pacman/Tips and tricks (Русский)

< Pacman

Ссылки по теме

- **Зеркала**
- **Создание пакетов**

 **Эта страница нуждается в сопроводителе** 

Статья не гарантирует актуальность информации. Помогите русскоязычному сообществу поддержкой подобных страниц. См. **Команда переводчиков ArchWiki**

Состояние перевода: На этой странице представлен перевод статьи **pacman/Tips and tricks**. Дата последней синхронизации: 19 марта 2016. Вы можете **помочь** синхронизировать перевод, если в английской версии произошли **изменения**.

Для общих методов улучшения гибкости предоставляемых советов или самого Pacman смотрите **Базовые утилиты** и **Bash**.

Contents

[hide]

Красота и комфорт

Графические оболочки

- **Arch-Update** — Индикатор обновлений для Gnome-Shell.
https://github.com/RaphaelRochet/arch-update || **`gnome-shell-extension-arch-update`**[AUR]

- **Discover** — Утилита управления пакетами для KDE, используя PackageKit.
https://projects.kde.org/projects/kde/workspace/discover || **`discover`**

- **GNOME packagekit** — Утилита для управления пакетами, на основе GTK.
http://www.freedesktop.org/software/PackageKit/ || **`gnome-packagekit`**

- **GNOME Software** — Центр приложений Gnome.
https://wiki.gnome.org/Apps/Software || **`gnome-software`**

- **kalu** — Маленькое приложение, добавляющее иконку в трей, которое будет регулярно проверять доступные обновления.
https://jjacky.com/kalu/ || **`kalu`**[AUR]

- **pcurses** — Управление пакетами при помощи текстового интерфейса curses.
https://github.com/schuay/pcurses || **`pcurses`**

- **tkPacman** — Зависит от Tcl/Tk и X11. Взаимодействует только с базой данных пакетов через интерфейс командной строки *pacman*.
http://sourceforge.net/projects/tkpacman || **`tkpacman`**[AUR]

Утилиты

- **Lostfiles** — Скрипт для обнаружения файлов, не принадлежащих ни одному пакету.
https://github.com/graysky2/lostfiles || **`lostfiles`**

- **Pacmatic** — Оболочка для Pacman, проверяющая новости Arch перед обновлением, во избежание частичных обновлений, и предупреждающая об изменении файлов настроек.
http://kmkeen.com/pacmatic || **`pacmatic`**

- **pacutils** — Библиотека для программ, основанных на libalpm.
https://github.com/andrewgregory/pacutils || **`pacutils`**

- **pkgfile** — Утилита, которая находит какому пакету принадлежит файл.
http://github.com/falconindy/pkgfile || **`pkgfile`**

- **pkgtools** — Коллекция скриптов для пакетов Arch Linux. https://github.com/Daenyth/pkgtools || pkgtools[AUR]

- **repoctl** — Утилита для управления локальными репозиториями. https://github.com/cassava/repoctl || repoctl[AUR]

- **repose** — Программа для создания репозиторев Arch Linux. https://github.com/vodik/repose || repose

- **snap-pac** — Заставить pacman автоматически использовать snapper для создания снимков до/после, как в openSUSE'овском YaST. https://github.com/wesbarnett/snap-pac || snap-pac

Обслуживание

Смотрите также **Обслуживание системы**.

Список пакетов

Вы можете получить список установленных пакетов с их версией, что полезно при составлении отчетов об ошибках или обсуждении установленных пакетов.

- Список всех явно установленных пакетов: `pacman -Qe`.
- Список всех пакетов в группе `group`: `pacman -Sg group`.
- Список всех явно установленных родных пакетов (то есть присутствующие в синхронизируемой базе), которые не являются прямыми или дополнительными зависимостями: `pacman -Qent`.
- Список всех внешних пакетов (обычно загруженных и установленных вручную): `pacman -Qm`.
- Список всех родных пакетов (установленных из синхронизированной(ых) баз(ы)): `pacman -Qn`.
- Список пакетов по регулярному выражению (regex): `pacman -Qs regex`.
- Список пакетов по регулярному выражению с пользовательским форматом вывода: `expac -s "%-30n %v" regex` (требуется **expac**).

С размером

Чтобы получить список установленных пакетов, отсортированный по размеру, который может быть полезен, когда необходимо освободить пространство на жестком диске. Есть два варианта: получение размера индивидуальных пакетов и получение размера пакета с его зависимостями.

Индивидуальные пакеты

Следующая команда покажет все установленные пакеты и их размер:

```
$ pacman -Qi | awk '/^Name/{name=$3} /^Installed Size/{print $4$5, name}' | sort -h
```

Пакеты с зависимостями

Чтобы получить список размеров пакетов с их зависимостями:

- Установите **expac** и выполните `expac -H M '%m\t%n' | sort -h`.
- Запустите **pacgraph** с опцией `-c`.

Чтобы перечислить размер нескольких загружаемых пакетов (оставьте *packages* пустым, чтобы перечислить все пакеты):

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

Чтобы получить список явно установленных пакетов не из **base** и не из **base-devel** с размерами и описанием:

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

Чтобы получить список пакетов, которым требуется обновление, с загружаемым размером:

```
$ pacman -Quq|xargs expac -S -H M '%k\t%n' | sort -sh
```

По дате

Чтобы увидеть список последних 20 установленных пакетов при помощи **expac**:

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

или используя секунды с начала эпохи (1970-01-01 UTC):

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

Не принадлежащие определенной группе или репозиторию

Примечание: Чтобы получить список пакетов, установленных ранее как зависимости, но теперь никому не принадлежащие, смотрите **#Удаление неиспользуемых пакетов**.

Следующая команда выведет список всех установленных пакетов, которые не принадлежат **base** или **base-devel**:

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

Список всех установленных пакетов, которые не зависят от других пакетов и которые не принадлежат **base** или **base-devel**:

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

Как выше, но с описаниями:

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

Список всех установленных пакетов, которые *не* из репозитория *имя_репозитория*:

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

Список всех установленных пакетов, которые находятся в репозитории *имя_репозитория* :

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

Находящиеся в разработке

Чтобы получить список всех пакетов, которые находятся в разработке или которые нестабильны:

```
$ pacman -Qq | grep -Ee '-(cvs|svn|git|hg|bzr|darcs)$'
```

Просмотр файлов, принадлежащих пакету с определенным размером

Это может пригодиться, если вы обнаружили, что конкретный пакет занимает много места, и вы хотите выяснить, какие файлы весят больше всего.

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

Поиск файлов, не принадлежащих ни одному пакету

Если в вашей системе присутствуют "беспризорные" файлы, не принадлежащие ни одному пакету (например, в случае установки программного обеспечения в обход пакетного менеджера), вам может потребоваться найти такие файлы для очистки системы. В таком случае необходимо выполнить следующую последовательность действий:

1. Создайте отсортированный список файлов, чью принадлежность конкретному пакету вы хотите проверить:

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

2. Создайте отсортированный список файлов, отслеживаемых пакетным менеджером (с удалением замыкающих слэшей из названий директорий):

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

3. Найдите строки в первом списке, отсутствующие во втором:

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

На практике применение этого приёмы осложняется тем, что многие важные файлы не принадлежат какому-либо пакету (к примеру, файлы, сгенерированные в процессе выполнения каких-либо приложений, пользовательские конфигурационные файлы, и т. д.) и будут включены в список "беспризорных", поэтому не стоит слепо полагаться исключительно на автоматику и пренебрегать ручной очисткой системы.

Совет: Скрипт **lostfiles** совершает схожие действия, но дополнительно сверяется с встроенным обширным перечнем файлов, необходимых системе, но не принадлежащих ни одному пакету. **aconfmgr** (**aconfmgr-git**[AUR]) также позволяет отслеживать беспризорные файлы.

Удаление неиспользуемых пакетов

Для *рекурсивного* удаления пакетов-сирот, от которых не зависят другие пакеты, и их конфигурационных файлов:

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

Если сироты не найдены, pacman завершит работу с ошибкой `error: no targets specified`. Это так же работает, как если бы никакие аргументы не были переданы в `pacman -Rns`.

Примечание: Посредством опции `-Qt` выводятся только настоящие сироты. Для добавления в список также пакетов, от которых опционально зависят другие пакеты, продублируйте флаг `-t`, то есть `-Qtt`.

Удаление всех пакетов, кроме группы base

Если существует необходимость удалить все пакеты, кроме принадлежащих к группе *base*, это можно сделать командой в одну строку:

```
# pacman -R $(comm -23 <(pacman -Qq|sort) <((for i in $(pacman -Qqg base); do pactree -ul $i; done)|sort -u|cut -d ' ' -f 1))
```

Этак команда была изначально разработана в **this discussion** и позднее доработана для этой статьи.

Получения списка зависимостей нескольких пакетов

Зависимости в алфавитном порядке без двойников.

Примечание: Используйте `pacman -Qi`, чтобы показать дерево локально установленных пакетов.

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

Или посредством **expac**:

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

Построение списка изменённых файлов из резервного копирования

Если вы хотите сделать резервное копирование ваших системных конфигурационных файлов, вы можете копировать все файлы из `/etc/`, но, скорее всего, вас интересуют только те файлы, в которых были сделаны изменения. Список модифицированных **заархивированных файлов** может быть построен следующей командой:

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

Запуск этой команды с root-привилегиями даёт гарантию того, что файлы, доступные для просмотра только от root-а (такие, как `/etc/sudoers`), будут включены в список.

Совет: Для построения списка всех изменённых файлов из базы данных Pacman (не только из резервного копирования) смотрите **#Listing all changed files from packages**.

Создание резервной копии базы данных Pacman

Следующая команда может быть использована для резервного копирования локальной базы данных Pacman:

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

Храните заархивированную базу данных на одном или нескольких внешних хранилищах, таких, как USB-накопитель, внешний жёсткий диск или диск CD-R.

База данных может быть восстановлена путём перемещения `pacman_database.tar.bz2` в `/` и последующего выполнения команды:

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

Примечание: Если файл с базой данных Pacman повреждён и нет других доступных резервных копий, есть надежда исправить ситуацию путём пересоздания базы данных. Подробнее смотрите тут: **Pacman tips#Restore pacman's local database**.

Совет: Пакет **pakbak-git**[AUR] предоставляет скрипт и сервис **systemd** для автоматизации этой задачи. Конфигурация доступна в `/etc/pakbak.conf`.

Лёгкий способ проверки списка изменений

Когда мейнтейнеры обновляют пакеты, правки часто комментируются в удобной форме. Пользователи могут легко просмотреть их из командной строки с помощью **paclog**[AUR][ссылка недействительна: package not found]. Эта утилита выводит список последних комментариев мейнтейнеров к правкам для пакетов из официальных репозиториев или AUR, используя `paclog package`.

Установка и восстановление

Альтернативные способы получения и восстановления пакетов.

Установка пакетов с CD/DVD или USB накопителя

Чтобы установить пакеты или группы пакетов, необходимо выполнить:

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

После этого необходимо записать папку "Packages" на CD/DVD или переместить ее на USB накопитель, внешний HDD, и т.д.

Установка:

1. Примонтируйте носитель:

```
# mkdir /mnt/repo
# mount /dev/sr0 /mnt/repo    #Для CD/DVD.
# mount /dev/sdxY /mnt/repo   #Для USB накопителя.
```

2. Отредактируйте `pacman.conf` и добавьте этот репозиторий *перед* всеми остальными (например extra, core, etc.). Это необходимо сделать для того, чтобы файлы с CD/DVD/USB смогли получить приоритет над теми, которые находятся в стандартных репозиториях. Ни в коем случае не надо оставлять раскомментированный репозиторий в самом низу:

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

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

3. В заключение, синхронизируйте базу данных pacman, чтобы получить возможность использовать подключенный репозиторий:

```
# pacman -Syu
```

Собственный локальный репозиторий

Используйте скрипт *repo-add*, идущий в комплекте с Pacman, чтобы сгенерировать базу данных для персонального репозитория. Используйте `repo-add --help` для более подробной информации. Чтобы добавить новый пакет в базу данных или заменить старую версию существующего, запустите:

```
$ repo-add /путь/к/repo.db.tar.gz /путь/к/имя_пакета-1.0-1-x86_64.pkg.tar.xz
```

Примечание: База данных пакетов - tar файл, обычно сжатый. Он должен иметь расширение ".db" или ".files", следующее после расширения архива ".tar", ".tar.gz", ".tar.bz2", ".tar.xz" или ".tar.Z". Не обязательно, чтобы сам файл существовал, но должны существовать все родительские каталоги.

Кроме того, при использовании `repo-add` учтите, что база данных и пакеты не обязательно должны располагаться в одном каталоге. Но при использовании pacman с этой базой данных, убедитесь, что они (база данных и пакеты) расположены рядом. Хранение всех собранных пакетов, включенных в репозиторий, в одной директории позволяет использовать glob расширение командной оболочки. Чтобы добавить или обновить несколько пакетов сразу:

```
$ repo-add /путь/к/repo.db.tar.gz /путь/к/*.pkg.tar.xz
```

Важно: *repo-add* добавляет записи в базу данных в том порядке, в каком они в командной строке. Если используются несколько версий одно пакета, следует следить за тем, чтобы самая свежая версия была добавлена последней. Следует отметить, что лексический порядок, используемый командной средой, зависит от локали и отличается от порядка **vercmp**, который использует pacman


Эта статья или раздел нуждается в переводе


Примечания: Сомнительная информация в оригинале. Согласно справочной странице, *repo-remove* принимает *имя пакета* вместо пакета. (обсуждение: **Talk:Pacman/Tips and tricks (Русский)#**)

repo-remove используется точно таким же образом, как и *repo-add*, но только для удаления выбранных пакетов из базы данных репозитория.

После того, как база данных локального репозитория создана, добавьте репозиторий в `pacman.conf` для каждой системы, которая будет использовать его. Пример собственного репозитория также располагается в файле `pacman.conf`. Имя репозитория это имя файла базы данных (без расширения). Например, для этого примера именем репозитория будет просто *repo*. Для указания расположения репозитория используется url `file://` или FTP **ftp://localhost/путь/до/каталога**.

При желании добавьте свой репозиторий в **список пользовательских репозиториев**, чтобы сообщество также могло использовать его.

Общий кэш pacman

Если у вас несколько машин с Arch в локальной сети, то вы можете обмениваться пакетами между ними, что значительно уменьшит время загрузки. Имейте ввиду, вы не должны делиться пакетами между разными архитектурами (то есть i686 и x86_64).

Только для чтения

Если вы ищете быстрое решение, то можете просто запустить сервер, который смогут использовать другие компьютеры, как первое зеркало: `darkhttpd /var/cache/pacman/pkg`. Просто добавьте этот сервер в верхней части списка зеркал. Помните, что вы можете получить много ошибок 404 из-за отсутствия кэша, но pacman попробует следующие (настоящие) зеркала, когда это случится.

Чтение-запись

Совет: Смотрите **pacserve** для альтернативного (и, возможно, более простого) решения.

Чтобы обмениваться пакетами между несколькими компьютерами, просто расшарьте `/var/cache/pacman/`, используя любой протокол монтирования на основе сети. Этот раздел показывает, как использовать shfs или sshfs, чтобы передавать кэш пакетов плюс связанные библиотеки-директории между несколькими компьютерами в локальной сети. Имейте в виду, что общий сетевой кэш может быть медленным в зависимости от выбранной файловой системы.

Сначала установите любую сетевую файловую систему, например **SSHFS**, **shfs**, ftpfs, **smbfs** или **NFS**.

Совет: Чтобы использовать sshfs или shfs, вам, возможно, стоит прочитать **Using SSH Keys**.

Затем, чтобы поделиться текущими пакетами, примонтируйте `/var/cache/pacman/pkg` с сервера в `/var/cache/pacman/pkg` на каждой компьютере клиента.

Примечание: Не делайте `/var/cache/pacman/pkg` или любого из его родителей (например, `/var`) символической ссылкой. Pacman ожидает, что это будут директории. Когда pacman переустановит или обновит себя, он удалит символические ссылки и создаст вместо них пустые директории. Однако во время операции pacman опирается на некоторые файлы, расположенные там, что нарушает процесс обновления. Подробнее: **FS#50298**.

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.

Важно: 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 our dynamic cache (read the comments for an explanation of the commands):

```
/etc/nginx/nginx.conf

http
{
```

```
    ...

    # nginx may need to resolve domain names at run time
    resolver 8.8.8.8 8.8.4.4;

    # Pacman Cache
    server
    {
        listen      8080;
        server_name cache.domain.local;
        root        /srv/http/pacman-cache;
        autoindex   on;

        # Requests for package db and signature files should redirect
upstream without caching
        location ~ \.(db|sig)$ {
            proxy_pass http://mirrors$request_uri;
        }

        # Requests for actual packages should be served directly from cache
if available.
        #   If not available, retrieve and save the package from an upstream
mirror.
        location ~ \.tar\.xz$ {
            try_files $uri @pkg_mirror;
        }

        # Retrieve package from upstream mirrors and cache for future
requests
        location @pkg_mirror {
            proxy_store    on;
            proxy_redirect off;
            proxy_store_access  user:rw group:rw all:r;
            proxy_next_upstream error timeout http_404;
            proxy_pass          http://mirrors$request_uri;
        }
    }

    # Upstream Arch Linux Mirrors
    # - Configure as many backend mirrors as you want in the blocks below
    # - Servers are used in a round-robin fashion by nginx
    # - Add "backup" if you want to only use the mirror upon failure of the
other mirrors
    # - Separate "server" configurations are required for each upstream
mirror so we can set the "Host" header appropriately
    upstream mirrors {
```

```
        server localhost:8001;
        server localhost:8002 backup;
        server localhost:8003 backup;
    }

    # Arch Mirror 1 Proxy Configuration
    server
    {
        listen      8001;
        server_name localhost;

        location / {
            proxy_pass       http://mirror.rit.edu$request_uri;
            proxy_set_header Host mirror.rit.edu;
        }
    }

    # Arch Mirror 2 Proxy Configuration
    server
    {
        listen      8002;
        server_name localhost;

        location / {
            proxy_pass       http://mirrors.acm.wpi.edu$request_uri;
            proxy_set_header Host mirrors.acm.wpi.edu;
        }
    }

    # Arch Mirror 3 Proxy Configuration
    server
    {
        listen      8003;
        server_name localhost;

        location / {
            proxy_pass       http://lug.mtu.edu$request_uri;
            proxy_set_header Host lug.mtu.edu;
        }
    }
}
```

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
...
```

Примечание: 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 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**AUR[ссылка недействительна: package not found] 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
```

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 Rollback Machine** for alternatives.

Совет: *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**AUR. It supports parallelization and can handle multiple input packages in one command, which *bacman* both does not support.

Резервное копирование и извлечение списка установленных пакетов

This article or section needs expansion.

Reason: Optional dependencies that are not required by any other package (`comm -13 <(pacman -Qdtq) <(pacman -Qdttq)`) are ignored by this procedure. (Discuss in Talk:Pacman/Tips and tricks (Русский)#)

Совет: You may want to use **plist-gist**[AUR][ссылка недействительна: package not found] or **bacpac** to automatise the below tasks.

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 -Qqen > 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 badpkdlist) )
```

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.

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

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

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

Listing all changed files from packages

If you are suspecting file corruption (e.g. by software / hardware failure), but don't know for sure whether / which files really got corrupted, you might want to compare with the hash sums in the packages. This can be done with the following script.

The script depends on the accuracy of pacman's database in `/var/lib/pacman/local/` and the used programs such as *bash*, *grep* and so on. 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**.

Примечание:

- 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.
- This could take a long time, depending on the hardware and installed packages.

```
#!/bin/bash -e

# Select the hash algorithm. Currently available (see mtree files and
mtree(5)):
# md5, sha256
algo="md5"

for package in /var/lib/pacman/local/*; do
    [ "$package" = "/var/lib/pacman/local/ALPM_DB_VERSION" ] && continue

    # get files and hash sums
    zgrep " ${algo}digest=" "$package/mtree" | grep -Ev '^\./\.[A-Z]+' | \
        sed 's/^\([^ ]*\).*'"${algo}"'digest=\([a-f0-9]*\).*/\1 \2/' | \
        while read -r file hash
    do
        # expand "\nnn" (in mtree) / "\0nnn" (for echo) escapes of ASCII
        # characters (octal representation)
        for ascii in $(grep -Eo '\\[0-9]{1,3}' <<< "$file"); do
            file="$(sed "s/\\$ascii/$(echo -e "\0${ascii:1}")/" <<< "$file")"
        done

        # check file hash
        if [ "$("${algo}sum" /"$file" | awk '{ print $1; }')" != "$hash" ];
then
            echo "$(basename "$package")" /"$file"
        fi
    done
done
```

Переустановка всех пакетов

Чтобы переустановить все родные пакеты, используйте:

```
# pacman -Qenq | pacman -S -
```

Внешние пакеты (AUR) должны быть установлены отдельно; вы можете перечислить их с помощью `pacman -Qemq`.

Pacman, по умолчанию, сохраняет аргументы установки.

Restore pacman's local database

 **This article or section is out of date.** 

Reason: *testdb* has been removed in pacman 5.0 [1]. (Discuss in Talk:Pacman/Tips and tricks (Русский)#)

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**) 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** to recreate the database. If not, then the likely solution is to re-install the entire system.

Generating the package recovery list

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

Create the log filter script and make it executable:

```
pacrecover
```

```
#!/bin/bash -e

. /etc/makepkg.conf

PKGCACHE=$((grep -m 1 '^CacheDir' /etc/pacman.conf || echo 'CacheDir =
/var/cache/pacman/pkg') | sed 's/CacheDir = //')

pkgdirs=("$@" "$PKGDEST" "$PKGCACHE")

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
```

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**[broken link: invalid section].

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/lib/pacman
```

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** package:

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

Or you can use **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.

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
```

Увеличение скорости загрузки

Примечание: Если у вас низкая скорость загрузки, убедитесь, что не используете зеркало ftp.archlinux.org (**новость с марта 2007 г.**), вместо него используйте другие зеркала из списка **зеркал**.

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**.

Скорость Pacman'а при загрузке пакетов также можно улучшить, если использовать другое приложение для загрузки вместо встроенного в Pacman менеджера закачек.

В любом случае, прежде чем делать какие-либо изменения, убедитесь, что используете самый свежий Pacman.

```
# pacman -Syu
```

Powerpill

Powerpill is a full wrapper for Pacman that uses parallel and segmented downloads to speed up the download process. Normally Pacman will download one package at a time, waiting for it to complete before beginning the next download. Powerpill takes a different approach: it tries to download as many packages as possible at once.

The **Powerpill wiki page** provides basic configuration and usage examples along with package and upstream links.

wget

Если Вам нужны более мощные настройки прокси, в отличии от тех, которые предоставляет встроенный менеджер закачек **pacman**'а.

Для использования `wget`, во первых **установите** пакет **`wget`** а затем отредактируйте `/etc/pacman.conf` приведя в секции `[options]` строку к следующему виду:

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

Вместо того чтобы размещать параметры `wget` в файле `/etc/pacman.conf`, Вы можете изменять непосредственно конфигурационный файл `wget`(общесистемный файл `/etc/wgetrc`, или пользовательские файлы `$HOME/.wgetrc`.

aria2

aria2 - легковесная утилита загрузки с возможностью докачки и сегментированной HTTP/HTTPS и FTP загрузки. **aria2** - одновременно поддерживает несколько HTTP/HTTPS и FTP подключений к зеркалам Arch, это означает, что может увеличиться скорости загрузки файлов и поиска пакетов.

Примечание: 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 the **powerpill**[broken link: invalid section] section above.

Install **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
```

Совет: **This alternative configuration for using pacman with aria2** tries to simplify configuration and adds more configuration options.

See **OPTIONS** in **aria2c(1)** for used aria2c options.

`-d, --dir`
Директорию для загруженных файлов брать из настроек **pacman**.
`-o, --output`
Имя полученного файла из загруженного.
`%o`
Переменная предоставляющая локальные файлы согласно настройкам pacman.
`%u`
Переменная предоставляюща загруженные URL согласно настройкам pacman.

Другие приложения

Есть и другие приложения для загрузок, которые можно использовать с Pacman. Вот они и связанные с ними параметры XferCommand:

- `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`

Categories:

- Package management (Русский)
- Русский