Disk Partitioning & Arch Linux Installation

Practical exercises — from loop devices to a fully working Arch Linux system

`fallocate` `losetup` `fdisk` `mkfs` `mount` `pacstrap` `grub`

1	2	3	4	5
Partitioning	**mounting**	**Pacstraping**	**editing files**	**Bootloader**
`fdisk`	`mount`	`pacstrap`	`nano/passwd`	`grub/refind`

5 simple steps

1	2	3	4	5
Partitioning	**mounting**	**Pacstraping**	**editing files**	**Bootloader**
`fdisk`	`mount`	`pacstrap`	`nano/passwd`	`grub/refind`

1 — Partitioning

Boot Arch ISO & connect to Wi-Fi

```
# Not needed in VM — skip to partitioning
$ ifconfig -a # find wifi interface
$ wpa_passphrase wifi_name pass > wifi_cfg
$ wpa_supplicant -i wlan0 -c ./wifi_cfg -B
$ dhclient wlan0
$ ping wp.pl
```

Create and format partitions

```
$ fdisk -l
$ fdisk /dev/vda # GPT: boot, swap, /, /home
$ mkfs.vfat /dev/vda1 # /boot — FAT32!
$ mkfs.ext4 /dev/vda2 # root — ext4/btrfs/xfs
```

Typical partition layout

1	/boot — 512 MB
2	/ (root) — 20–40 GB
3	/home — remaining

⚠ /boot MUST be FAT32 for UEFI. Never format as ext4.

↔ In a VM: skip Wi-Fi — the virtual NIC gets network automatically via DHCP.

2 — Mounting

```
# Mount root (/) partition FIRST
$ mount /dev/vda2 /mnt

# Create mount points inside the mounted root
$ mkdir /mnt/boot /mnt/home ...

# Mount /boot partition
$ mount /dev/vda1 /mnt/boot

# Mount /home partition
$ mount /dev/vda3 /mnt/home

# Turn on swap (if you have a swap partition)
$ swapon /dev/vda4
```

⚠ Always mount root first! Subdirectories must exist inside it before mounting children into them.

↞ All Arch install tools (pacstrap, genfstab, arch-chroot) assume the target is at /mnt. Verify with findmnt /mnt.

3 — Pacstrap

```
# -K = initialise new pacman keyring in /mnt
$ pacstrap -K /mnt base linux linux-firmware grub refind nano wpa_supplicant net-tools
```

Package	Description
`base`	Minimal system essentials — shell, coreutils, pacman
`linux`	The Linux kernel
`linux-firmware`	Hardware firmware blobs (WiFi, GPU, etc.)
`grub`	Legacy + UEFI bootloader manager
`refind`	rEFInd — modern UEFI boot manager (alternative to GRUB)
`nano`	Simple text editor
`wpa_supplicant`	WiFi authentication client (WPA2/3)
`net-tools`	Legacy network utilities (ifconfig, route...)

4 — Files editing (before arch-chroot)

```
$ cp /etc/resolv.conf /mnt/etc/ # DNS

$ nano /mnt/etc/vconsole.conf
KEYMAP=pl2
FONT=Lat2-Terminus16
XKBLAYOUT=pl
FONT_MAP=8859-2

$ nano /mnt/etc/locale.conf
LANG=pl_PL.UTF-8

$ nano /mnt/etc/locale.gen
pl_PL.UTF-8 UTF-8
en_US.UTF-8 UTF-8

$ nano /mnt/etc/hostname
```

```
$ nano /mnt/etc/pacman.conf
# uncomment multilib (remove #):
[multilib]
Include = /etc/pacman.d/mirrorlist

# Enter the new system
$ arch-chroot /mnt
[root@archiso /]# ← inside!
```

↔ arch-chroot auto-mounts /proc /sys /dev /run — smarter than plain chroot.

⚠ Multilib enables 32-bit packages — needed for Steam, Wine, Proton. Uncomment both lines together.

5 — Install bootloader

(inside arch-chroot /mnt)

```
# Set root password
# passwd

# Create regular user
# useradd -m -G users,wheel,audio,video -s /bin/bash user_name

# Install GRUB (UEFI + GPT)
# grub-install --target=x86_64-efi --efi-directory=/boot --bootloader-id=GRUB
# grub-mkconfig -o /boot/grub/grub.cfg
```

```
# OR install rEFInd (auto-detects kernels)
# refind-install

# GRUB for BIOS/MBR (legacy systems)
# grub-install /dev/sda # whole disk!
# grub-mkconfig -o /boot/grub/grub.cfg

# reboot
```

↔ wheel group = sudo access. rEFInd needs no manual config — auto-detects kernels.

Arch Linux is Ready!

1. Log in at the console
2. Get network: dhclient enp0s3
3. Start SSH: systemctl start sshd
4. Connect via PuTTY (as user, not root)
5. Use su / sudo for admin tasks

5 simple steps — recap

1	2	3	4	5
Partitioning	**mounting**	**Pacstraping**	**editing files**	**Bootloader**
`fdisk`	`mount`	`pacstrap`	`nano/passwd`	`grub/refind`

↭ The same 5 steps apply to all scenarios — real hardware, VM, and loop device. Only the device path changes.

1 — Partitioning (loop device variant)

```
# Same Wi-Fi setup (not needed in VM)
$ ifconfig -a
$ wpa_passphrase wifi_name pass > wifi_cfg && wpa_supplicant -i wlan0 -c ./wifi_cfg -B
$ dhclient wlan0

# Using a loop device instead of a real disk
$ fdisk -l
$ fdisk /dev/loop0 # I'll use loop device
$ mkfs.vfat /dev/loop0p1 # format /boot — FAT32
$ mkfs.ext4 /dev/loop0p2 # format root
```

↞ Loop device: losetup -P --find --show disk.img creates /dev/loop0 + /dev/loop0p1, /dev/loop0p2… From fdisk's view it's identical to /dev/sda.

1 — Partitioning (/dev/sda variant)

```
# Create partitions — /dev/sda in this example
$ fdisk -l # list disks first
$ fdisk /dev/sda # create FAT32 boot partition first
$ mkfs.vfat /dev/sda1 # format /boot partition
```

↭ Device names vary: /dev/sda (SATA/USB), /dev/vda (VM), /dev/nvme0n1 (NVMe SSD), /dev/loop0 (loop device). Always run fdisk -l first to identify your disk.

1 — Partitioning (/dev/sdb + benchmark)

```
# Check encryption performance before choosing cipher
$ cryptsetup benchmark

# Partition /dev/sdb (in this example)
$ fdisk /dev/sdb # create FAT32 boot partition first
$ mkfs.vfat /dev/sdb1 # format /boot partition
```

↞ Run cryptsetup benchmark before installing if you plan to use LUKS encryption — helps choose the fastest cipher for your specific CPU.

Create encrypted partition — choose cipher

```
$ fdisk /dev/vda # partition 1 = boot (EFI), 2 = root (/)
$ cryptsetup benchmark # find the best algorithm

# Tests are approximate using memory only (no storage IO).
PBKDF2-sha256 4215380 iterations per second for 256-bit key
argon2id 8 iterations, 1048576 memory, 4 parallel threads for 256-bit key

# Algorithm | Key | Encryption | Decryption
aes-cbc 128b 1555.4 MiB/s 5063.2 MiB/s
aes-cbc 256b 1240.3 MiB/s 4315.7 MiB/s
aes-xts 256b 5976.2 MiB/s 6067.7 MiB/s ← best!
serpent-xts 256b 611.2 MiB/s 567.0 MiB/s
twofish-xts 256b 375.4 MiB/s 377.3 MiB/s
aes-xts 512b 5437.1 MiB/s 5552.3 MiB/s
```

↞ On CPUs with AES-NI hardware acceleration, aes-xts-plain64 with 512-bit key is fastest. Run benchmark on the target machine to confirm.

Create encrypted partition, key-file protected

```
# 1 – Create safe key-file on USB
$ mkdir /usb && mount /dev/sda1 /usb
$ head -c 1024 /dev/urandom > /usb/not_a_key

# 2 – Encrypt the partition
$ cryptsetup luksFormat --type luks2 --cipher aes-xts-plain64 \
--key-size 512 --hash sha512 --key-file /usb/not_a_key /dev/vda2

# 3 – Open the encrypted drive
$ cryptsetup luksOpen --key-file /usb/not_a_key /dev/vda2 arch_linux
$ fdisk -l # find /dev/mapper/arch_linux

# 4 – Format & mount
$ mkfs.ext4 /dev/mapper/arch_linux
$ mount /dev/mapper/arch_linux /mnt
$ mkdir /mnt/boot && mkfs.vfat /dev/vda1 && mount /dev/vda1 /mnt/boot
$ pacstrap -K /mnt base linux linux-firmware nano ...
```

Create encrypted whole drive, key-file protected

```
# Create safe key-file on USB (same as before)
$ mkdir /usb && mount /dev/sda1 /usb
$ head -c 1024 /dev/urandom > /usb/not_a_key

# Encrypt the WHOLE drive (/dev/vda not /dev/vda2)
$ cryptsetup luksFormat --type luks2 --cipher aes-xts-plain64 \
--key-size 512 --hash sha512 --key-file /usb/not_a_key /dev/vda
# NOTE: no boot partition on the drive – whole drive is encrypted!!!

# Open, format & mount
$ cryptsetup luksOpen --key-file /usb/not_a_key /dev/vda enigma
$ mkfs.ext4 /dev/mapper/enigma
$ mount /dev/mapper/enigma /mnt
$ mkdir /mnt/boot && mount /dev/sda1 /mnt/boot # /boot is on the USB!
$ pacstrap -K /mnt base linux linux-firmware nano ...
```

Encrypted whole drive — no header (step 1)

The drive has a crypto_LUKS signature — detectable. We need to erase it.

```
$ fdisk /dev/vda

Welcome to fdisk (util-linux 2.40.4).
Changes will remain in memory only, until you decide to write them.

The device contains 'crypto_LUKS' signature and it will be removed
by a write command. See fdisk(8) man page and --wipe option.

Device does not contain a recognized partition table.
Created a new DOS (MBR) disklabel with disk identifier 0x02411ef0.
Command (m for help):
```

↞ The LUKS signature at the start of the drive proves it is encrypted. To achieve plausible deniability, we must destroy it — next slide shows how.

Encrypted whole drive — no header (step 2)

Overwrite the first 2 MB with random data to destroy the LUKS signature

```
# Clear the first two megabytes (destroys partition table AND LUKS signature)
$ head -c 2M /dev/urandom > /dev/vda

# Now fdisk shows an empty drive — no signature at all
$ fdisk /dev/vda

Welcome to fdisk (util-linux 2.40.4).
Changes will remain in memory only, until you decide to write them.

Device does not contain a recognized partition table. ← clean!
Created a new DOS (MBR) disklabel with disk identifier 0x79982863.
Command (m for help):

# fdisk -l now shows empty drive — no LUKS evidence
```

Encrypted whole drive — no header (step 3)

Store the LUKS header externally on the USB — drive looks like random noise

```
# Encrypt with detached header stored on USB
$ cryptsetup luksFormat --type luks2 --cipher aes-xts-plain64 \
--key-size 512 --hash sha512 \
--key-file /usb/not_a_key --header /usb/hidden_header /dev/vda
# NOTE: no boot partition on the drive – whole drive encrypted!!!

# Open (need BOTH key + header from USB)
$ cryptsetup luksOpen \
--key-file /usb/not_a_key --header /usb/hidden_header \
/dev/vda enigma

# Format & mount as usual
$ mkfs.ext4 /dev/mapper/enigma
$ mount /dev/mapper/enigma /mnt && mkdir /mnt/boot
$ mount /dev/sda1 /mnt/boot # USB's FAT32 as /boot
```

↞ Without the USB (key + header), the drive looks like random data with no detectable encryption marker. Maximum deniability.

pacstrap → genfstab → arch-chroot

```
# Install the base Arch Linux software
$ pacstrap -K /mnt base linux linux-firmware dhclient wpa_supplicant \
grub nano mc dosfstools net-tools lsof cryptsetup \
openssh sof-firmware linux-firmware-marvell efibootmgr

# Generate the fstab file (use -U for UUIDs)
$ genfstab -U /mnt >> /mnt/etc/fstab
$ cat /mnt/etc/fstab # always verify!

# Change root into the new environment
$ arch-chroot /mnt
[root@archiso /]# ← you are now INSIDE the new system
```

⚠ Use >> (append) not > (overwrite) for genfstab — otherwise you'll wipe fstab each time.

Set the time zone

(inside arch-chroot /mnt)

```
# Generic form
# ln -sf /usr/share/zoneinfo/Region/City /etc/localtime

# For Poland
# ln -sf /usr/share/zoneinfo/Europe/Warsaw /etc/localtime

# List available timezones
# ls /usr/share/zoneinfo/
# ls /usr/share/zoneinfo/Europe/
```

↞ ln -sf creates a symlink — /etc/localtime points to the timezone file. The -f flag overwrites any existing symlink.

Hardware clock, locale & console

(inside arch-chroot /mnt)

```
# Sync hardware clock
# hwclock --systohc

# Set locales
# echo "en_US.UTF-8 UTF-8" > /etc/locale.gen
# echo "en_US ISO-8859-1" >> /etc/locale.gen
# echo "pl_PL.UTF-8 UTF-8" >> /etc/locale.gen
# echo "pl_PL ISO-8859-2" >> /etc/locale.gen
# locale-gen
```

```
# Console keyboard and font
# echo KEYMAP=pl >> /etc/vconsole.conf
# echo FONT=Lat2-Terminus16 >> /etc/vconsole.conf
# echo FONT_MAP=8859-2 >> /etc/vconsole.conf
```

Hostname & mkinitcpio for encrypted boot

(inside arch-chroot /mnt)

```
# Set hostname
# echo "arch" >> /etc/hostname

# /etc/mkinitcpio.conf
MODULES=(vfat)
BINARIES=()
FILES=()
HOOKS=(base udev autodetect microcode modconf kms
keyboard keymap consolefont block filesystems
fsck encrypt)
```

```
# /etc/initcpio/hooks/unlock_usb
run_hook() {
mkdir -p /usb
mount -t vfat /dev/sda1 /usb
cryptsetup luksOpen \
--key-file /usb/not_a_key \
--header /usb/hidden_header \
/dev/vda enigma
}

# /etc/initcpio/install/unlock_usb
build() { add_runscript }

# Regenerate kernel + ramdisk
# mkinitcpio -P
```

Root password, updates & user creation

(inside arch-chroot /mnt)

```
# Set root password
# passwd

# Update package database
# pacman -Syu

# Install CPU microcode (pick the right one!)
# pacman -S intel-ucode # Intel CPUs
# pacman -S amd-ucode # AMD CPUs

# Create a regular system user
# useradd -m -G users,wheel,audio,video,ftp,disk -s /bin/bash username
# useradd -m -G users,wheel,audio,video,ftp,disk -s /bin/bash harnas # example
```

↔ CPU microcode fixes hardware bugs (Spectre/Meltdown etc). Always install it — GRUB picks it up automatically with grub-mkconfig.

GRUB config for encrypted drives

nano /boot/grub/grub.cfg — menuentry examples

```
# Encrypted partition (key on USB)
menuentry 'Arch Linux - Encrypted partition' {
linux /vmlinuz-linux root=/dev/mapper/arch_linux rw loglevel=3 quiet \
cryptdevice=/dev/nvme0n1p6:arch_linux cryptkey=UUID=ABCE-914E:vfat:/key_file
initrd /intel-ucode.img /initramfs-linux.img
}

# Whole-disk encryption with detached header
menuentry 'Arch Linux - whole disk enc' {
linux /vmlinuz-linux root=UUID=0d416310-b881-489d-8a45-ee75a638f9bf rw quiet \
cryptdevice=/dev/nvme0n1:hell:allow-discards \
cryptheader=UUID=ABCE-914E:vfat:/luks_header cryptkey=UUID=ABCE-914E:vfat:/new_key
initrd /intel-ucode.img /initramfs-linux.img
}
```

GRUB install & get partition UUIDs

```
# pacman -S grub

# UEFI + GPT:
# grub-install --target=x86_64-efi --efi-directory=/boot/ --bootloader-id=GRUB
# MBR + BIOS:
# grub-install /dev/sda # whole disk, not partition!

# Get UUIDs
# ls /dev/disk/by-uuid/ -al
lrwxrwxrwx A009-D2C3 -> ../../sda1 ← /boot
lrwxrwxrwx e3e1ae5e-b1c8-... -> ../../sda2 ← /

# Append UUID list to grub.cfg as reference (use >> not >!)
# ls /dev/disk/by-uuid/ -al >> /boot/grub/grub.cfg
# or:
# blkid >> /boot/grub/grub.cfg
```

⚠ Use >> (append) not > (overwrite) when adding UUID info to grub.cfg!

/boot folder contents

Verify all required files are present before rebooting

```
# ls /boot -al

drwxr-xr-x EFI/ ← UEFI bootloader files
-rwxr-xr-x vmlinuz-linux ← the Arch kernel
-rwxr-xr-x initramfs-linux.img ← the ramdisk (initrd)
-rwxr-xr-x initramfs-linux-fallback.img ← fallback ramdisk
-rwxr-xr-x intel-ucode.img ← Intel CPU microcode
drwxr-xr-x grub/ ← GRUB files & grub.cfg

# Also visible: kernels from other distros — safe to ignore:
-rwxr-xr-x vmlinuz-6.7.10-gentoo-dist
-rwxr-xr-x initramfs-6.7.10-gentoo-dist.img

# You can remove unused kernels and ramdisks to save space
```

grub.cfg — full configuration example

```
# nano /boot/grub/grub.cfg
set default="0" set timeout_style=menu set timeout=15
set gfxmode=auto load_video insmod gfxterm terminal_output gfxterm
insmod efi_uga insmod vbe insmod vga insmod part_gpt
insmod ext2 insmod vfat insmod fat

# Set GRUB root to the /boot partition (FAT32 UUID)
search --no-floppy --fs-uuid --set=root A009-D2C3 ← UUID of /boot

menuentry 'Arch - by UUID' {
linux /vmlinuz-linux root=UUID=e3e1ae5e-b1c8-47b8-8daa-d4edaace0a65 ro
initrd /initramfs-linux.img /intel-ucode.img
}

menuentry 'Arch - by name' {
linux /vmlinuz-linux root=/dev/sda2
initrd /initramfs-linux.img /intel-ucode.img
}
```

↞ GRUB 'root' = the /boot partition. Kernel 'root=' = the Linux / partition. They can be different devices.

GRUB visual customization

```
set menu_color_normal=blue/yellow
set menu_color_highlight=light-green/brown
set color_normal=white/red
set color_highlight=blue/magenta
set background_image /path/to/your/image.jpg
```

↞ Colors are specified as foreground/background. Example: light-green/brown = light-green text on brown background.

First boot — network & SSH

```
$ ifconfig -a # see network adapters list
$ dhclient enp0s3 # get IP from DHCP server
$ systemctl start sshd # start SSH server
$ systemctl enable sshd # start SSH at every boot
```

↞ Now you can login via PuTTY — as user, not root! Then use su to switch to root, or configure sudo by uncommenting %wheel ALL=(ALL:ALL) ALL in /etc/sudoers.

The full graphical stack — from firmware to desktop

1

Boot managers

GRUB, rEFInd, LiLo — select OS/kernel at startup, before Linux loads

2

Display servers

X11 / Xorg, Wayland — bridge between kernel and graphical applications

3

Login managers

GDM, SDDM, LightDM — graphical login screen, starts after boot

4

Window managers

i3, KWin, Openbox — controls window placement and appearance

5

Desktop environments

KDE Plasma, GNOME, XFCE — full suite: WM + panel + apps + settings

Boot Managers

Responsible for selecting and launching the OS during boot. Operate before the Linux kernel is loaded.

↭ rEFInd is the easiest: refind-install, done. GRUB is more powerful and supports BIOS systems too.

Display Servers

Handle communication between the OS and the graphical interface. Provide the foundation for WMs and DEs to render graphics.

Key features

Default GNOME, KDE, Xfce, MATE, LXDE, Cinnamon, ...

Provide a graphical or text-based interface to log in. Start after boot, handle authentication and session selection.

Name	Description
GDM	GNOME Display Manager. Modern, somewhat heavy.
SDDM	Simple and beautiful. Supports Wayland and X11. QML themeable.
LightDM	Lightweight, highly customizable, supports many themes.
LXDM	Very light and fast, but less configurable.
XDM	Classic, minimal, very basic UI.
Ly	Terminal-based display manager (TTY).
SDDM-Greeter	Customizable frontend for SDDM.
CDM	Very light; runs entirely in the terminal
nodm	Auto-login without showing a login screen
tbsm	Terminal-based session manager written in Bash

Desktop Environments & Window Managers

Full graphical suites — WM + panel + apps bundled together

KDE Plasma

Full DE (Qt)

Feature-rich, customisable. Excellent Wayland. Plasma 6 is fast. DM: SDDM

GNOME

Full DE (GTK)

Clean, opinionated. Activity-based. Best Wayland support. DM: GDM

XFCE

Lightweight

Fast, low RAM. Traditional layout. Great for VMs & older hardware. DM: LightDM

LXQt

Lightweight Qt

Very low RAM usage. Qt-based LXDE successor. Good for weak hardware.

Cinnamon

Traditional DE

Windows-like layout. GNOME 3 fork. User-friendly for Windows switchers.

i3

Tiling WM (X11)

Minimal tiling WM — build the stack yourself. Plain text file config.

Hyprland

Tiling (Wayland)

Modern Wayland compositor with animations. Growing fast. Eye-catching.

Openbox

Floating WM

Lightweight floating WM. Used in LXDE and minimal setups.

Controls window placement and appearance — can be used standalone without a full DE

Tiling WMs (auto-arranged, no overlap)

`i3`	Very popular, simple, fast. Configured via a plain text file.
`sway`	Modern Wayland-compatible i3 replacement.
`bspwm`	Binary space partitioning; controlled with sxhkd.
`awesome`	Highly configurable and scriptable in Lua.
`xmonad`	Written in Haskell, stable and fast.
`herbstluftwm`	Manual tiling, tree-style layout.

Floating & Dynamic WMs

`Openbox`	Lightweight, highly configurable; used in LXDE.
`Fluxbox`	Fast; fork of Blackbox with taskbar.
`IceWM`	Extremely light; looks like Windows 95/98.
`JWM`	Even lighter; used in Puppy Linux.
`FVWM`	Old-school, powerful, popular in the 90s.
`Qtile`	Fully written in Python; flexible and dynamic.
`dwm`	Ultra-minimal; edit C source code to configure.

SPELL FOR PLASMA USERS

```
# pacman -S kde-accessibility-meta kde-applications-meta kde-graphics-meta \
kde-multimedia-meta kde-network-meta kde-office-meta kde-system-meta \
kde-utilities-meta kdevelop-meta plasma-meta plasma-nm \
git k3b ntfs-3g fuse firefox thunderbird vlc spotify-launcher \
nextcloud-client signal-desktop atop discord partitionmanager gparted
# systemctl enable NetworkManager
# systemctl enable sddm
# systemctl enable bluetooth
# Remember to use the sync command before rebooting!
```

↔ To transfer files between Windows and your VM, use WinSCP — connects over SFTP/SSH. Download: winscp.net

Plasma minimal (core only)

```
# pacman -S plasma-meta kde-system-meta kde-utilities-meta kde-network-meta kde-graphics-meta kde-multimedia-meta kde-office-meta
```

Plasma "Witcher" — developer / power-user build

```
# pacman -S \
plasma-meta kde-system-meta kde-utilities-meta kde-network-meta \
firefox thunderbird vlc spotify-launcher nextcloud-client signal-desktop discord \
git base-devel sudo bash-completion man-db man-pages \
htop btop neovim wget curl rsync unzip zip \
pipewire pipewire-alsa pipewire-pulse wireplumber \
networkmanager-openvpn \
dosfstools exfatprogs mtools ntfs-3g fuse \
qemu-full virt-manager dnsmasq bridge-utils edk2-ovmf \
mesa vulkan-radeon vulkan-intel strace ltrace lsof tcpdump nmap \
dolphin konsole ark okular gwenview spectacle \
partitionmanager gparted grub efibootmgr \
tmux zsh fzf ripgrep fd bpytop iftop atop
```

5 — Enable services

(inside arch-chroot /mnt — before first reboot)

```
# Enable NetworkManager — handles wired + wireless connections
# systemctl enable NetworkManager

# Enable SSH server — start at every boot
# systemctl enable sshd

# Enable SDDM display manager (for KDE / any desktop)
# systemctl enable sddm

# Enable Bluetooth
# systemctl enable bluetooth
```

↔ systemctl enable creates a symlink so the service starts at boot. It does NOT start it immediately — use systemctl start or enable --now for that.

systemctl — Managing Services

systemd's CLI for controlling services, targets, and the init system

Start & Stop

`systemctl start <svc>`
Start immediately

`systemctl stop <svc>`
Stop immediately

`systemctl restart <svc>`
Stop then start

`systemctl reload <svc>`
Reload config

`systemctl kill <svc>`
Send signal to process

Enable & Disable

`systemctl enable <svc>`
Auto-start at boot

`systemctl disable <svc>`
Don't start at boot

`systemctl enable --now`
Enable + start now

`systemctl disable --now`
Disable + stop now

`systemctl mask <svc>`
Permanently prevent

Status & Inspection

`systemctl status <svc>`
State + recent logs

`systemctl is-active <svc>`
active / inactive

`systemctl is-enabled <svc>`
enabled / disabled

`systemctl list-units --type=service`
List all services

`journalctl -u <svc> -f`
Follow log live

Power & Targets

`systemctl reboot`
Reboot the system

`systemctl poweroff`
Shut down cleanly

`systemctl suspend`
Suspend to RAM

`systemctl get-default`
Show default target

`systemctl set-default multi-user.target`
Boot to console only

journalctl — Reading Logs

systemd logging — kernel and all service output in one place

```
$ journalctl -f # follow live
$ journalctl -u sshd # specific service
$ journalctl -u NetworkManager -f # follow service
$ journalctl -p err # errors only
$ journalctl -b # since last boot
$ journalctl -b -1 # previous boot
$ journalctl -k # kernel only
$ journalctl --since "1 hour ago" # time filter
$ journalctl -u sshd -n 50 # last 50 lines
```

Troubleshooting workflow

```
$ systemctl status sshd
→ quick summary + recent log
$ journalctl -u sshd -n 50
→ full recent history
$ systemctl restart sshd
→ fix and restart
```

Priority levels (-p)

emerg / alert / crit	Critical system issues
err	Error conditions
warning	Warning conditions
info	Informational messages
debug	Debug-level messages