ARCH LINUX · ROLLING · PACMAN · AUR

Arch Linux Handbook

For advanced users

Run the system from the inside. pacman and the AUR in depth, rolling-release maintenance, systemd, networking, ufw, Btrfs snapshots, kernels and boot loaders, NVIDIA drivers and recovery with arch-chroot.

CONTENTS

Yellow italic text (like `<package>`) marks the parts you fill in. In the terminal, paste is Ctrl+Shift+V and copy is Ctrl+Shift+C. Commands that start with `sudo` change the system; read what they do before running them.

pacman in depth

pacman's settings are in `/etc/pacman.conf`; its transaction history is in `/var/log/pacman.log`.

`$ pacman -Qo <file-path>`	Tells you which package a file belongs to.
`$ pacman -Ql <package>`	Files of an installed package.
`$ sudo pacman -Fy && pacman -F <file-name>`	Searches files in packages that aren't installed (e.g. which package provides a command).
`$ pacman -Qm`	Packages not in the official repositories (AUR and manual installs).
`$ pacman -Qkk <package>`	Checks a package's files; shows changed or missing ones.
`$ pactree <package>`	Dependency tree. Reverse (who depends on it): `pactree -r <package>` (pacman-contrib).
`$ sudo pacman -S --needed <package>`	Doesn't reinstall what's already up to date.
`$ sudo pacman -D --asexplicit <package>`	Marks a dependency as explicitly installed so orphan cleanup won't remove it.
`$ paccache -r`	Deletes all but the last 3 versions of each package from the cache. `paccache -ruk0` clears cached files of uninstalled packages.
`$ pacman -Qqe > packages.txt`	Backs up the list of explicitly installed packages. On a new system: `sudo pacman -S --needed - < packages.txt`.

TIP To keep a package from being updated, add it to the `IgnorePkg =` line in `/etc/pacman.conf`. Holding it for long creates partial-upgrade risk.

```
$ grep -iE 'installed|upgraded|removed' /var/log/pacman.log | tail -30
```

Recent package operations.

```
$ expac -H M '%m\t%n' | sort -h | tail -20
```

The 20 largest packages (expac package).

The AUR in depth

You can install packages without an AUR helper too; this is what yay does behind the scenes.

```
$ git clone https://aur.archlinux.org/<package>.git && cd <package>
```

Downloads the recipe.

```
$ less PKGBUILD
```

Read the recipe: source URLs, build and install steps.

```
$ makepkg -si
```

Installs dependencies, builds and installs the package.

```
$ git pull && makepkg -si
```

Updates a manually installed AUR package.

```
$ yay -G <package>
```

Downloads a recipe without building it.

```
$ yay -Qua
```

AUR packages with pending updates.

```
$ yay -Ps
```

System statistics: package counts, largest packages, cache size.

```
$ yay -Sc
```

Clears yay's build cache (`~/.cache/yay`).

TIP Build settings are in `/etc/makepkg.conf` . To speed up long builds, add the line `MAKEFLAGS="-j$(nproc)"` .

Living with a rolling release: maintenance

The secret to a trouble-free Arch is regular, deliberate updating: update often, read the news, merge .pacnew files.

```
$ checkupdates
```

Lists pending updates without touching the system (pacman-contrib).

```
$ yay -Pw
```

Shows unread Arch News; look before updating.

```
$ sudo pacdiff
```

Finds `.pacnew` / `.pacsave` files, shows the differences and lets you merge them.

```
$ sudo pacman -U
/var/cache/pacman/pkg/<package>-
<version>.pkg.tar.zst
```

After a bad update, downgrades a package to the older version in the cache.

> TIP If it's not in the cache, download the old version from the Arch Linux Archive (archive.archlinux.org/packages) and install it with the same command. The `downgrade` tool from the AUR does this with a menu.

```
$ sudo pacman -Sy archlinux-keyring && sudo
pacman -Su
```

On signature errors, updates the keyring first (the one exception to the rule).

```
$ sudoedit /etc/xdg/reflector/reflector.conf
```

Countries and sorting used by reflector.timer.

```
$ sudo timeshift --create --comments "before
update"
```

A snapshot before a big update. `timeshift-autosnap` from the AUR takes one automatically before every pacman transaction.

```
$ systemctl --failed && journalctl -b -p err
```

A quick health check after updating.

systemd services

systemd manages services (background daemons), timers and the boot process.

```
$ systemctl status <service>
```

Shows status, the latest log lines and the PID.

```
$ sudo systemctl start|stop|restart <service>
```

Starts / stops / restarts a service.

```
$ sudo systemctl enable --now <service>
```

Starts it at boot and starts it now.

`$ sudo systemctl disable --now <service>`	Stops it from starting at boot and stops it now.
`$ sudo systemctl mask <service>`	Prevents the service from being started at all. `unmask` undoes it.
`$ systemctl --failed`	Lists failed units.
`$ systemctl list-units --type=service --state=running`	Running services.
`$ systemctl list-unit-files --state=enabled`	Units that start at boot.
`$ systemctl list-timers`	Scheduled jobs (systemd's answer to cron).
`$ systemctl cat <service>`	Shows the unit file and any drop-ins.
`$ sudo systemctl edit <service>`	Creates an override (drop-in) without touching the packaged file.
`$ sudo systemctl daemon-reload`	Makes systemd reread unit files after you change them.
`$ systemctl --user enable --now <service>`	User services; definitions live under `~/.config/systemd/user/`.
`$ loginctl enable-linger $USER`	Lets user services run even when you are not logged in.
`$ systemd-analyze blame`	Lists how long each service took at boot.
`$ systemd-analyze critical-chain`	Shows the critical path of the boot.

Logs: journalctl

System logs are kept in the systemd journal. On Arch the journal is persistent; there is no separate syslog service.

`$ journalctl -b`	Everything since this boot.
`$ journalctl -b -1 -p err`	Errors from the previous boot. The first place to look after a crash.
`$ journalctl -u <service> -f`	Follows a service's log live.
`$ journalctl -p warning --since "1 hour ago"`	Warnings and worse from the last hour.
`$ journalctl --since today --grep "<word>"`	Searches today's log for text.
`$ journalctl -k`	Kernel messages only (like dmesg).
`$ journalctl --list-boots`	Lists recorded boots.
`$ journalctl --disk-usage`	Space used by the logs.
`$ sudo journalctl --vacuum-time=2weeks`	Deletes logs older than two weeks. `--vacuum-size=500M` also works.
`$ coredumpctl list`	Records of crashed programs (if systemd-coredump is installed). `coredumpctl info <PID>` gives details.
`$ sudo dmesg -w`	Follows the kernel ring buffer live (to see what happens when you plug in USB).

Networking: nmcli, ip, ss

On desktop installs NetworkManager runs the network; its command-line tool is `nmcli` and its text UI is `nmtui`.

`$ nmcli device status`	Network devices and their state.
`$ nmcli device wifi list`	Visible Wi-Fi networks with signal strength.

`$ nmcli device wifi connect "<SSID>" password "<password>"`	Connects to a Wi-Fi network.
`$ nmcli connection show`	Saved connections.
`$ nmcli connection up|down "<name>"`	Brings a connection up / down.
`$ sudo nmcli connection modify "<name>" ipv4.method manual ipv4.addresses 192.168.1.50/24 ipv4.gateway 192.168.1.1 ipv4.dns "1.1.1.1 9.9.9.9"`	Sets a static IP. Then `nmcli connection up "<name>"`.
`$ nmtui`	Menu-driven text UI; useful without a graphical desktop.
`$ sudo hostnamectl set-hostname <new-name>`	Changes the computer's name.
`$ ip -br a`	Interfaces and IP addresses, briefly.
`$ ip route`	Routing table and default gateway.
`$ resolvectl status`	DNS servers (when systemd-resolved is used). Otherwise `cat /etc/resolv.conf`.
`$ ss -tulpn`	Listening TCP/UDP ports and which program owns them (sudo to see the program).
`$ ping -c 4 <address>`	Reachability test.
`$ tracepath <address>`	The route packets take.
`$ curl -I https://<site>`	Fetches a site's HTTP headers.
`$ dig <domain>`	DNS lookup (bind package).

`$ iwctl`	iwd's interactive tool (for Wi-Fi on the install ISO): `device list`, `station wlan0 scan`, `station wlan0 get-networks`, `station wlan0 connect "<SSID>"`.
`$ sudo systemctl enable --now NetworkManager`	Enables NetworkManager; the desktop network menu uses it.

Firewall: ufw

Arch has no firewall by default. ufw (Uncomplicated Firewall) is a simple front end to nftables.

`$ sudo pacman -S ufw && sudo systemctl enable --now ufw`	Installs ufw and enables its service.
`$ sudo ufw status verbose`	Status and rules.
`$ sudo ufw default deny incoming && sudo ufw default allow outgoing`	Block incoming, allow outgoing (a sensible desktop default).
`$ sudo ufw enable`	Turns the firewall on and keeps it on at boot.
`$ sudo ufw allow ssh`	Allows SSH (22). `sudo ufw limit ssh` slows down brute-force attempts.
`$ sudo ufw allow 8080/tcp`	Opens a port.
`$ sudo ufw allow from 192.168.1.0/24 to any port 445`	Allows a port only from the local network.
`$ sudo ufw status numbered`	Lists rules with numbers.
`$ sudo ufw delete <number>`	Deletes a rule by number.
`$ sudo ufw app list`	Ready-made app profiles defined by packages.

```
$ sudo pacman -S gufw
```

Graphical interface.

> **WARNING** Docker writes its own iptables/nftables rules, so ports it publishes bypass ufw. If you use Docker, bind ports to localhost like `127.0.0.1:8080:80`.

Users and permissions

On this system sudo rights come from membership in the `wheel` group.

```
$ id
```

Your user ID and groups.

```
$ sudo useradd -m -G wheel -s /bin/bash <user>
```

New user with a home folder and sudo rights. Then `sudo passwd <user>`.

```
$ sudo usermod -aG <group> <user>
```

Adds a user to a group (without `-a` it removes them from other groups). Takes effect after logging in again.

```
$ sudo userdel -r <user>
```

Deletes a user along with their home folder.

```
$ passwd
```

Changes your own password.

```
$ chmod +x <file>
```

Makes a file executable.

```
$ chmod 644 <file>
```

Owner reads/writes, others read. 755 is typical for folders and scripts.

```
$ chmod -R u+rwX,go-rwx <folder>
```

Makes a folder accessible only to its owner.

```
$ sudo chown -R <user>:<group> <path>
```

Changes the owner.

```
$ setfacl -m u:<user>:rw <file>
```

Gives a specific user extra permissions (ACL). `getfacl` shows them.

```
$ sudo -i
```

Opens a root shell. Type exit when done.

```
$ sudo visudo -f /etc/sudoers.d/<name>
```

Edits sudo rules with syntax checking.

```
$ sudo EDITOR=nano visudo
```

Remove the # at the start of the `%wheel ALL=(ALL:ALL) ALL` line; otherwise the wheel group cannot use sudo.

SSH

```
$ sudo pacman -S openssh && sudo systemctl enable --now sshd
```

Installs and starts the SSH server.

```
$ ssh-keygen -t ed25519 -C "<label>"
```

Creates a key pair (`~/.ssh/id_ed25519`).

```
$ ssh-copy-id <user>@<server>
```

Uploads your public key to the server; after that you log in without a password.

```
$ ssh <user>@<server> -p <port>
```

Connects.

```
$ ssh -L 8080:localhost:80 <user>@<server>
```

Tunnels local port 8080 to port 80 on the server.

```
$ scp <file> <user>@<server>:~/
```

Copies a file to the server. `-r` for folders.

> **TIP** Define short names in `~/.ssh/config` :
> `Host myserver · HostName 192.168.1.10 · User user · IdentityFile ~/.ssh/id_ed25519`
> Then just `ssh myserver` .

> **TIP** To turn off password logins, write `PasswordAuthentication no` into `/etc/ssh/sshd_config.d/10-local.conf` and restart the SSH service.

Btrfs, snapshots and zram

When Btrfs is chosen, archinstall creates the `@` , `@home` , `@log` , `@pkg` and `@.snapshots` subvolumes. This layout works with both Timeshift and Snapper.

```
$ sudo btrfs subvolume list /
```

Lists subvolumes.

`$ sudo btrfs filesystem usage /`	Real usage according to Btrfs.
`$ sudo btrfs scrub start /`	Checks data integrity against checksums. Follow with `scrub status /`.
`$ sudo pacman -S snapper snap-pac`	Snapper, plus snap-pac to take snapshots automatically before and after every pacman transaction.
`$ sudo snapper -c root list`	Snapper snapshots.
`$ sudo snapper -c root undochange <before>..<after>`	Reverts the changes between two snapshots.
`$ sudo pacman -S grub-btrfs && sudo systemctl enable --now grub-btrfsd`	Adds snapshots to the GRUB menu, so you can boot a broken system from an older snapshot. To use it with Timeshift, edit the service to use the `--timeshift-auto` option.
`$ sudo pacman -S zram-generator`	Compressed swap in RAM. Write `[zram0]` and `zram-size = ram / 2` into `/etc/systemd/zram-generator.conf` and reboot.
`$ zramctl && swapon --show`	zram and swap status.

WARNING Snapper's `create-config` wants to create its own `.snapshots` subvolume and conflicts with archinstall's `@.snapshots`. Follow the order on the Arch Wiki's Snapper page.

Kernel, initramfs and boot loader

Arch can have several kernels installed side by side. Installing `linux-lts` as a fallback lets you boot it from the menu if a new kernel misbehaves.

`$ sudo pacman -S linux-lts linux-lts-headers`	Installs the long-term support kernel as a fallback. Others: `linux-zen`, `linux-hardened`.

```
$ uname -r && pacman -Q linux
```

Running and installed kernel versions; if they differ, you need to restart.

```
$ sudoedit /etc/mkinitcpio.conf
```

initramfs settings (HOOKS, MODULES).

```
$ sudo mkinitcpio -P
```

Rebuilds the initramfs for all kernels.

```
$ bootctl status
```

Status and entries when systemd-boot is used.

TIP With systemd-boot, kernel parameters are on the `options` line of the files in `/boot/loader/entries/*.conf` ; the menu timeout is `timeout` in `/boot/loader/loader.conf` .

```
$ sudoedit /etc/default/grub
```

Settings when GRUB is used; parameters go on the `GRUB_CMDLINE_LINUX_DEFAULT` line.

```
$ sudo grub-mkconfig -o /boot/grub/grub.cfg
```

Regenerates the GRUB configuration (after a new kernel or microcode).

```
$ cat /proc/cmdline
```

Parameters of the running kernel.

```
$ lsmod | grep <module>
```

Loaded modules. `modinfo` for details, `sudo modprobe` to load.

Drivers: NVIDIA, AMD, Intel

Graphics drivers are in the official repositories. multilib must be enabled for 32-bit game support.

```
$ sudo pacman -S nvidia-open nvidia-utils
lib32-nvidia-utils
```

Driver for NVIDIA Turing (GTX 16xx / RTX 20xx) and later, with the `linux` kernel. Use `nvidia-open-lts` for LTS and `nvidia-open-dkms` for other kernels.

TIP After installing NVIDIA, remove `kms` from the HOOKS line in `/etc/mkinitcpio.conf` (keeps nouveau from loading) and run `sudo mkinitcpio -P` . For older cards, see the legacy driver branches in the AUR via the Arch Wiki's NVIDIA page.

`$ nvidia-smi`	GPU status and usage.
`$ sudo pacman -S mesa vulkan-radeon lib32-mesa lib32-vulkan-radeon`	AMD: OpenGL, Vulkan and hardware video decoding.
`$ sudo pacman -S mesa vulkan-intel lib32-vulkan-intel intel-media-driver`	Intel: OpenGL, Vulkan and hardware video decoding.
`$ sudo pacman -S libva-utils && vainfo`	Verifies VA-API hardware acceleration.
`$ lspci -k | grep -A 3 -E "VGA|3D"`	The graphics card and the kernel driver in use.
`$ sudo pacman -S steam`	Steam (needs multilib; it asks for a Vulkan driver during install, pick the one for your card).

Podman, Docker, Distrobox

Distrobox lets you use other distributions' tools without cluttering the host; Podman or Docker run services. Podman is largely compatible with Docker commands and doesn't need root.

`$ sudo pacman -S podman distrobox`	Installs Podman and Distrobox.
`$ distrobox create -i registry.fedoraproject.org/fedora:latest -n <name>`	Creates a container of another distribution (e.g. `-i archlinux`, `-i ubuntu:24.04`).
`$ distrobox enter <name>`	Enters it; your home folder is shared. Inside, `distrobox-export --app <app>` adds the app to the host menu.
`$ podman run -it --rm debian bash`	Starts a throwaway container.
`$ podman ps -a`	All containers.
`$ podman images`	Local images. `podman system prune -a` removes unused ones.

```
$ podman run -d -p 8080:80 -v
~/site:/usr/share/nginx/html
docker.io/library/nginx
```

Runs a service with a mounted folder.

```
$ sudo pacman -S docker docker-compose &&
sudo systemctl enable --now docker
```

Installs Docker and starts its service.

```
$ sudo usermod -aG docker $USER
```

Use Docker without sudo (log in again). This group is equivalent to root access.

> TIP To run a Podman container as a systemd service, use Quadlet: write a `.container` file under `~/.config/containers/systemd/`, run `systemctl --user daemon-reload` and start it.

Text processing and pipelines

`|` passes one command's output to the next. `>` writes to a file (overwriting), `>>` appends, `2>&1` sends errors to the same place. `&&` runs if the previous command succeeded, `||` if it failed.

```
$ grep -rni "<text>" <folder>
```

Recursive, case-insensitive search with line numbers.

```
$ grep -v "^#" <file> | grep -v "^$"
```

Drops comments and blank lines; handy for reading config files.

```
$ sed -i 's/<old>/<new>/g' <file>
```

Find and replace in a file. Try without `-i` first and check the output.

```
$ awk '{print $1, $3}' <file>
```

Prints the 1st and 3rd column of each line.

```
$ sort <file> | uniq -c | sort -rn | head
```

Most frequently repeated lines.

```
$ wc -l <file>
```

Line count.

```
$ cut -d: -f1 /etc/passwd
```

First colon-separated field (user names).

```
$ find . -name "*.log" -mtime +7 -delete
```

Deletes .log files older than 7 days.

```
$ find . -type f -name "*.jpg" -print0 |
xargs -0 -I{} cp {} ~/Pictures/
```

Copies found files one by one (safe with spaces in names).

```
$ <command> | tee <file>
```

Writes output to both the screen and a file.

```
$ echo "<line>" | sudo tee -a /etc/<file>
```

Appends to a root-owned file (`sudo echo >>` doesn't work).

```
$ diff -u <old> <new>
```

Differences between two files.

```
$ watch -n 2 <command>
```

Reruns a command every 2 seconds.

```
$ curl -s <URL> | jq '.'
```

Pretty-prints JSON (jq package).

Archives, backups, syncing

```
$ tar -czvf <archive>.tar.gz <folder>
```

Archives a folder with gzip.

```
$ tar --zstd -cvf <archive>.tar.zst <folder>
```

Archives with zstd; faster than gzip and compresses better.

```
$ tar -xvf <archive>
```

Extracts any tar archive. Use `-C <folder>` for a target folder.

```
$ tar -tvf <archive>
```

Lists the contents without extracting.

```
$ zip -r <archive>.zip <folder>
```

Creates a zip. Extract with `unzip <archive>.zip`.

```
$ 7z x <archive>.7z
```

Extracts 7z / rar and similar formats (7zip package).

```
$ rsync -avh --progress <source>/ <target>/
```

Syncs a folder; copies only what changed. The trailing / means “the contents”.

`$ rsync -avh --delete --dry-run <source>/ <target>/`	Mirror, deleting from the target what's not in the source; see what would happen first with --dry-run.
`$ rsync -avz -e ssh <source>/ <user>@<server>:<path>/`	Remote backup over SSH.
`$ rclone config`	Sets up cloud targets such as Google Drive, OneDrive and S3; sync with `rclone sync`.

Customizing the shell

Bash settings live in `~/.bashrc` in your home folder. Add aliases and variables at the end of the file.

`$ nano ~/.bashrc`	Opens the shell configuration file.
`alias update='sudo pacman -Syu'`	System update in one word. Write it into the file and save.
`alias ll='ls -lah --group-directories-first'`	A handy detailed listing.
`$ source ~/.bashrc`	Loads the changes without opening a new terminal.
`export PATH="$HOME/.local/bin:$PATH"`	Adds your own scripts folder to PATH.
`$ echo $SHELL`	The shell you are using.
`$ sudo pacman -S zsh && chsh -s /usr/bin/zsh`	Installs Zsh and makes it the default (log in again). Same method for Fish.
`$ type <command>`	Shows whether a name is an alias, a function or a file.

> **TIP** When writing a script, put `#!/usr/bin/env bash` on the first line and `set -euo pipefail` on the second; it stops on errors and catches undefined variables. Then `chmod +x script.sh`.

Hidden GNOME settings with gsettings

All GNOME settings live in the dconf database. gsettings reaches options that aren't in the Settings app.

`$ gsettings set org.gnome.desktop.interface color-scheme 'prefer-dark'`	Dark style ('default' for light).
`$ gsettings set org.gnome.desktop.interface clock-show-seconds true`	Shows seconds on the clock.
`$ gsettings set org.gnome.desktop.interface clock-show-weekday true`	Shows the weekday next to the clock.
`$ gsettings set org.gnome.mutter center-new-windows true`	Opens new windows in the center of the screen.
`$ gsettings set org.gnome.desktop.peripherals.touchpad tap-to-click true`	Tap to click.
`$ gsettings set org.gnome.shell disable-user-extensions true`	Turns off all extensions at once (troubleshooting). false turns them back on.
`$ gsettings list-recursively org.gnome.desktop.interface`	All keys and values in a schema.
`$ gsettings reset <schema> <key>`	Resets a setting to its default.
`$ dconf dump / > ~/gnome-settings.ini`	Backs up all GNOME settings to a file.
`$ dconf load / < ~/gnome-settings.ini`	Restores the backup; useful for carrying settings to a new install.
`$ sudo pacman -S dconf-editor`	A graphical tool to browse every key with descriptions.

Performance and power

For power profiles install `sudo pacman -S power-profiles-daemon` and enable it with `sudo systemctl enable --now power-profiles-daemon`; GNOME and KDE use it.

`$ powerprofilesctl list`	Power profiles and which one is active (power-profiles-daemon).
`$ powerprofilesctl set performance`	Switches profile: power-saver / balanced / performance.
`$ sudo pacman -S lm_sensors`	CPU and disk temperatures; run `sensors` after installing.
`$ sudo pacman -S btop nvtop`	Advanced system and GPU monitors.
`$ sudo iotop -o`	Processes writing the most to disk (iotop package).
`$ systemd-analyze`	Total boot time (firmware, loader, kernel, userspace).
`$ sudo fstrim -av`	Runs TRIM on SSDs. To do it weekly: `sudo systemctl enable --now fstrim.timer`.
`$ upower -i $(upower -e | grep BAT)`	Battery health, capacity and charge information.
`$ free -h && swapon --show`	RAM and swap areas.

Troubleshooting and recovery

What to try, in order, when the system won't boot or an update broke something. On Arch the most powerful tools are the install ISO and `arch-chroot`.

> TIP **1. Boot the fallback kernel or a snapshot.** If linux-lts is installed, pick it from the boot menu; with grub-btrfs you can boot an older snapshot.

TIP **2. Switch to a text console.** Ctrl+Alt+F3 (F2–F6). Log in and check errors with `journalctl -b -p err` and `systemctl --failed`.

```
$ sudo systemctl restart display-manager
```

Restarts the graphical login manager (GDM, SDDM); the open session closes.

TIP **3. arch-chroot from the ISO.** Boot the Arch install USB and run the following in order (you're already root on the ISO). Adjust disk names to your system using `lsblk`.

```
# mount -o subvol=@ /dev/<nvme0n1p2> /mnt
```

Mounts the Btrfs root subvolume (for ext4, drop `-o subvol=@`).

```
# mount /dev/<nvme0n1p1> /mnt/boot
```

Mounts the EFI partition (archinstall mounts it at /boot).

```
# arch-chroot /mnt
```

Switches into the installed system; it binds /dev, /proc and /sys by itself.

```
# pacman -Syu && mkinitcpio -P
```

Inside the chroot, finishes an interrupted update and rebuilds the initramfs.

```
# grub-mkconfig -o /boot/grub/grub.cfg
```

If you use GRUB, regenerates its configuration. For systemd-boot: `bootctl install`.

```
# pacman -Qkk 2>/dev/null | grep -v ' 0
altered'
```

Finds packages with damaged files; reinstall them with `pacman -S <package>`.

Commands are written for current Arch Linux (pacman 7, a GNOME or KDE Plasma desktop installed with archinstall). Package names can change over time on a rolling release; when in doubt, the Arch Wiki wins. Commands that start with `sudo` change the system; read what they do before running them.