pacman/Tips and tricks

< Pacman

For general methods to improve the flexibility of the provided tips or *pacman* itself, see **Core utilities** and **Bash**.

Related articles

Mirrors

Creating packages

Contents

Maintenance

Note: Instead of using *comm* (which requires sorted input with *sort*) in the sections below, you may also use `grep -Fxf` or `grep -Fxvf`.

See also **System maintenance**.

Listing packages

You may want to get the list of installed packages with their version, which is useful when reporting bugs or discussing installed packages.

- List all explicitly installed packages: `pacman -Qe`.
- List all explicitly installed native packages (i.e. present in the sync database) that are not direct or optional dependencies: `pacman -Qent`.
- List all foreign packages (typically manually downloaded and installed): `pacman -Qm`.
- List all native packages (installed from the sync database(s)): `pacman -Qn`.
- List packages by regex: `pacman -Qs` *regex*.
- List packages by regex with custom output format: `expac -s "%-30n %v"` *regex* (needs **expac (https://www.archlinux.org/packages/?name=expac)**).

With size

To get a list of installed packages sorted by size, which may be useful when freeing space on your hard drive:

- Install **expac (https://www.archlinux.org/packages/?name=expac)** and run `expac -H M '%m\t%n' | sort -h`.
- Run **pacgraph (https://www.archlinux.org/packages/?name=pacgraph)** with the `-c` option.

To list the download size of several packages (leave *packages* blank to list all packages):

```
$ expac -S -H M '%k\t%n' packages
```

To list explicitly installed packages not in **base (https://www.archlinux.org/groups/x86_64/base/)** nor **base-devel (https://www.archlinux.org/groups/x86_64/base-devel/)** with size and description:

```
$ expac -H M "%011m\t%-20n\t%10d" $(comm -23 <(pacman -Qqen | sort) <(pacman -Qqg base base-de
vel | sort)) | sort -n
```

By date

To list the 20 last installed packages with **expac (https://www.archlinux.org/packages/?name=expac)**, run:

```
$ expac --timefmt='%Y-%m-%d %T' '%l\t%n' | sort | tail -n 20
```

or, with seconds since the epoch (1970-01-01 UTC):

```
$ expac --timefmt=%s '%l\t%n' | sort -n | tail -n 20
```

Not in a specified group or repository

Note: To get a list of packages installed as dependencies but no longer required by any installed package, see **#Removing unused packages (orphans)**.

List explicitly installed packages not in the **base (https://www.archlinux.org/groups/x86_64/base/)** or **base-devel (https://www.archlinux.org/groups/x86_64/base-devel/)** groups:

```
$ comm -23 <(pacman -Qeq | sort) <(pacman -Qgq base base-devel | sort)
```

List all installed packages unrequired by other packages, and which are not in the **base (https://www.archlinux.org/groups/x86_64/base/)** or **base-devel (https://www.archlinux.org/groups/x86_64/base-devel/)** groups:

```
$ comm -23 <(pacman -Qqt | sort) <(pacman -Sqg base base-devel | sort)
```

As above, but with descriptions:

```
$ expac -HM '%-20n\t%10d' $(comm -23 <(pacman -Qqt | sort) <(pacman -Qqg base base-devel | sor
t))
```

List all installed packages that are *not* in the specified repository *repo_name*

```
$ comm -23 <(pacman -Qq | sort) <(pacman -Slq repo_name | sort)
```

List all installed packages that are in the *repo_name* repository:

```
$ comm -12 <(pacman -Qq | sort) <(pacman -Slq repo_name | sort)
```

List all packages on the Arch Linux ISO that are not in the base group:

```
$ comm -23 <(wget -q -O - https://git.archlinux.org/archiso.git/plain/configs/releng/packages.
both) <(pacman -Qqg base | sort)
```

Development packages

To list all development/unstable packages, run:

```
$ pacman -Qq | awk '/^.+(-cvs|-svn|-git|-hg|-bzr|-darcs)$/'
```

Listing files owned by a package with size

This one might come in handy if you have found that a specific package uses a huge amount of space and you want to find out which files make up the most of that.

```
$ pacman -Qlq package | grep -v '/$' | xargs du -h | sort -h
```

Identify files not owned by any package

If your system has stray files not owned by any package (a common case if you do not **use the package manager to install software**), you may want to find such files in order to clean them up. The general process for doing so is:

1. Create a sorted list of the files you want to check ownership of:

   ```
   $ find /etc /opt /usr | sort > all_files.txt
   ```

2. Create a sorted list of the files tracked by *pacman* (and remove the trailing slashes from directories):

   ```
   $ pacman -Qlq | sed 's|/$||' | sort > owned_files.txt
   ```

3. Find lines in the first list that are not in the second:

   ```
   $ comm -23 all_files.txt owned_files.txt
   ```

This process is tricky in practice because many important files are not part of any package (e.g. files generated at runtime, custom configs) and so will be included in the final output, making it difficult to pick out the files that can be safely deleted.

Tip: The `lostfiles (https://aur.archlinux.org/packages/lostfiles/)`AUR script performs similar steps, but also includes an extensive blacklist to remove common false positives from the output. **aconfmgr (https://github.com/CyberShadow/aconfmgr)** (`aconfmgr-git (https://aur.archlinux.org/packages/aconfmgr-git/)`AUR) also allows tracking orphaned files using a configuration script.

Removing unused packages (orphans)

For recursively removing orphans and their configuration files:

```
# pacman -Rns $(pacman -Qtdq)
```

If no orphans were found *pacman* outputs `error: no targets specified`. This is expected as no arguments were passed to `pacman -Rns`.

Note: The arguments `-Qt` list only true orphans. To include packages which are *optionally* required by another package, pass the `-t` flag twice (*i.e.*, `-Qtt`).

Removing everything but base group

If it is ever necessary to remove all packages except the base group, try this one-liner:

```
# pacman -R $(comm -23 <(pacman -Qq | sort) <((for i in $(pacman -Qqg base); do pactree -ul
"$i"; done) | sort -u))
```

The one-liner was originally devised in **this discussion (https://bbs.archlinux.org/viewtopic.php?id=130176)**, and later improved in this article.

Getting the dependencies list of several packages

Dependencies are alphabetically sorted and doubles are removed.

Note: To only show the tree of local installed packages, use `pacman -Qi`.

```
$ pacman -Si packages | awk -F'[:<=>]' '/^Depends/ {print $2}' | xargs -n1 | sort -u
```

Alternatively, with **expac (https://www.archlinux.org/packages/?name=expac)**:

```
$ expac -l '\n' %E -S packages | sort -u
```

Listing changed backup files

If you want to backup your system configuration files you could copy all files in `/etc/`, but usually you are only interested in the files that you have changed. Modified **backup files** can be viewed with the following command:

```
# pacman -Qii | awk '/^MODIFIED/ {print $2}'
```

Running this command with root permissions will ensure that files readable only by root (such as `/etc/sudoers`) are included in the output.

Tip: See **#Listing all changed files from packages** to list all changed files *pacman* knows, not only backup files.

Back-up the pacman database

The following command can be used to back up the local *pacman* database:

```
$ tar -cjf pacman_database.tar.bz2 /var/lib/pacman/local
```

Store the backup *pacman* database file on one or more offline media, such as a USB stick, external hard drive, or CD-R.

The database can be restored by moving the `pacman_database.tar.bz2` file into the `/` directory and executing the following command:

```
# tar -xjvf pacman_database.tar.bz2
```

Note: If the *pacman* database files are corrupted, and there is no backup file available, there exists some hope of rebuilding the *pacman* database. Consult **#Restore pacman's local database**.

Tip: The **`pakbak-git (https://aur.archlinux.org/packages/pakbak-git/)`**AUR package provides a script and a **systemd** service to automate the task. Configuration is possible in `/etc/pakbak.conf`.

Check changelogs easily

When maintainers update packages, commits are often commented in a useful fashion. Users can quickly check these from the command line by installing **`pacolog (https://aur.archlinux.org/packages/pacolog/)`**AUR. This utility lists recent commit messages for packages from the official repositories or the AUR, by using `pacolog <package>`.

Installation and recovery

Alternative ways of getting and restoring packages.

Installing packages from a CD/DVD or USB stick

To download packages, or groups of packages:

```
# cd ~/Packages
# pacman -Syw base base-devel grub-bios xorg gimp --cachedir .
# repo-add ./custom.db.tar.gz ./*
```

Then you can burn the "Packages" folder to a CD/DVD or transfer it to a USB stick, external HDD, etc.

To install:

1. Mount the media:

```
# mkdir /mnt/repo
# mount /dev/sr0 /mnt/repo    #For a CD/DVD.
# mount /dev/sdxY /mnt/repo   #For a USB stick.
```

2. Edit `pacman.conf` and add this repository *before* the other ones (e.g. extra, core, etc.). This is important. Do not just uncomment the one on the bottom. This way it ensures that the files from the CD/DVD/USB take precedence over those in the standard repositories:

```
/etc/pacman.conf
```

```
[custom]
SigLevel = PackageRequired
Server = file:///mnt/repo/Packages
```

3. Finally, synchronize the *pacman* database to be able to use the new repository:

```
# pacman -Syu
```

Custom local repository

Use the *repo-add* script included with *pacman* to generate a database for a personal repository. Use `repo-add --help` for more details on its usage. To add a new package to the database, or to replace the old version of an existing package in the database, run:

```
$ repo-add /path/to/repo.db.tar.gz /path/to/package-1.0-1-x86_64.pkg.tar.xz
```

Note: A package database is a tar file, optionally compressed. Valid extensions are *.db* or *.files* followed by an archive extension of *.tar*, *.tar.gz*, *.tar.bz2*, *.tar.xz*, or *.tar.Z*. The file does not need to exist, but all parent directories must exist.

The database and the packages do not need to be in the same directory when using *repo-add*, but keep in mind that when using *pacman* with that database, they should be together. Storing all the built packages to be included in the repository in one directory also allows to use shell glob expansion to add or update multiple packages at once:

```
$ repo-add /path/to/repo.db.tar.gz /path/to/*.pkg.tar.xz
```

Warning: *repo-add* adds the entries into the database in the same order as passed on the command line. If multiple versions of the same package are involved, care must be taken to ensure that the correct version is added last. In particular, note that lexical order used by the shell depends on the locale and differs from the **vercmp (https://www.archlinux.org/pacman/vercmp.8.html)** ordering used by *pacman*.

repo-remove is used to remove packages from the package database, except that only package names are specified on the command line.

```
$ repo-remove /path/to/repo.db.tar.gz pkgname
```

Once the local repository database has been created, add the repository to `pacman.conf` for each system that is to use the repository. An example of a custom repository is in `pacman.conf`. The repository's name is the database filename with the file extension omitted. In the case of the example above the repository's name would simply be *repo*. Reference the repository's location using a `file://` url, or via FTP using **ftp://localhost/path/to/directory**.

If willing, add the custom repository to the **list of unofficial user repositories**, so that the community can benefit from it.

Network shared pacman cache

If you happen to run several Arch boxes on your LAN, you can share packages so that you can greatly decrease your download times. Keep in mind you should not share between different architectures (i.e. i686 and x86_64) or you will run into problems.

Read-only cache

If you are looking for a quick solution, you can simply run a standalone webserver which other computers can use as a first mirror:

```
# ln -s /var/lib/pacman/sync/*.db /var/cache/pacman/pkg
$ sudo -u http darkhttpd /var/cache/pacman/pkg --no-server-id
```

You could also run darkhttpd as a systemd service for convenience. Just add this server at the top of your `/etc/pacman.d/mirrorlist` in client machines with `Server = http://mymirror:8080`. Make sure to keep your mirror updated.

Distributed read-only cache

There are Arch-specific tools for automatically discovering other computers on your network offering a package cache. Try **pacserve**, **`pkgdistcache (https://aur.archlinux.org/packages/pkgdistcache/)`**AUR, or **`paclan (https://aur.archlinux.org/packages/paclan/)`**AUR. pkgdistcache uses Avahi instead of plain UDP which may work better in certain home networks that route instead of bridge between WiFi and Ethernet.

Historically, there was **PkgD (https://bbs.archlinux.org/viewtopic.php?id=64391)** and **multipkg (https://github.com/toofishes/multipkg)**, but they are no longer maintained.

Read-write cache

In order to share packages between multiple computers, simply share `/var/cache/pacman/` using any network-based mount protocol. This section shows how to use **shfs** or **SSHFS** to share a package cache plus the related library-directories between multiple computers on the same local network. Keep in mind that a network shared cache can be slow depending on the file-system choice, among other factors.

First, install any network-supporting filesystem packages: **shfs-utils (https://www.archlinux.org/packages/?name=shfs-utils)**, **sshfs (https://www.archlinux.org/packages/?name=sshfs)**, **curlftpfs (https://www.archlinux.org/packages/?name=curlftpfs)**, **samba (https://www.archlinux.org/packages/?name=samba)** or **nfs-utils (https://www.archlinux.org/packages/?name=nfs-utils)**.

Tip:

- To use *sshfs* or *shfs*, consider reading **Using SSH Keys**.
- By default, *smbfs* does not serve filenames that contain colons, which results in the client downloading the offending package afresh. To prevent this, use the `mapchars` mount option on the client.

Then, to share the actual packages, mount `/var/cache/pacman/pkg` from the server to `/var/cache/pacman/pkg` on every client machine.

Note: Do not make `/var/cache/pacman/pkg` or any of its ancestors (e.g., `/var`) a symlink. *Pacman* expects these to be directories. When *pacman* re-installs or upgrades itself, it will remove the symlinks and create empty directories instead. However during the transaction *pacman* relies on some files residing there, hence breaking the update process. Refer to **FS#50298 (https://bugs.archlinux.org/task/50298)** for further details.

two-way with rsync

Another approach in a local environment is **rsync**. Choose a server for caching and enable the **Rsync#rsync daemon**. On clients synchronize two-way with this share via rsync protocol. Filenames that contain colons are no problem for the rsync protocol.

Draft example for a client, using `uname -m` within the share name ensures an architecture dependant sync:

```
# rsync rsync://server/share_$(uname -m)/ /var/cache/pacman/pkg/ ...
# pacman ...
# paccache ...
# rsync /var/cache/pacman/pkg/ rsync://server/share_$(uname -m)/  ...
```

Dynamic reverse proxy cache using nginx

nginx can be used to proxy requests to official upstream mirrors and cache the results to local disk. All subsequent requests for that file will be served directly from the local cache, minimizing the amount of internet traffic needed to update a large number of servers with minimal effort.

Warning: This method has a limitation. You must use mirrors that use the same relative path to package files and you must configure your cache to use that same path. In this example, we are using mirrors that use the relative path `/archlinux/$repo/os/$arch` and our cache's `Server` setting in `mirrorlist` is configured similarly.

In this example, we will run the cache server on `http://cache.domain.local:8080/` and storing the packages in `/srv/http/pacman-cache/`.

Create the directory for the cache and adjust the permissions so nginx can write files to it:

```
# mkdir /srv/http/pacman-cache
# chown http:http /srv/http/pacman-cache
```

Next, configure nginx as the **dynamic cache (https://gist.github.com/anonymous/97ec4148f643de925e433bed3dc7ee7d)** (read the comments for an explanation of the commands).

Finally, update your other Arch Linux servers to use this new cache by adding the following line to the `mirrorlist` file:

```
/etc/pacman.d/mirrorlist
```

```
Server = http://cache.domain.local:8080/archlinux/$repo/os/$arch
...
```

Note: You will need to create a method to clear old packages, as this directory will continue to grow over time. `paccache` (which is included with *pacman*) can be used to automate this using retention criteria of your choosing. For example, `find /srv/http/pacman-cache/ -type d -exec paccache -v -r -k 2 -c {} \;` will keep the last 2 versions of packages in your cache directory.

Synchronize pacman package cache using synchronization programs

Use **Resilio Sync** or **Syncthing** to synchronize the *pacman* cache folders (i.e. `/var/cache/pacman/pkg`).

Preventing unwanted cache purges

By default, `pacman -Sc` removes package tarballs from the cache that correspond to packages that are not installed on the machine the command was issued on. Because *pacman* cannot predict what packages are installed on all machines that share the cache, it will end up deleting files that should not be.

To clean up the cache so that only *outdated* tarballs are deleted, add this entry in the `[options]` section of `/etc/pacman.conf`:

```
CleanMethod = KeepCurrent
```

Recreate a package from the file system

To recreate a package from the file system, use *bacman* (included with *pacman*). Files from the system are taken as they are, hence any modifications will be present in the assembled package. Distributing the recreated package is therefore discouraged; see **ABS** and **Arch**

Linux Archive for alternatives.

Tip: *bacman* honours the `PACKAGER`, `PKGDEST` and `PKGEXT` options from `makepkg.conf`. Custom options for the compression tools can be configured by exporting the relevant environment variable, for example `XZ_OPT="-T 0"` will enable parallel compression for *xz*.

An alternative tool would be **fakepkg (https://aur.archlinux.org/packages/fakepkg/)**AUR. It supports parallelization and can handle multiple input packages in one command, which *bacman* both does not support.

List of installed packages

Keeping a list of explicitly installed packages can be useful to speed up installation on a new system:

```
$ pacman -Qqe > pkglist.txt
```

Note: When using option `-t`, when reinstalling the list all the non-top-level packages would be set as dependencies. With opion `-n`, foreign packages (e.g. from AUR) are ommited from the list.

To install packages from the list backup, run:

```
# pacman -S - < pkglist.txt
```

Tip:

- To skip already installed packages, use `--needed`.
- Use `comm -13 <(pacman -Qqdt | sort) <(pacman -Qqdtt | sort) > optdeplist.txt` to also create a list of the installed optional dependencies which can be reinstalled with `--asdeps`.

In case the list includes foreign packages, such as **AUR** packages, remove them first:

```
# pacman -S $(comm -12 <(pacman -Slq | sort) <(sort pkglist.txt))
```

To remove all the packages on your system that are not mentioned in the list:

```
# pacman -Rsu $(comm -23 <(pacman -Qq | sort) <(sort pkglist.txt))
```

Tip: These tasks can be automated. See **bacpac (https://aur.archlinux.org/packages/bacpac/)**AUR, **packup (https://aur.archlinux.org/packages/packup/)**AUR, **pacmanity (https://aur.archlinux.org/packages/pacmanity/)**AUR, and **pug (https://aur.archlinux.org/packages/pug/)**AUR for examples.

If you would like to keep an up-to-date list of explicitly installed packages (e.g. in combination with a versioned `/etc/`), you can set up a **hook**. Example:

```
[Trigger]
Operation = Install
Operation = Remove
Type = Package
Target = *

[Action]
When = PostTransaction
Exec = /bin/sh -c '/usr/bin/pacman -Qqe > /etc/packages.txt'
```

Listing all changed files from packages

If you are suspecting file corruption (e.g. by software/hardware failure), but are unsure if files were got corrupted, you might want to compare with the hash sums in the packages. This can be done with **`pacutils (https://www.archlinux.org/packages/?name=pacutils)`**:

```
# paccheck --md5sum --quiet
```

For recovery of the database see **#Restore pacman's local database**. The `mtree` files can also be **extracted as `.MTREE` from the respective package files**.

Note: This should **not** be used as is when suspecting malicious changes! In this case security precautions such as using a live medium and an independent source for the hash sums are advised.

Reinstalling all packages

To reinstall all native packages, use:

```
# pacman -Qnq | pacman -S -
```

Foreign (AUR) packages must be reinstalled separately; you can list them with `pacman -Qmq`.

Pacman preserves the **installation reason** by default.

Restore pacman's local database

See **Pacman/Restore local database**.

Recovering a USB key from existing install

If you have Arch installed on a USB key and manage to mess it up (e.g. removing it while it is still being written to), then it is possible to re-install all the packages and hopefully get it back up and working again (assuming USB key is mounted in `/newarch`)

```
# pacman -S $(pacman -Qq --dbpath /newarch/var/lib/pacman) --root /newarch --dbpath /newarch/v
ar/lib/pacman
```

Viewing a single file inside a .pkg file

For example, if you want to see the contents of `/etc/systemd/logind.conf` supplied within the **systemd (https://www.archlinux.org/packages/?name=systemd)** package:

```
$ tar -xOf /var/cache/pacman/pkg/systemd-204-3-x86_64.pkg.tar.xz etc/systemd/logind.conf
```

Or you can use **vim (https://www.archlinux.org/packages/?name=vim)** to browse the archive:

```
$ vim /var/cache/pacman/pkg/systemd-204-3-x86_64.pkg.tar.xz
```

Find applications that use libraries from older packages

Even if you installed a package the existing long-running programs (like daemons and servers) still keep using code from old package libraries. And it is a bad idea to let these programs running if the old library contains a security bug.

Here is a way how to find all the programs that use old packages code:

```
# lsof +c 0 | grep -w DEL | awk '1 { print $1 ": " $NF }' | sort -u
```

It will print running program name and old library that was removed or replaced with newer content.

Performance

Database access speeds

Pacman stores all package information in a collection of small files, one for each package. Improving database access speeds reduces the time taken in database-related tasks, e.g. searching packages and resolving package dependencies. The safest and easiest method is to run as root:

```
# pacman-optimize
```

This will attempt to put all the small files together in one (physical) location on the hard disk so that the hard disk head does not have to move so much when accessing all the data. This method is safe, but is not foolproof: it depends on your filesystem, disk usage and empty space fragmentation. Another, more aggressive, option would be to first remove uninstalled packages from cache and to remove unused repositories before database optimization:

```
# pacman -Sc && pacman-optimize
```

Download speeds

Note: If your download speeds have been reduced to a crawl, ensure you are using one of the many **mirrors** and not ftp.archlinux.org, which is **throttled since March 2007 (https://www.archlinux.org/news/302/)**.

When downloading packages *pacman* uses the mirrors in the order they are in `/etc/pacman.d/mirrorlist`. The mirror which is at the top of the list by default however may not be the fastest for you. To select a faster mirror, see **Mirrors**.

Pacman's speed in downloading packages can also be improved by using a different application to download packages, instead of *pacman*'s built-in file downloader.

In all cases, make sure you have the latest *pacman* before doing any modifications.

```
# pacman -Syu
```

Powerpill

Powerpill is a *pacman* wrapper that uses parallel and segmented downloading to try to speed up downloads for *pacman*.

wget

This is also very handy if you need more powerful proxy settings than *pacman*'s built-in capabilities.

To use `wget`, first **install** the **`wget (https://www.archlinux.org/packages/?name=wget)`** package then modify `/etc/pacman.conf` by uncommenting the following line in the `[options]` section:

```
XferCommand = /usr/bin/wget -c -q --show-progress --passive-ftp -O %o %u
```

Instead of uncommenting the `wget` parameters in `/etc/pacman.conf`, you can also modify the `wget` configuration file directly (the system-wide file is `/etc/wgetrc`, per user files are `$HOME/.wgetrc`.

aria2

aria2 is a lightweight download utility with support for resumable and segmented HTTP/HTTPS and FTP downloads. aria2 allows for multiple and simultaneous HTTP/HTTPS and FTP connections to an Arch mirror, which should result in an increase in download speeds for both file and package retrieval.

Note: Using aria2c in *pacman*'s XferCommand will **not** result in parallel downloads of multiple packages. *Pacman* invokes the XferCommand with a single package at a time and waits for it to complete before invoking the next. To download multiple packages in parallel, see **Powerpill**.

Install `aria2 (https://www.archlinux.org/packages/?name=aria2)`, then edit `/etc/pacman.conf` by adding the following line to the `[options]` section:

```
XferCommand = /usr/bin/aria2c --allow-overwrite=true --continue=true --file-allocation=none --
log-level=error --max-tries=2 --max-connection-per-server=2 --max-file-not-found=5 --min-split
-size=5M --no-conf --remote-time=true --summary-interval=60 --timeout=5 --dir=/ --out %o %u
```

Tip: **This alternative configuration for using *pacman* with aria2 (https://bbs.archlinux.org/viewtopic.php?pid=1491879#p1491879)** tries to simplify configuration and adds more configuration options.

See **OPTIONS (http://aria2.sourceforge.net/manual/en/html/aria2c.html#options)** in `aria2c(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/aria2c.1)` for used aria2c options.

- `-d, --dir` : The directory to store the downloaded file(s) as specified by *pacman*.
- `-o, --out` : The output file name(s) of the downloaded file(s).
- `%o` : Variable which represents the local filename(s) as specified by *pacman*.
- `%u` : Variable which represents the download URL as specified by *pacman*.

Other applications

There are other downloading applications that you can use with *pacman*. Here they are, and their associated XferCommand settings:

- `snarf` : `XferCommand = /usr/bin/snarf -N %u`
- `lftp` : `XferCommand = /usr/bin/lftp -c pget %u`
- `axel` : `XferCommand = /usr/bin/axel -n 2 -v -a -o %o %u`
- `hget` : `XferCommand = /usr/bin/hget %u -n 2 -skip-tls false` (please read the **documentation on the Github project page (https://github.com/huydx/hget)** for more info)

Utilities

- **Lostfiles** — Script that identifies files not owned by any package.

  https://github.com/graysky2/lostfiles || `lostfiles (https://aur.archlinux.org/packages/lostfiles/)`AUR

- **Pacmatic** — *Pacman* wrapper to check Arch News before upgrading, avoid partial upgrades, and warn about configuration file changes.

  http://kmkeen.com/pacmatic || `pacmatic (https://www.archlinux.org/packages/?name=pacmatic)`

- **pacutils** — Helper library for libalpm based programs.

  https://github.com/andrewgregory/pacutils || pacutils (https://www.archlinux.org/packages/?name=pacutils)

- **pkgfile** — Tool that finds what package owns a file.

  http://github.com/falconindy/pkgfile || pkgfile (https://www.archlinux.org/packages/?name=pkgfile)

- **pkgtools** — Collection of scripts for Arch Linux packages.

  https://github.com/Daenyth/pkgtools || pkgtools (https://aur.archlinux.org/packages/pkgtools/)AUR

- **repoctl** — Tool to help manage local repositories.

  https://github.com/cassava/repoctl || repoctl (https://aur.archlinux.org/packages/repoctl/)AUR

- **repose** — An Arch Linux repository building tool.

  https://github.com/vodik/repose || repose (https://www.archlinux.org/packages/?name=repose)

- **snap-pac** — Make *pacman* automatically use snapper to create pre/post snapshots like openSUSE's YaST.

  https://github.com/wesbarnett/snap-pac || snap-pac (https://www.archlinux.org/packages/?name=snap-pac)

Graphical front-ends

Warning: Some front-ends such as octopi (https://aur.archlinux.org/packages/octopi/)AUR **[2] (https://github.com/aarnt/octopi/issues/134#issuecomment-142099266)** or pamac-aur (https://aur.archlinux.org/packages/pamac-aur/)AUR **[3] (https://github.com/manjaro/pamac/blob/master/data/systemd/pamac-mirrorlist.service#L9)** perform **partial upgrades** periodically.

- **Aarchup** — Fork of archup. Has the same options as archup plus a few other features. For differences between both please check **changelog (https://bbs.archlinux.org/viewtopic.php?id=119129)**.

  https://github.com/aericson/aarchup/ || aarchup (https://aur.archlinux.org/packages/aarchup/)AUR

- **Arch-Update** — Update indicator for Gnome-Shell.

  https://github.com/RaphaelRochet/arch-update || gnome-shell-extension-arch-update (https://aur.archlinux.org/packages/gnome-shell-extension-arch-update/)AUR

- **Arch-Update-Notifier** — Update indicator for KDE.

  https://github.com/I-Dream-in-Code/kde-arch-update-plasmoid || `plasma5-applets-kde-arch-update-notifier-git (https://aur.archlinux.org/packages/plasma5-applets-kde-arch-update-notifier-git/)`AUR

- **Discover** — A collection of package management tools for KDE, using PackageKit.

  https://projects.kde.org/projects/kde/workspace/discover || `discover (https://www.archlinux.org/packages/?name=discover)`

- **GNOME packagekit** — GTK based package management tool

  http://www.freedesktop.org/software/PackageKit/ || `gnome-packagekit (https://www.archlinux.org/packages/?name=gnome-packagekit)`

- **GNOME Software** — Gnome Software App. (Curated selection for GNOME)

  https://wiki.gnome.org/Apps/Software || `gnome-software (https://www.archlinux.org/packages/?name=gnome-software)`

- **kalu** — A small application that will add an icon to your systray and sit there, regularly checking if there's anything new for you to upgrade.

  https://jjacky.com/kalu/ || `kalu (https://aur.archlinux.org/packages/kalu/)`AUR

- **pcurses** — Package management in a curses frontend

  https://github.com/schuay/pcurses || `pcurses (https://www.archlinux.org/packages/?name=pcurses)`

- **PkgBrowser** — Application for searching and browsing Arch packages, showing details on selected packages.

  https://bitbucket.org/kachelaqa/pkgbrowser/wiki/Home || `pkgbrowser (https://aur.archlinux.org/packages/pkgbrowser/)`AUR

- **tkPacman** — Depends only on Tcl/Tk and X11, and interacts with the package database via the CLI of *pacman*.

  http://sourceforge.net/projects/tkpacman || `tkpacman (https://aur.archlinux.org/packages/tkpacman/)`AUR

Retrieved from "https://wiki.archlinux.org/index.php?title=Pacman/Tips_and_tricks&oldid=514251"
