" to downloads, i.e. contents that are not redistributable, you may only use the file name itself as the source. This means and requires that users already have the restricted source in the build directory prior to building the package. When in doubt, ask.

### How can I vote for packages in the AUR?

See **#Voting for packages**.

### What is a Trusted User / TU?

A **Trusted User**, in short TU, is a person who is chosen to oversee AUR and the **community repository**. They are the ones who maintain popular **PKGBUILDs** in *community*, and overall keep the AUR running.

### What is the difference between the Arch User Repository and the community repository?

The Arch User Repository is where all **PKGBUILDs** that users submit are stored, and must be built manually with **makepkg**. When `PKGBUILD`s receive enough community interest and the support of a TU, they are moved into the **community repository** (maintained by the TUs), where the binary packages can be installed with **pacman**.

### Foo in the AUR is outdated; what should I do?

See **#Flagging packages out-of-date**.

In the meantime, you can try updating the package yourself by editing the **PKGBUILD** locally. Sometimes, updates do not require changes to the build or package process, in which case simply updating the `pkgver` or `source` array is sufficient.

### Foo in the AUR does not compile when I run makepkg; what should I do?

You are probably missing something trivial, see **#Debugging the package build process**.

### ERROR: One or more PGP signatures could not be verified!; what should I do?

Most likely you do not have the required public key(s) in your personal keyring to verify downloaded files. See **Makepkg#Signature checking** for details.

### How do I create a PKGBUILD?

Consult the **AUR submission guidelines#Rules of submission**, then see **creating packages**.

### I have a PKGBUILD I would like to submit; can someone check it to see if there are any errors?

There are several channels available to submit your package for review; see **#Debugging the package build process**.

### How to get a PKGBUILD into the community repository?

Usually, at least 10 votes are required for something to move into **community**. However, if a **TU** wants to support a package, it will often be found in the repository.

Reaching the required minimum of votes is not the only requirement, there has to be a TU willing to maintain the package. TUs are not required to move a package into the *community* repository even if it has thousands of votes.

Usually when a very popular package stays in the AUR it is because:

- Arch Linux already has another version of a package in the repositories

- Its license prohibits redistribution
- It helps retrieve user-submitted PKGBUILDs. AUR helpers are unsupported (https://bbs.archlinux.org/viewtopic.php?pid=828310#p828310) by definition.

See also Rules for Packages Entering the community Repo.

### How can I speed up repeated build processes?

See Makepkg#Improving compile times.

### What is the difference between foo and foo-git packages?

Many AUR packages come in "stable" release and "unstable" development versions. Development packages usually have a suffix denoting their Version Control System and are not intended for regular use, but may offer new features or bugfixes. Because these packages only download the latest available source when you execute `makepkg`, their `pkgver()` in the AUR does not reflect upstream changes. Likewise, these packages cannot perform an authenticity checksum on any VCS source.

See also System maintenance#Use proven software packages.

### Why has foo disappeared from the AUR?

It is possible the package has been adopted by a TU and is now in the community repository.

Packages may be deleted if they did not fulfill the rules of submission. See the aur-requests archives (https://lists.archlinux.org/pipermail/aur-requests/) for the reason for deletion.

**Note:** The git repository for a deleted package typically remains available. See AUR submission guidelines#Requests for details.

If the package used to exist in AUR3, it might not have been migrated to AUR4 (https://lists.archlinux.org/pipermail/aur-general/2015-August/031322.html). See the #Git repositories for AUR3 packages where these are preserved.

### How do I find out if any of my installed packages disappeared from AUR?

The simplest way is to check the HTTP status of the package's AUR page:

```
$ comm -23 <(pacman -Qqm | sort) <(curl https://aur.archlinux.org/packages.gz | gzip -cd | sort)
```

### How can I obtain a list of all AUR packages?

- https://aur.archlinux.org/packages.gz
- Use `aurpkglist` from `python3-aur (https://aur.archlinux.org/packages/python3-aur/)`AUR

## See also

- **AUR Web Interface (https://aur.archlinux.org)**
- **AUR Mailing List (https://lists.archlinux.org/listinfo/aur-general)**

Retrieved from "**https://wiki.archlinux.org/index.php?title=Arch_User_Repository&oldid=710558**"