

pacman tips

From ArchWiki

This is a collection of common tips for new pacman users.

[Related articles](#)

Contents

- 1 Cosmetic and Convenience
 - 1.1 Color output
 - 1.2 Shortcuts
 - 1.2.1 Configure the shell
 - 1.2.2 Usage
 - 1.2.3 Notes
 - 1.3 Operations and Bash syntax
- 2 Maintenance
 - 2.1 Listing installed packages with size
 - 2.2 Listing installed packages with version
 - 2.3 Identify files not owned by any package
 - 2.4 Removing orphaned packages
 - 2.5 Removing unused packages
 - 2.6 Removing everything but base group
 - 2.7 Listing official installed packages only
 - 2.8 Getting the dependencies list of several packages
 - 2.9 Getting the size of several packages
 - 2.10 Listing changed configuration files
 - 2.11 Listing all packages that nothing else depends on
 - 2.12 Backing up local database with systemd
- 3 Installation and recovery
 - 3.1 Installing packages from a CD/DVD or USB stick
 - 3.2 Custom local repository
 - 3.3 Network shared pacman cache
 - 3.3.1 Read-only cache
 - 3.3.2 Read-write cache
 - 3.3.3 Synchronize pacman package cache using BitTorrent Sync
 - 3.3.4 Preventing unwanted cache purges
 - 3.4 Backing up and retrieving a list of installed packages
 - 3.5 List installed packages that are not in a specified group or repository
 - 3.6 Reinstalling all packages
 - 3.7 Restore pacman's local database
 - 3.7.1 Log filter script
 - 3.7.2 Generating the package recovery list

[pacman](#)

[Improve pacman performance](#)

[Mirrors](#)

[Creating packages](#)

- 3.7.3 Performing the recovery
- 3.8 Recovering a USB key from existing install
- 3.9 Extracting contents of a .pkg file
- 3.10 Viewing a single file inside a .pkg file
- 3.11 Find applications that use libraries from older packages

Cosmetic and Convenience

Color output

As of version 4.1, Pacman has a color option. Uncomment the "Color" line in `pacman.conf`.

Shortcuts

The following instructions allow users to run some of the more common pacman commands without the need to type them fully via a script alias.

Configure the shell

Add the following examples, which work in both Bash and Zsh:

```
# Pacman alias examples
alias pacupg='sudo pacman -Syu'          # Synchronize with repositories and then upgrade package
alias pacdl='pacman -Sw'                  # Download specified package(s) as .tar.xz ball
alias pacin='sudo pacman -S'              # Install specific package(s) from the repositories
alias pacins='sudo pacman -U'             # Install specific package not from the repositories but
alias pacre='sudo pacman -R'              # Remove the specified package(s), retaining its configu
alias pacrem='sudo pacman -Rns'           # Remove the specified package(s), its configuration(s)
alias pacrep='pacman -Si'                 # Display information about a given package in the repos
alias pacreps='pacman -Ss'                # Search for package(s) in the repositories
alias pacloc='pacman -Qi'                 # Display information about a given package in the local
alias paclocs='pacman -Qs'                # Search for package(s) in the local database
alias paclo="pacman -Qdt"                 # List all packages which are orphaned
alias pacc="sudo pacman -Scc"             # Clean cache - delete all not currently installed packa
alias paclf="pacman -Ql"                  # List all files installed by a given package
alias pacown="pacman -Qo"                 # Show package(s) owning the specified file(s)
alias pacexpl="pacman -D --asexp"         # Mark one or more installed packages as explicitly inst
alias pacimpl="pacman -D --asdep"         # Mark one or more installed packages as non explicitly

# Additional pacman alias examples
alias pacupd='sudo pacman -Sy && sudo abs' # Update and refresh the local package and AB
alias pacinsd='sudo pacman -S --asdeps'    # Install given package(s) as dependencies
alias pacmir='sudo pacman -Syy'            # Force refresh of all package lists after up
```

The following commands could be useful, but also dangerous. Please, know perfectly what are you doing when you use them:

```
# dealing with the following message from pacman:
#
# error: couldnt lock database: file exists
# if you are sure a package manager is not already running, you can remove /var/lib/pacman/d
```

```
alias pacunlock="sudo rm /var/lib/pacman/db.lck" # Delete the lock file /var/lib/pacman/db.lck
alias paclock="sudo touch /var/lib/pacman/db.lck" # Create the lock file /var/lib/pacman/db.lck
```

Usage

Perform the respective commands by simply typing the alias name. For example, to synchronize with repositories and then upgrade packages that are out of date on the local system:

```
$ pacupg
```

Install packages from repositories:

```
$ pacin <package1> <package2> <package3>
```

Install a custom built package:

```
$ pacins /path/to/<package>
```

Completely remove a locally installed package:

```
$ pacrem <package>
```

Search for available packages in the repositories:

```
$ pacreps <keywords>
```

Display information about a package (e.g. size, dependencies) in the repositories:

```
$ pacrep <keywords>
```

Notes

The aliases used above are merely examples. By following the syntax samples above, rename the aliases as convenient. For example:

```
alias pacrem='sudo pacman -Rns'
alias pacout='sudo pacman -Rns'
```

In the case above, the commands `pacrem` and `pacout` both call your shell to execute the same command.

Operations and Bash syntax

In addition to pacman's standard set of features, there are ways to extend its usability through rudimentary Bash commands/syntax.

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

```
# pacman -S kde-{applets,theme,tools}
```

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

```
# pacman -S kde-{ui-{kde,kdemod},kdeartwork}
```

- Sometimes, `-s`'s builtin ERE 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-'
```

- pacman has the `-q` operand to hide the version column, so it is possible to query and reinstall packages with "compiz" as part of their name:

```
# pacman -S $(pacman -Qq | grep compiz)
```

- Or install all packages available in a repository (kde-unstable for example):

```
# pacman -S $(pacman -Slq kde-unstable)
```

Maintenance

House keeping, in the interest of keeping a clean system and following The Arch Way.

See also System maintenance.

Listing installed packages with size

You may want to get the list of installed packages sorted by size, which may be useful when freeing space on your hard drive.

- Use `pacsysclean` from pacman (<https://www.archlinux.org/packages/?name=pacman>) package.
- Install `expac` (<https://www.archlinux.org/packages/?name=expac>) and run `expac -s "%-30n %m" | sort -rhk 2`
- Invoke `pacgraph` with the `-c` option to produce a list of all installed packages

with their respective sizes on the system. `pacgraph`

(<https://www.archlinux.org/packages/?name=pacgraph>) is available from [community].

- List explicitly installed packages not in base or base-devel with size and description:
`expac -HM "%011m\t%-20n\t%10d" $(comm -23 <(pacman -Qgen|sort) <(pacman -Qgg base base-devel|sort)) | sort -n`
- DEPRECATED:
`pacman -Qi | egrep "^(Name|Installed Size)" | sed -e 'N;s/\n/ /' | awk '{ print $7, $3}' | sort -n`
 (Note that this will not work with recent versions of pacman that use human readable sizes in `pacman -Qi`)

Listing installed packages with version

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 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> | awk 'BEGIN { RS="\n" ; FS="/" } { print $2 }' | awk '{ if(NF > 0) print $1, $2 }'`
- Install `expac` (<https://www.archlinux.org/packages/?name=expac>) and run
`expac -s "%-30n %v"`
- List all packages with version and repo: Install `yaourt`
 (<https://aur.archlinux.org/packages/yaourt/>) and run `yaourt -Q`

Identify files not owned by any package

Periodic checks for files outside of pacman database are recommended. These files are often some 3rd party applications installed using the usual procedure (e.g. `./configure && make && make install`). Search the file-system for these files (or symlinks) using this simple script:

```
pacman-disowned
-----
#!/bin/sh

tmp=${TMPDIR-/tmp}/pacman-disowned-$UID-$$
db=$tmp/db
fs=$tmp/fs

mkdir "$tmp"
trap 'rm -rf "$tmp"' EXIT

pacman -Qlq | sort -u > "$db"

find /etc /opt /usr ! -name lost+found \( -type d -printf '%p\n' -o -print \) | sort > "$fs"

comm -23 "$fs" "$db"
```

To generate the list:

```
$ pacman-disowned > non-db.txt
```

Note that one should **not** delete all files listed in `non-db.txt` without confirming each entry. There could be various configuration files, logs, etc., so use this list responsibly and only proceed after extensively searching for cross-references using `grep`.

Here are some one-liner scripts that will be helpful.

Show dirs that do not belong to any package:

```
alias pacman-disowned-dirs="comm -23 <(sudo find / \( -path '/dev' -o -path '/sys' -o -path '/r
```

Show files that do not belong to any package:

```
alias pacman-disowned-files="comm -23 <(sudo find / \( -path '/dev' -o -path '/sys' -o -path '/r
```

Removing orphaned packages

For *recursively* removing orphans and their configuration files:

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

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

The following **alias** is easily inserted into `~/.bashrc` and removes orphans if found:

```
~/.bashrc
```

```
# '[r]emove [o]rphans' - recursively remove ALL orphaned packages
alias pacro="/usr/bin/pacman -Qtdq > /dev/null && sudo /usr/bin/pacman -Rns \$(/usr/bin/pacman -
```

The following **function** is easily inserted into `~/.bashrc` and removes orphans if found:

```
~/.bashrc
```

```
orphans() {
    if [[ ! -n $(pacman -Qdt) ]]; then
        echo "No orphans to remove."
    else
```

```

} sudo pacman -Rns $(pacman -Qdtq)
fi
}
```

Note: The above scripts and commands have limitations as they do not take into account optional dependencies. This bears the risk to remove packages which actually are not orphans. To make sure that only real orphans are removed, use `pkg-list_true_orphans` (http://xyne.archlinux.ca/projects/pkg_scripts/#help-message-pkg-list_true_orphans) from the package `pkg_scripts` (https://aur.archlinux.org/packages/pkg_scripts/).

Removing unused packages

Because a lighter system is easier to maintain, occasionally looking through explicitly installed packages and *manually* selecting unused packages to be removed can be helpful.

To list explicitly installed packages available in the official repositories:

```
$ pacman -Qen
```

To list explicitly installed packages not available in official repositories:

```
$ pacman -Qem
```

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 -Qgg base); do pactree -ul $i; do
```

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

Notes:

1. `comm` requires sorted input otherwise you get e.g.
`comm: file 1 is not in sorted order .`
2. `pactree` prints the package name followed by what it provides. For example:

```
$ pactree -lu logrotate
```

```
logrotate
popt
glibc
```

```
linux-api-headers
tzdata
dcron cron
bash
readline
ncurses
gzip
```

The `dcron cron` line seems to cause problems, that is why `cut -d ' ' -f 1` is needed - to keep just the package name.

Listing official installed packages only

```
$ pacman -Qqn
```

This list packages that are found in the sync database(s). If the user has unofficial repositories configured, it will list packages from such repositories too.

Getting the dependencies list of several packages

Dependencies are alphabetically sorted and doubles are removed. Note that you can use `pacman -Qi` to improve response time a little. But you will not be able to query as many packages. Unfound packages are simply skipped (hence the `2>/dev/null`). You can get dependencies of AUR packages as well if you use `yaourt -Si`, but it will slow down the queries.

```
$ pacman -Si $@ 2>/dev/null | awk -F ": " -v filter="^Depends" \ ' $0 ~ filter {gsub(/[>=<][^ ]*/
```

Alternatively, you can use `expac`: `expac -l '\n' %E -S $@ | sort -u`.

Getting the size of several packages

You can use (and tweak) this little shell function:

```
~/.bashrc
```

```
pacman-size()
{
    CMD="pacman -Si"
    SEP=": "
    TOTAL_SIZE=0

    RESULT=$(eval "${CMD} $@ 2>/dev/null" | awk -F "$SEP" -v filter="Size" -v pkg="^Name" \
        '$0 ~ pkg {pkgname=$2} $0 ~ filter {gsub(/\..*/,"") ; printf("%6s KiB %s\n", $2, pkgname)')

    echo "$RESULT"

    ## Print total size.
    echo "$RESULT" | awk '{TOTAL=$1+TOTAL} END {printf("Total : %d KiB\n",TOTAL)}'
```


As told for the dependencies list, you can use `pacman -Qi` instead, but not `yaourt` since AUR's PKGBUILD do not have size information.

A nice one-liner:

```
$ pacman -Si "$@" 2>/dev/null | awk -F ": " -v filter="Size" -v pkg="Name" '$0 ~ pkg {pkgname=$2
```

You should replace `"$@"` with packages, or put this line in a shell function.

Listing changed configuration 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. In this case you want to list those changed configuration files, we can do this with the following command:

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

The following script does the same. You need to run it as root or with `sudo`.

```
changed-files.sh
```

```
#!/bin/bash
for package in /var/lib/pacman/local/*; do
    sed '/^%BACKUP%$/,/^%/!d' $package/files | tail -n+2 | grep -v '^$' | while read file ha
        [ "$(md5sum /$file | (read hash file; echo $hash))" != "$hash" ] && echo $(basename
done
```

Listing all packages that nothing else depends on

If you want to generate a list of all installed packages that nothing else depends on, you can use the following script. This is very helpful if you are trying to free hard drive space and have installed a lot of packages that you may not remember. You can browse through the output to find packages which you no longer need.

Note: This script will show all packages that nothing else depends on, including those explicitly installed. To get a list of packages installed as dependencies but no longer required by any installed package, see [#Removing orphaned packages](#).

```
clean
```

```
#!/bin/bash

# This script is designed to help you clean your computer from unneeded
# packages. The script will find all packages that no other installed package
# depends on. It will output this list of packages excluding any you have
```

```
# placed in the ignore list. You may browse through the script's output and
# remove any packages you do not need.

# Enter groups and packages here which you know you wish to keep. They will
# not be included in the list of unrequired packages later.
ignoregrp="base base-devel"
ignorepkg=""

comm -23 <(pacman -Qqt | sort) <(echo $ignorepkg | tr ' ' '\n' | cat <(pacman -Sqq $ignoregrp)
```

For list with descriptions for packages:

```
expac -HM "%-20n\t%10d" $( comm -23 <(pacman -Qqt|sort) <(pacman -Qqq base base-devel|sort) )
```

Backing up Local database with systemd

systemd can take snapshots of the pacman local database everytime it is modified.

Note: There is a more configurable version in the AUR: `pakbak-git`
(<https://aur.archlinux.org/packages/pakbak-git/>)

Tip: Save the following script as `/usr/lib/systemd/scripts/pakbak_script`.

Note: Change the value of `$pakbak` to modify where the backed up database is stored.

```
#!/bin/bash

declare -r pakbak="/pakbak.tar.xz"; ## set backup location
tar -cJf "$pakbak" "/var/lib/pacman/local"; ## compress & store pacman local database in $pakbak
```

Tip: Save the following service file as `/usr/lib/systemd/system/pakbak.service`.

```
[Unit]
Description=Back up pacman database

[Service]
Type=oneshot
ExecStart=/bin/bash /usr/lib/systemd/scripts/pakbak_script
RemainAfterExit=no
```

Tip: Save the following path file as `/usr/lib/systemd/system/pakbak.path`.

```
[Unit]
Description=Back up pacman database

[Path]
PathChanged=/var/lib/pacman/local
Unit=pakbak.service

[Install]
```

```
WantedBy=multi-user.target
```

Tip: To start the backup service :

```
# systemctl start pakebak.path
```

To enable the backup service automatically on reboot :

```
# systemctl enable pakebak.path
```

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 -Sy
```

Custom local repository

pacman 3 introduced a new script named `repo-add` which makes generating a database for a personal repository much easier. Use `repo-add --help` for more details on its usage.

Simply store all of the built packages to be included in the repository in one directory, and execute the following command (where *repo* is the name of the custom repository):

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

Note that when using `repo-add`, the database and the packages do not need to be in the same directory. But when using `pacman` with that database, they should be together.

To add a new package (and remove the old if it exists), run:

```
$ repo-add /path/to/repo.db.tar.gz /path/to/packageToAdd-1.0-1-i686.pkg.tar.xz
```

Note: If there is a package that needs to be removed from the repository, read up on `repo-remove`.

Once the local repository has been made, add the repository to `pacman.conf`. The name of the `db.tar.gz` file is the repository name. Reference it directly using a `file://` url, or access it 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

Read-only cache

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

`darkhttpd /var/cache/pacman/pkg`. Just add this server at the top of your mirror list. Be aware that you might get a lot of 404 errors, due to cache misses, depending on what you do, but `pacman` will try the next (real) mirrors when that happens.

Read-write cache

Tip: See `pacserve` for an alternative (and probably simpler) solution than what follows.

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; for example `sshfs`, `shfs`, `ftpfs`, `smbfs` or `nfs`.

Tip: To use `sshfs` or `shfs`, consider reading [Using SSH Keys](#).

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

Synchronize pacman package cache using BitTorrent Sync

BitTorrent Sync is a new way of synchronizing folder via network (it works in LAN and over the internet). It is peer-to-peer so you do not need to set up a server: follow the link for more information. How to share a pacman cache using BitTorrent Sync:

- First install the `btsync` (<https://aur.archlinux.org/packages/btsync/>) package from the AUR on the machines you want to sync
- Follow the installation instructions of the AUR package or on the BitTorrent Sync wiki page
 - set up BitTorrent Sync to work for the root account. This process requires read/write to the pacman package cache.
 - make sure to set a good password on `btsync`'s web UI
 - start the `systemd` daemon for `btsync`.
 - in the `btsync` Web GUI add a new synchronized folder on the first machine and generate a new Secret. Point the folder to `/var/cache/pacman/pkg`
 - Add the folder on all the other machines using the same Secret to share the cached packages between all systems. Or, to set the first system as a master and the others as slaves, use the Read Only Secret. Be sure to point it to `/var/cache/pacman/pkg`

Now the machines should connect and start synchronizing their cache. Pacman works as expected even during synchronization. The process of syncing is entirely automatic.

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
```

Backing up and retrieving a list of installed packages

It is good practice to keep periodic backups of all pacman-installed packages. In the event of a system crash which is unrecoverable by other means, pacman can then easily reinstall the very same packages onto a new installation.

- First, backup the current list of non-local packages:
`$ pacman -Qgen > pkglist.txt`
- Store the `pkglist.txt` on a USB key or other convenient medium or `gist.github.com` or Evernote, Dropbox, etc.
- Copy the `pkglist.txt` file to the new installation, and navigate to the directory containing it.
- Issue the following command to install from the backup list:
`# pacman -S $(< pkglist.txt)`

In the case you have a list which was not generated like mentioned above, there may be foreign packages in it (i.e. packages not belonging to any repos you have configured, or packages from the AUR).

In such a case, you may still want to install all available packages from that list:

```
# pacman -S --needed $(comm -12 <(pacman -Slq|sort) <(sort badpkgdlist) )
```

Explanation:

- `pacman -Slq` lists all available softwares, but the list is sorted by repository first, hence the `sort` command.
- Sorted files are required in order to make the `comm` command work.
- The `-12` parameter display lines common to both entries.
- The `--needed` switch is used to skip already installed packages.

You may also try to install all unavailable packages (those not in the repos) from the AUR using `yaourt` (not recommended unless you know exactly what you are doing):

```
$ yaourt -S --needed $(comm -13 <(pacman -Slq|sort) <(sort badpkgdlist) )
```

Finally, you may want to remove all the packages on your system that are not mentioned in the list.

Warning: Use this command wisely, and always check the result prompted by pacman.

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

List installed packages that are not in a specified group or repository

The following command will list any installed packages that are not in either base (https://www.archlinux.org/groups/x86_64/base/) or base-devel (https://www.archlinux.org/groups/x86_64/base-devel/), and as such were likely installed manually by the user:

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

List all installed packages that are not in specified repository (*repo_name* in example):

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

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

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

Reinstalling all packages

To reinstall all native packages, use:

```
# pacman -Qeq | pacman -S -
```

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

Pacman preserves the installation reason by default.

Restore pacman's local database

Signs that pacman needs a local database restoration:

- `pacman -Q` gives absolutely no output, and `pacman -Syu` erroneously reports that the system is up to date.
- When trying to install a package using `pacman -S package`, and it outputs a list of already satisfied dependencies.
- When `testdb` (part of `pacman` (<https://www.archlinux.org/packages/?name=pacman>)) reports database inconsistency.

Most likely, pacman's database of installed software, `/var/lib/pacman/local`, has been corrupted or deleted. While this is a serious problem, it can be restored by following the instructions below.

Firstly, make sure pacman's log file is present:

```
$ ls /var/log/pacman.log
```

If it does not exist, it is *not* possible to continue with this method. You may be able to use Xyne's package detection script (<https://bbs.archlinux.org/viewtopic.php?pid=670876>) to recreate the database. If not, then the likely solution is to re-install the entire system.

Log filter script

```
pacrecover
#!/bin/bash -e
. /etc/makepkg.conf
PKG_CACHE=$(grep -m 1 '^CacheDir' /etc/pacman.conf || echo 'CacheDir = /var/cache/pacman/pkg')
pkgdirs=("$@" "${PKG_CACHE}" "${PKGDEST}")
while read -r -a parampart; do
    pkgname="${parampart[0]}-${parampart[1]}-*.pkg.tar.xz"
    for pkgdir in ${pkgdirs[@]}; do
        pkgpath="$pkgdir/$pkgname"
        [ -f $pkgpath ] && { echo $pkgpath; break; };
    done || echo ${parampart[0]} 1>&2
done
```

Make the script executable:

```
$ chmod +x pacrecover
```

Generating the package recovery list

Warning: If for some reason your pacman cache or makepkg package destination contain packages for other architectures, remove them before continuation.

Run the script (optionally passing additional directories with packages as parameters):

```
$ paclog-pkglist /var/log/pacman.log | ./pacrecover >files.list 2>pkglist.orig
```


This way two files will be created: `files.list` with package files, still present on machine and `pkglist.orig`, packages from which should be downloaded. Later operation may result in mismatch between files of older versions of package, still present on machine, and files, found in new version. Such mismatches will have to be fixed manually.

Here is a way to automatically restrict second list to packages available in a repository:

```
$ { cat pkglist.orig; pacman -Slq; } | sort | uniq -d > pkglist
```

Check if some important *base* package are missing, and add them to the list:

```
$ comm -23 <(pacman -Sgq base) pkglist.orig >> pkglist
```

Proceed once the contents of both lists are satisfactory, since they will be used to restore pacman's installed package database; `/var/lib/pacman/local/`.

Performing the recovery

Define bash alias for recovery purposes:

```
# recovery-pacman() {
    pacman "$@" \
    --log /dev/null \
    --noscriptlet \
    --dbonly \
    --force \
    --nodeps \
    --needed \
    #
}
```

`--log /dev/null` allows to avoid needless pollution of pacman log, `--needed` will save some time by skipping packages, already present in database, `--nodeps` will allow installation of cached packages, even if packages being installed depend on newer versions. Rest of options will allow **pacman** to operate without reading/writing filesystem.

Populate the sync database:

```
# pacman -Sy
```

Start database generation by installing locally available package files from `files.list`:

```
# recovery-pacman -U $(< files.list)
```

Install the rest from `pkglist`:

```
# recovery-pacman -S $(< pkglist)
```

Update the local database so that packages that are not required by any other package are marked as explicitly installed and the other as dependences. You will need be extra careful in the future when removing packages, but with the original database lost is the best we can do.

```
# pacman -D --asdeps $(pacman -Qq)
# pacman -D --asexplicit $(pacman -Qtq)
```

Optionally check all installed packages for corruption:

```
# pacman -Qk
```

Optionally #Identify files not owned by any package.


Update all packages:

```
# pacman -Su
```

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/var
```



Extracting contents of a .pkg file

The .pkg files ending in .xz are simply tar'ed archives that can be decompressed with:

```
$ tar xvf package.tar.xz
```

If you want to extract a couple of files out of a .pkg file, this would be a way to do it.

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>), then 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.

Retrieved from "https://wiki.archlinux.org/index.php?title=Pacman_tips&oldid=350406"
Category: Package management

-
- This page was last modified on 24 December 2014, at 22:53.
 - Content is available under GNU Free Documentation License 1.3 or later unless otherwise noted.