OPEN OPERATING SYSTEMS

DISK PARTITIONING & FILESYSTEMS

Practical exercises — from loop devices to Arch Linux install

Part 1

Loop devices & partitioning

Part 2

Creating filesystems

Part 3

Mounting — visual guide

Part 4

Arch Linux install from scratch

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

PART 1

Loop Devices & Partitioning

Create a virtual disk inside a file · partition it · no real hardware needed

fallocate / dd	losetup	fdisk / parted	kpartx / -P flag
Create a raw file = our virtual disk	Connect file to /dev/loop device	Create partition table inside file	Expose partitions as /dev/loop0p1...

PART 1

What is a loop device?

A loop device makes the kernel treat a regular file as a block device — exactly like a real disk

Real disk vs loop device — from the kernel's point of view, they are identical:

	Real disk (/dev/sda)	Loop device (/dev/loop0)
fdisk	✓ works	✓ works identically
mkfs.ext4	✓ works	✓ works identically
mount	✓ works	✓ works identically
Risk	⚠ data loss possible	✓ safe — just a file

LAB 1.1

Create a virtual disk file

Two tools to create a raw file of a specific size — choose one:

Method A — fallocate (fast, recommended)

```
# Create a 2 GB file instantly
$ fallocate -l 2G disk.img

# Verify size
$ ls -lh disk.img
  -rw-r--r-- 1 user 2.0G disk.img

# Check it looks like raw data
$ file disk.img
  disk.img: data
```

Method B — dd (slower, fills with zeros)

```
# Create a 2 GB file filled with zeros
# bs=1M = block size, count=2048 = 2 GB
$ dd if=/dev/zero of=disk.img bs=1M count=2048

# With progress indicator
$ dd if=/dev/zero of=disk.img \
     bs=1M count=2048 status=progress

# Result identical to fallocate
$ ls -lh disk.img
```

fallocate -l SIZE FILE

-l = length
Supports: K, M, G, T
Example: -l 500M

dd if=SRC of=DST

if=input of=output
bs=block size
count=num blocks

Why 2 GB?

Large enough to hold a real OS.
Use any size — even 500M for practice.

LAB 1.2

Attach the file to a loop device

Tell the kernel to expose our file as a block device at /dev/loop0

```
# Attach disk.img to the next available loop device
$ sudo losetup --find --show disk.img
  /dev/loop0

# --find  = pick next free /dev/loopN automatically
# --show  = print the device name chosen

# Verify: list all loop devices
$ losetup -l
  NAME       SIZELIMIT  OFFSET  AUTOCLEAR  RO  BACK-FILE
  /dev/loop0         0       0          0   0  /home/user/disk.img

# The file is now accessible as a block device!
$ lsblk /dev/loop0
  NAME    MAJ:MIN  RM   SIZE  RO  TYPE  MOUNTPOINTS
  loop0     7:0     0     2G   0  loop
```

WHAT HAPPENED

/home/user/disk.img

Regular file
(2 GB of bytes)

/dev/loop0

Block device
(looks like a disk)

fdisk /dev/loop0

Now partitionable
like a real disk!

To detach when done: `sudo losetup -d /dev/loop0` (never forget this!)

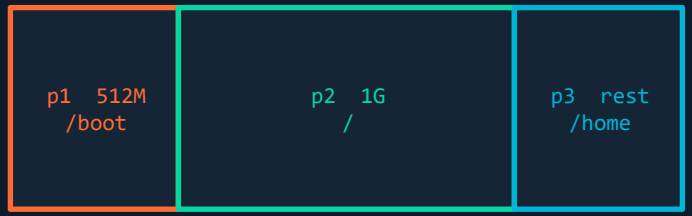
LAB 1.3

Partition the loop device with fdisk

fdisk is interactive — follow the menu to create a GPT partition table

```
$ sudo fdisk /dev/loop0

# Inside fdisk — type these commands:
  g          # Create GPT partition table
  n          # New partition (1 of 3)
    Partition: 1  |  First: [Enter]  |  Last: +512M
  n          # New partition (2 of 3)
    Partition: 2  |  First: [Enter]  |  Last: +1G
  n          # New partition (3 of 3 — uses rest)
    Partition: 3  |  First: [Enter]  |  Last: [Enter]
  p          # Print table — verify before saving!
  w          # Write and exit

# Expose partition nodes
$ sudo losetup -d /dev/loop0
$ sudo losetup -P --find --show disk.img
  /dev/loop0
$ ls /dev/loop0*
  /dev/loop0  /dev/loop0p1  /dev/loop0p2  /dev/loop0p3
```

RESULTING LAYOUT



Key fdisk commands:

g	New GPT table
o	New MBR table
n	New partition
d	Delete partition
p	Print table
t	Change type
w	Write + exit
q	Quit (no save)

LAB 1.4

Alternative: parted (non-interactive)

parted can be scripted — useful for automation and install scripts

```
# All-in-one non-interactive: GPT + 3 partitions
$ sudo parted -s /dev/loop0 \
    mklabel gpt \
    mkpart primary 1MiB 513MiB \
    mkpart primary 513MiB 1537MiB \
    mkpart primary 1537MiB 100%

# -s = script mode (no interactive prompts)
# 1MiB start = alignment (never start at 0!)
# 100% = use all remaining space

# Verify
$ sudo parted /dev/loop0 print
  Number  Start    End      Size    Name     Flags
   1      1049kB   538MB    537MB   primary
   2      538MB    1611MB   1074MB  primary
   3      1611MB   2147MB   536MB   primary
```

Quick reference — partition tools:

fdisk	parted	gdisk	cfdisk
Interactive TUI	CLI + scriptable	fdisk for GPT only	Curses TUI

PART 2

Creating Filesystems

Format partitions · ext4 · vfat · btrfs · what a filesystem actually is

ext4

Most common Linux FS
Journaling, stable, fast

vfat

FAT32 — for EFI /boot
Compatible with all OS

btrfs

Modern Copy-on-Write
Snapshots, compression

xfs

High-performance
Used by RHEL/Fedora

PART 2

What is a filesystem? (concept)

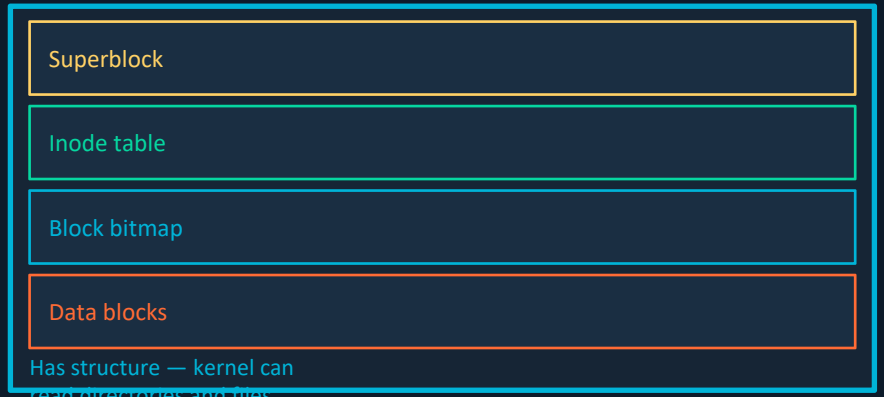
BEFORE mkfs — raw partition

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

No structure — kernel cannot read files or directories.

mkfs .ext4

AFTER mkfs — structured data



Has structure — kernel can read directories and files.

Key filesystem concepts:

Superblock

FS metadata: size, type, block size, free space

Inode

One per file/dir. Stores permissions, timestamps, block locations

Data block

Where file contents live (typically 4 KB each)

Journal

Log of recent changes — used to recover after crashes

LAB 2.1

Format partitions with mkfs

Each partition gets its own filesystem. Use the right type for each role.

```
# Make sure partitions are visible
$ ls /dev/loop0*
  /dev/loop0  /dev/loop0p1  /dev/loop0p2  /dev/loop0p3

# Format p1 as FAT32 (for EFI /boot — required!)
$ sudo mkfs.vfat -F 32 /dev/loop0p1
  mkfs.fat 4.2 (2021-01-31)

# Format p2 as ext4 (root / partition)
$ sudo mkfs.ext4 /dev/loop0p2
  Creating filesystem with 262144 4k blocks and 65536 inodes
  Writing superblocks and accounting information: done

# Format p3 as ext4 with a label (for /home)
$ sudo mkfs.ext4 -L home /dev/loop0p3

# Verify all three have filesystems
$ lsblk -f /dev/loop0
  NAME       FSTYPE  LABEL   UUID   MOUNTPOINTS
  loop0p1    vfat    FAT32   xxxx
  loop0p2    ext4    1.0     yyyy
  loop0p3    ext4    home    zzzz
```

mkfs quick ref

mkfs.ext4 DEV
Format as ext4

mkfs.ext4 -L LABEL
Set volume label

mkfs.vfat -F 32
Format as FAT32

mkfs.btrfs DEV
Format as btrfs

mkfs.xfs DEV
Format as XFS

lsblk -f
Show FS types

blkid DEV
Show UUID+type

fsck DEV
Check/repair FS

PART 3

Mounting — the visual guide

What mount really does · loop device vs real disk · /etc/fstab

The key insight

mount doesn't copy data. It attaches a device socket to a directory. The directory becomes a window into the filesystem.

Physical disk vs loop file

From mount's perspective: identical. Both are block devices. mount doesn't care what's behind /dev/loop0 or /dev/sda.

PART 3

What mount does — visually

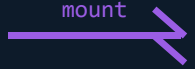
mount DEVICE DIRECTORY → attaches the device's filesystem to that directory in the tree

BEFORE mount /dev/loop0p2 /mnt

```
/
├── home/
├── etc/
└── mnt/
        (empty)
```

`/dev/loop0p2  (ext4, 1 GB)`

Contains: bin/ etc/ usr/ var/...



AFTER mount /dev/loop0p2 /mnt

```
/
├── home/
├── etc/
└── mnt/
        ├── bin/
        ├── etc/
        └── usr/
```

umount /mnt → /mnt becomes empty again · the data on the device is NOT affected

LAB 3.1

Mount loop device partitions

Mount all three partitions and explore their contents

```
# Create mount point directories
$ mkdir -p /mnt/myroot /mnt/myroot/boot /mnt/myroot/home

# Mount root partition first!
$ sudo mount /dev/loop0p2 /mnt/myroot

# Mount /boot (EFI) inside the root
$ sudo mount /dev/loop0p1 /mnt/myroot/boot

# Mount /home inside the root
$ sudo mount /dev/loop0p3 /mnt/myroot/home

# Verify the mount tree
$ findmnt /mnt/myroot
  TARGET             SOURCE        FSTYPE  SIZE
  /mnt/myroot        /dev/loop0p2  ext4    1.0G
  /mnt/myroot/boot   /dev/loop0p1  vfat    512M
  /mnt/myroot/home   /dev/loop0p3  ext4    rest

# Check free space
$ df -h /mnt/myroot /mnt/myroot/boot
```

mount options

`-o ro`

Read-only

`-o rw`

Read-write (default)

`-o noexec`

No execute

`-o loop`

Force loop device

`-t ext4`

Force fs type

`--bind`

Bind mount (mirror dir)

`-o remount`

Remount with new opts

`findmnt`

Show mount tree

`lsblk`

Show block devices

PART 3

Physical disk vs loop file — the difference

From mount's perspective: identical. The difference is only in setup.

	Real physical disk	Loop file (disk.img)
Device node	/dev/sda, /dev/nvme0n1	/dev/loop0
How it appears	Hardware detected at boot	losetup --find --show disk.img
Partitions	/dev/sda1, /dev/sda2...	/dev/loop0p1, /dev/loop0p2...
Partition scan	Automatic	losetup -P or partprobe
Format with mkfs	mkfs.ext4 /dev/sda2	mkfs.ext4 /dev/loop0p2
Mount	mount /dev/sda2 /mnt	mount /dev/loop0p2 /mnt
Risk	⚠ destroys real data	✓ just a file — safe
Portability	Tied to machine	✓ just copy disk.img

Tip: loop files are perfect for practising Arch installs — start over by deleting disk.img and creating a new one.

PART 3

/etc/fstab — auto-mount at boot

fstab tells the kernel what to mount automatically at startup

```
# /etc/fstab — one line per mount point
# DEVICE          MOUNTPOINT  FSTYPE  OPTIONS    DUMP PASS
UUID=xxxx-yyyy     /           ext4    defaults    0    1
UUID=aaaa-bbbb     /boot       vfat    defaults    0    2
UUID=cccc-dddd     /home       ext4    defaults    0    2
tmpfs              /tmp        tmpfs   size=2G     0    0

# Get UUID of a partition:
$ blkid /dev/loop0p2
  /dev/loop0p2: UUID="a1b2c3d4" TYPE="ext4"

# Auto-generate fstab from mounted devices:
$ genfstab -U /mnt/myroot
  UUID=a1b2...  /       ext4  rw,relatime  0  1
  UUID=e5f6...  /boot   vfat  rw,relatime  0  2
  UUID=g7h8...  /home   ext4  rw,relatime  0  2
```

fstab fields

DEVICE
UUID=..., LABEL=...
or /dev/sdaN

MOUNTPOINT
Where to mount:
/, /home, /boot

FSTYPE
ext4, vfat, btrfs
tmpfs, swap

OPTIONS
defaults, ro, noexec
compress=zstd

DUMP
0 = skip backup
(always 0 now)

PASS
fsck order:
0=skip 1=first 2=after

Always use UUID= not /dev/sdaN — device names change between boots. UUIDs never change.

PART 4

Arch Linux
Install from Scratch

loop file · partitions · filesystems · pacstrap · chroot · GRUB

LAB 4.1

Steps 1–3: disk file, partitions, filesystems

```
# STEP 1: Create the disk file (4 GB)
$ fallocate -l 4G arch.img

# STEP 2: Attach + partition
$ sudo losetup -P --find --show arch.img
  /dev/loop0

$ sudo parted -s /dev/loop0 \
    mklabel gpt \
    mkpart ESP fat32 1MiB 513MiB \
    set 1 esp on \
    mkpart primary ext4 513MiB 100%

# STEP 3: Format
# p1 = EFI System Partition (must be FAT32!)
$ sudo mkfs.vfat -F 32 /dev/loop0p1

# p2 = root filesystem
$ sudo mkfs.ext4 /dev/loop0p2

# Verify
$ lsblk -f /dev/loop0
  NAME       FSTYPE  SIZE
  loop0p1    vfat    512M
  loop0p2    ext4    3.5G
```

LAB 4.2

Step 4: mount partitions at /mnt

Create the directory tree and mount each partition into the right place

```
# Mount root partition first
$ sudo mount /dev/loop0p2 /mnt

# Create /boot directory INSIDE the mounted root
$ sudo mkdir -p /mnt/boot

# Mount EFI partition inside
$ sudo mount /dev/loop0p1 /mnt/boot

# Verify
$ findmnt /mnt
  TARGET      SOURCE        FSTYPE  SIZE
  /mnt        /dev/loop0p2  ext4    3.5G
  /mnt/boot   /dev/loop0p1  vfat    512M

# Check space
$ df -h /mnt /mnt/boot
  /dev/loop0p2  3.4G  24K  3.2G  1%  /mnt
  /dev/loop0p1  511M    0  511M  0%  /mnt/boot
```

MOUNT TREE

```
/mnt                    ← loop0p2 (ext4)
├─ boot/                ← loop0p1 (vfat)
├─ bin/
├─ etc/                 ↑ created by
├─ usr/                   pacstrap
└─ var/
```

⚠ Mount / (loop0p2) BEFORE creating /mnt/boot and mounting loop0p1 inside it!

LAB 4.3

Step 5: pacstrap — install base system

pacstrap downloads and installs Arch Linux packages directly into /mnt

```
# -K = initialize a new pacman keyring in /mnt
# base = minimal Arch userland
# linux = the kernel
# linux-firmware = hardware firmware blobs
$ sudo pacstrap -K /mnt base linux linux-firmware

# pacstrap will:
#   1. Create /mnt/etc/pacman.d/gnupg (key database)
#   2. Download packages from Arch mirrors
#   3. Install into /mnt (not into the running system!)
#   4. Run post-install hooks inside /mnt

# Output looks like:
#   :: Synchronizing package databases...
#   (1/N) installing linux ...

# Recommended: add useful tools in the same command
$ sudo pacstrap -K /mnt base linux linux-firmware \
    vim nano grub efibootmgr networkmanager
```

LAB 4.4

Steps 6–7: genfstab and arch-chroot

Generate the filesystem table, then chroot into the new system to configure it

```
# STEP 6: Generate /etc/fstab
$ genfstab -U /mnt >> /mnt/etc/fstab
$ cat /mnt/etc/fstab    # verify!
  UUID=aaaa  /       ext4  rw,relatime  0 1
  UUID=bbbb  /boot   vfat  rw,relatime  0 2

# STEP 7: chroot into the new system
# arch-chroot auto-mounts /proc /sys /dev /run
$ sudo arch-chroot /mnt

# You are now INSIDE the new system!
[root@archiso /]# uname -r   # kernel from /mnt/boot
  6.x.x-arch1-1

# Set timezone
[root@archiso /]# ln -sf /usr/share/zoneinfo/Europe/Warsaw /etc/localtime
[root@archiso /]# hwclock --systohc

# Set hostname
[root@archiso /]# echo 'myhostname' > /etc/hostname

# Set root password
[root@archiso /]# passwd

# Enable NetworkManager
[root@archiso /]# systemctl enable NetworkManager
```

LAB 4.5

Step 8: install GRUB bootloader

GRUB is the first thing that runs after UEFI — it loads the Linux kernel

```
# Inside arch-chroot!

# Install GRUB EFI binary
[root@archiso /]# grub-install \
    --target=x86_64-efi \
    --efi-directory=/boot \
    --bootloader-id=ARCH \
    --removable

# --target=x86_64-efi    EFI mode (i386-pc for BIOS/MBR)
# --efi-directory=/boot  where EFI partition is mounted
# --bootloader-id=ARCH   name in UEFI firmware menu
# --removable            writes /boot/EFI/BOOT/BOOTX64.EFI
#                        (fallback path — essential for loop devices!)

# Generate GRUB config (detects kernels, creates menu)
[root@archiso /]# grub-mkconfig -o /boot/grub/grub.cfg
  Generating grub configuration file ...
  Found linux image: /boot/vmlinuz-linux
  Found initrd image: /boot/initramfs-linux.img
  done

# Verify EFI file was created
[root@archiso /]# ls /boot/EFI/BOOT/
  BOOTX64.EFI
```

LAB 4.6

Finish installation and test with QEMU

Final steps inside chroot, then unmount cleanly and boot the image

```
# Still inside arch-chroot
[root@archiso /]# echo 'en_US.UTF-8 UTF-8' >> /etc/locale.gen
[root@archiso /]# locale-gen
[root@archiso /]# echo 'LANG=en_US.UTF-8' > /etc/locale.conf
[root@archiso /]# exit

# Back on host — unmount in REVERSE order!
$ sudo umount /mnt/boot
$ sudo umount /mnt
$ sudo losetup -d /dev/loop0

# Test with QEMU (optional)
$ qemu-system-x86_64 \
    -drive file=arch.img,format=raw \
    -m 2G \
    -bios /usr/share/ovmf/OVMF.fd \
    -nographic

# Or verify without QEMU:
$ sudo losetup -P --find --show arch.img
$ sudo mount /dev/loop0p2 /mnt
$ sudo mount /dev/loop0p1 /mnt/boot
$ ls /mnt/boot
  vmlinuz-linux  initramfs-linux.img  EFI/  grub/
$ sudo umount /mnt/boot /mnt
$ sudo losetup -d /dev/loop0
```

SUMMARY

Complete command reference

Loop device

```
fallocate -l 4G file.img     # create disk file
losetup -P --find --show f   # attach + scan
losetup -l                   # list loop devices
losetup -d /dev/loop0        # detach
```

Partitioning

```
fdisk /dev/loop0             # interactive
parted -s /dev/loop0 mklabel gpt
parted -s /dev/loop0 mkpart primary 1MiB 513MiB
lsblk /dev/loop0             # verify
```

Filesystems

```
mkfs.vfat -F 32 /dev/loop0p1  # FAT32 for EFI
mkfs.ext4 /dev/loop0p2        # ext4
lsblk -f                      # show FS types
blkid /dev/loop0p2            # show UUID
```

Mounting

```
mount /dev/loop0p2 /mnt       # mount root
mount /dev/loop0p1 /mnt/boot  # mount EFI
findmnt /mnt                  # show tree
umount /mnt/boot && umount /mnt
```

Arch install

```
pacstrap -K /mnt base linux linux-firmware grub efibootmgr
genfstab -U /mnt >> /mnt/etc/fstab
arch-chroot /mnt
```

REFERENCE

Common errors and fixes

`losetup: /dev/loop0 is busy`

losetup -d /dev/loop0
Maybe still mounted: umount first

`mount: /dev/loop0p1 does not exist`

losetup -P --find --show disk.img
The -P flag scans for partition nodes

`mkfs: disk.img contains a filesystem`

Use the PARTITION (/dev/loop0p1), not the disk file
or fdisk first to create partitions

`grub-install: EFI variables not supported`

Add --removable to grub-install
Or: modprobe efivars

`pacstrap: failed to retrieve files`

Check internet: ping archlinux.org
Update mirrors: reflector --save /etc/pacman.d/mirrorlist

`umount: target is busy`

Kill processes: fuser -km /mnt
or lsof +D /mnt
Then try again