OPEN OPERATING SYSTEMS

DISK PARTITIONING & Arch Installation

Practical exercises — from loop devices to Arch Linux install

Part 1

Creating filesystems

Part 2

Mounting — visual guide

Part 3

Arch Linux install from scratch

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

Step 1 — Connect to the Lab Server

Log in via SSH from your terminal:

```
$ ssh user_name@oos.wimic.agh.edu.pl
```

Reminder:
Use any terminal: Linux/macOS have one built in.
Windows: use PuTTY or Windows Terminal.
To log in to your account, use the login and password from your email.

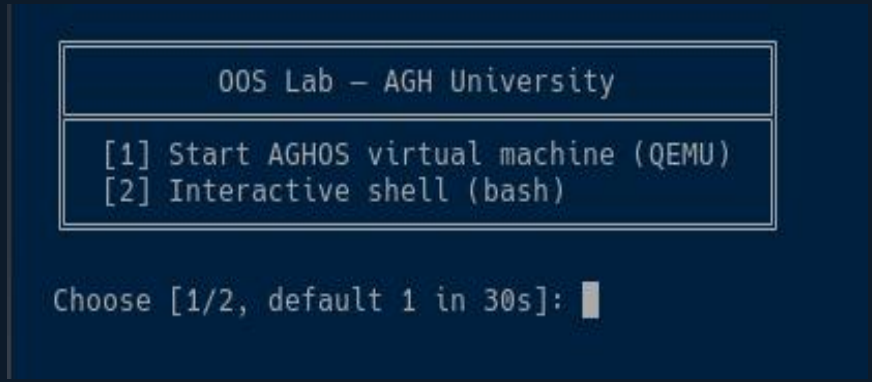
Then select:
Option 1 - Run the AGHOS ISO via SSH
Option 2 - Simply log in to the system.

SETUP

Step 2 — Select AGHOS Boot Menu

After logging in to the lab server, type 1 and press Enter.

This launches your personal QEMU virtual machine session,
booted from the AGHOS ISO image.



If you select 1 (run from ISO), after AGHOS boots, you'll see the AGHOS login screen. Log in with username: aghos · No password required.

SETUP

Step 3 — Select Language / Keyboard Layout

Select language you want from the language menu.

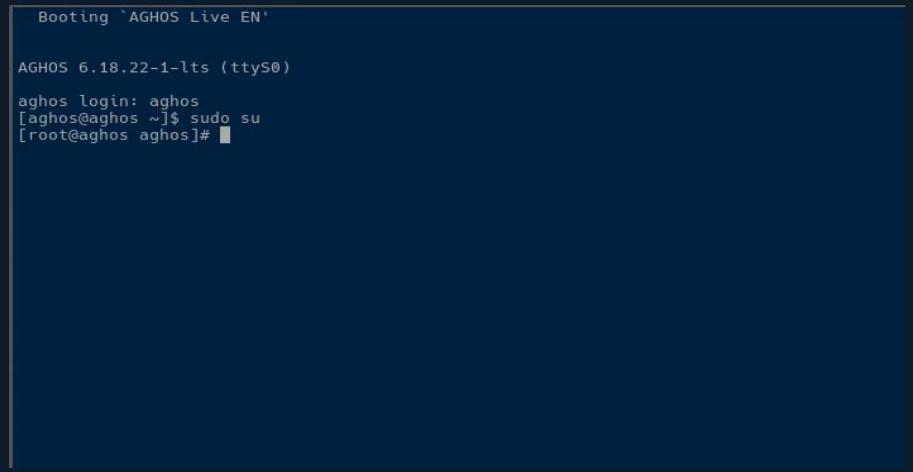
This sets the keyboard layout for the live session.

Step 4 — Login and Gain Root Privileges

After booting, log in as aghos — no password required.
Then escalate to root:

```
$ sudo su
```

Note:
Initial boot may take several minutes — the system image is pulled over the network.
This is normal. Wait until the shell prompt appears.



ADVANCED

How the Lab Infrastructure Works

What happens when you choose option 1:

The server starts a dedicated QEMU virtual machine for each student, booting from the AGHOS ISO. You see the boot output in your terminal because GRUB detects headless mode and routes all output via serial port.

To end your session:
Do NOT use exit — that only logs you out of the shell.
Instead, shut down the VM properly:

```
$ shutdown -h now
```

Option 2 at login: plain host shell (no QEMU).

The virtual disk:

The 100 GB disk.img file on the host appears as /dev/vda inside your QEMU session — it is your virtual hard drive.

If disk.img is too large to download, create a smaller image, copy only your data into it, and download that instead. Figuring out how is left as an exercise!

Each student has their own isolated QEMU session — your changes do not affect anyone else.

PART 1

Loop Devices & Partitioning

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

fallocate / dd

Create a raw file
= our virtual disk

losetup

Connect file to
/dev/loop device

fdisk / parted

Create partition
table inside file

kpartx / -P flag

Expose partitions
as /dev/loop0p1...

How to create a virtual disk file

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

Method A — fallocate (fast, recommended)

```
# Check the free space on the disk
$ df -h


$ fallocate -l 30G 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

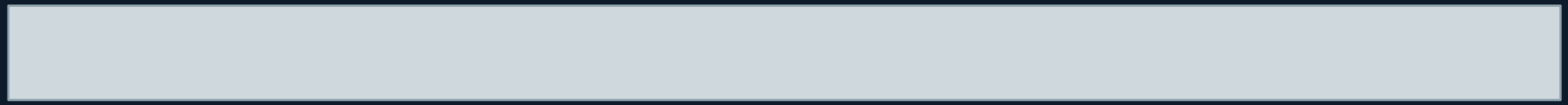
dd if=SRC of=DST

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

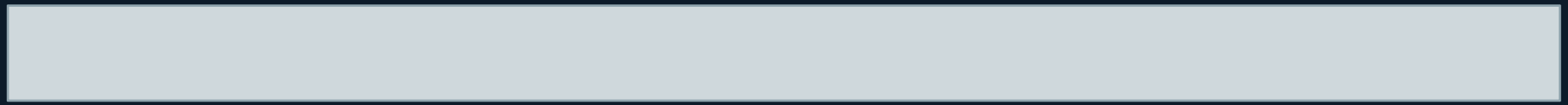
Why 2 GB?

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

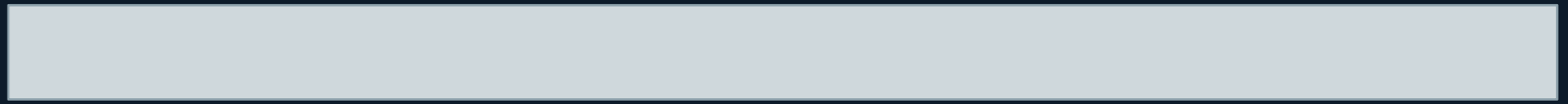
Drive 1 (/dev/nvme0n1); 1TB (very fast SSD drive)



Drive 2 (/dev/nvme1n1); 1TB (very fast SSD drive)

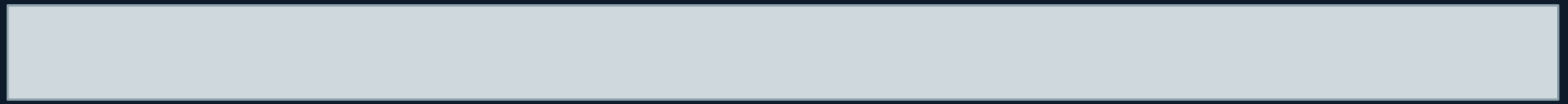


Drive 3 (/dev/sda); 10TB (slow SATA drive - cheap but spacious)

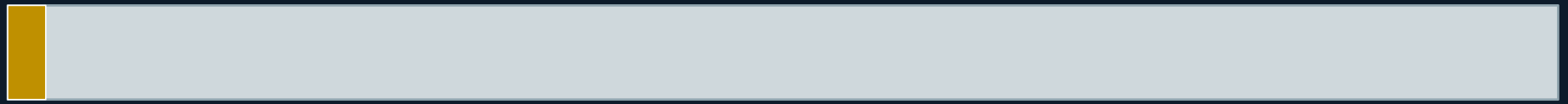


PART 1

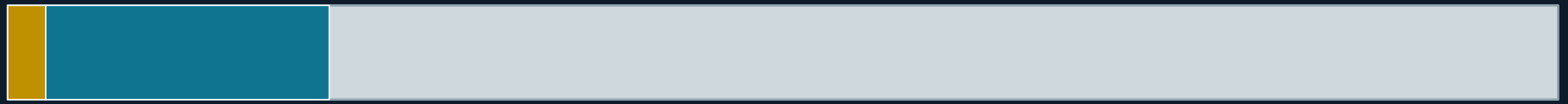
Drive 1 — Partition Plan



Drive 2 — Games Partition

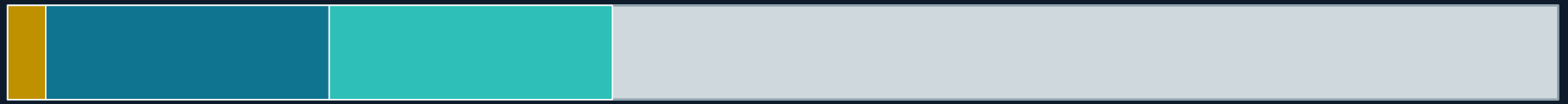


Drive 3 — Media Partition



PART 1

All Drives — Partition Overview



PART 1

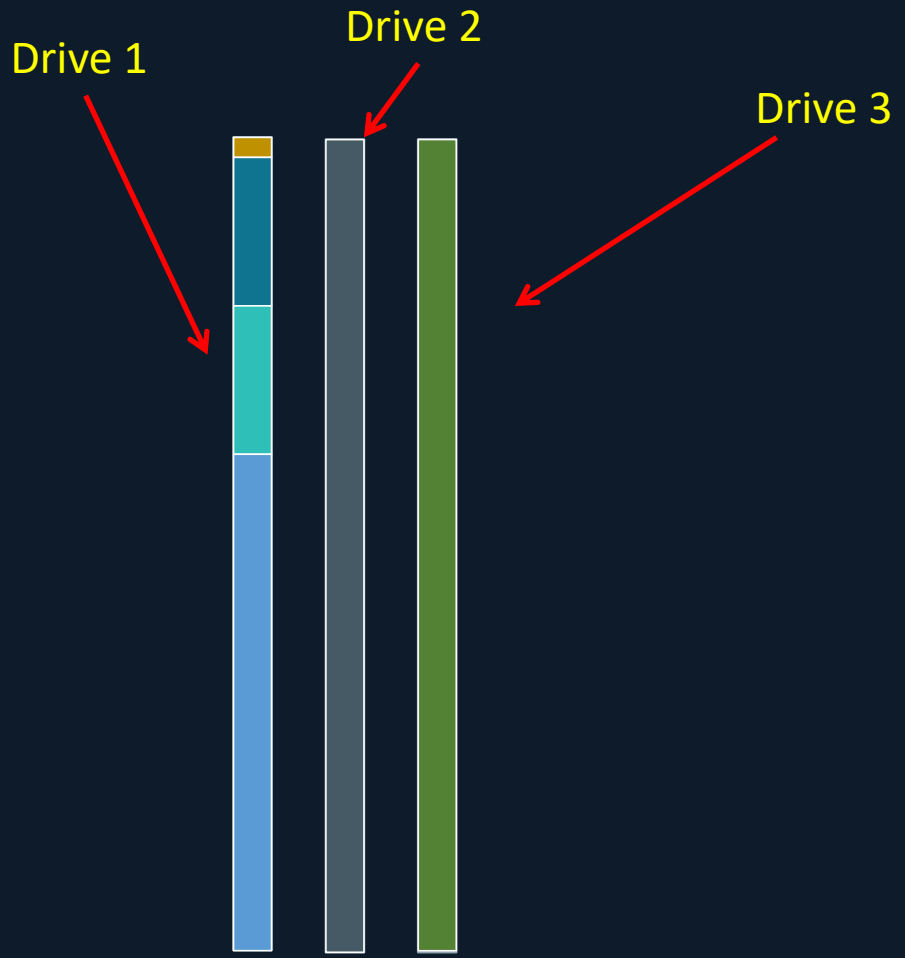
All Drives — Before Partitioning

Partition Assignment Summary

very fast SSD drive - for /boot, root (/), /var and /home

very fast SSD drive - for /home/user_name/games

slow SATA drive - spacious (10TB) for /home/user_name/media


Drive 1
Drive 2
Drive 3

Linux Filesystem Tree — Booted Live System

Live or installed Linux

```
/
├── /bin
├── /boot
├── /etc
├── /lib
├──**/mnt**
├── /usr
├── /var
├── /tmp
├── /root
└── /home/
```

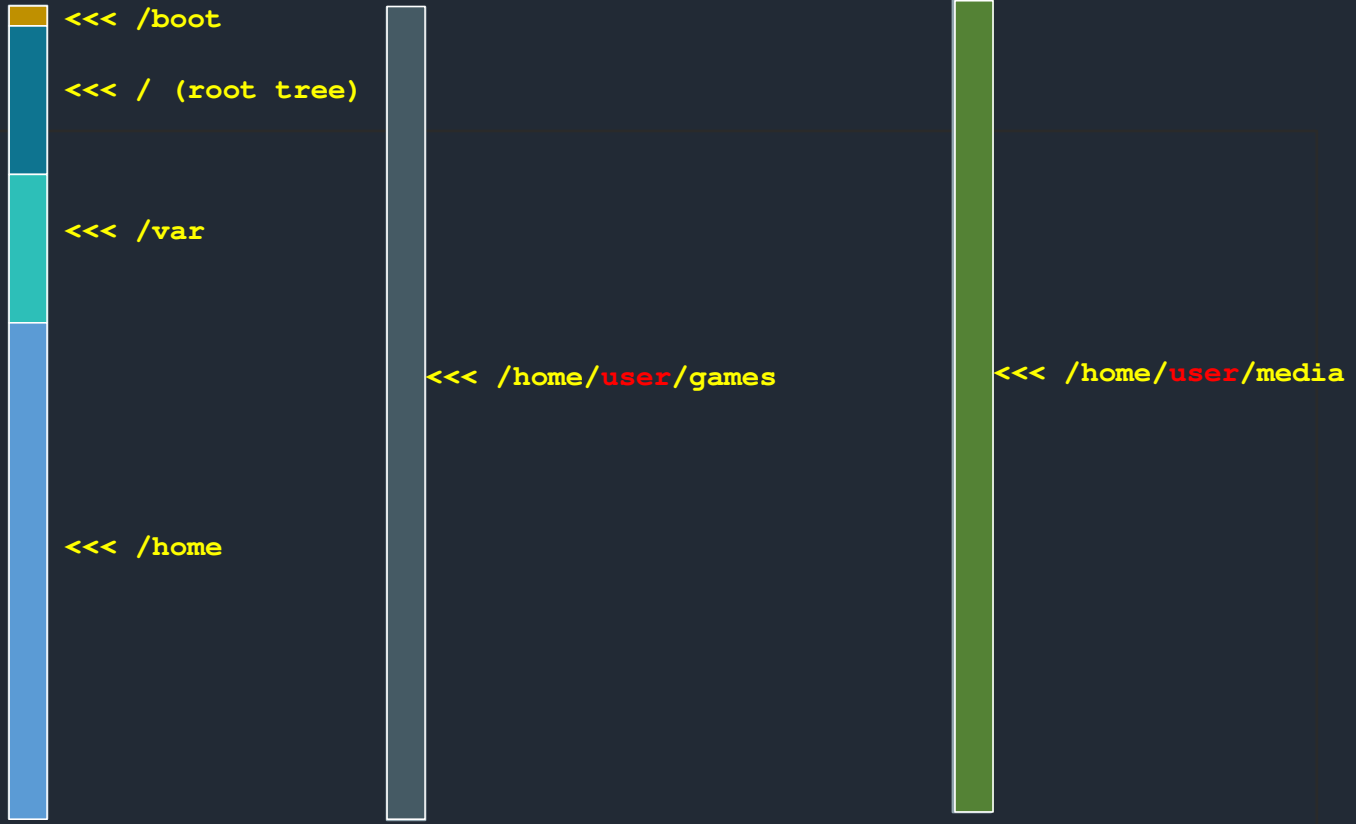
Partition Assignment Plan

This is the partition assignment plan — no mounts have happened yet.

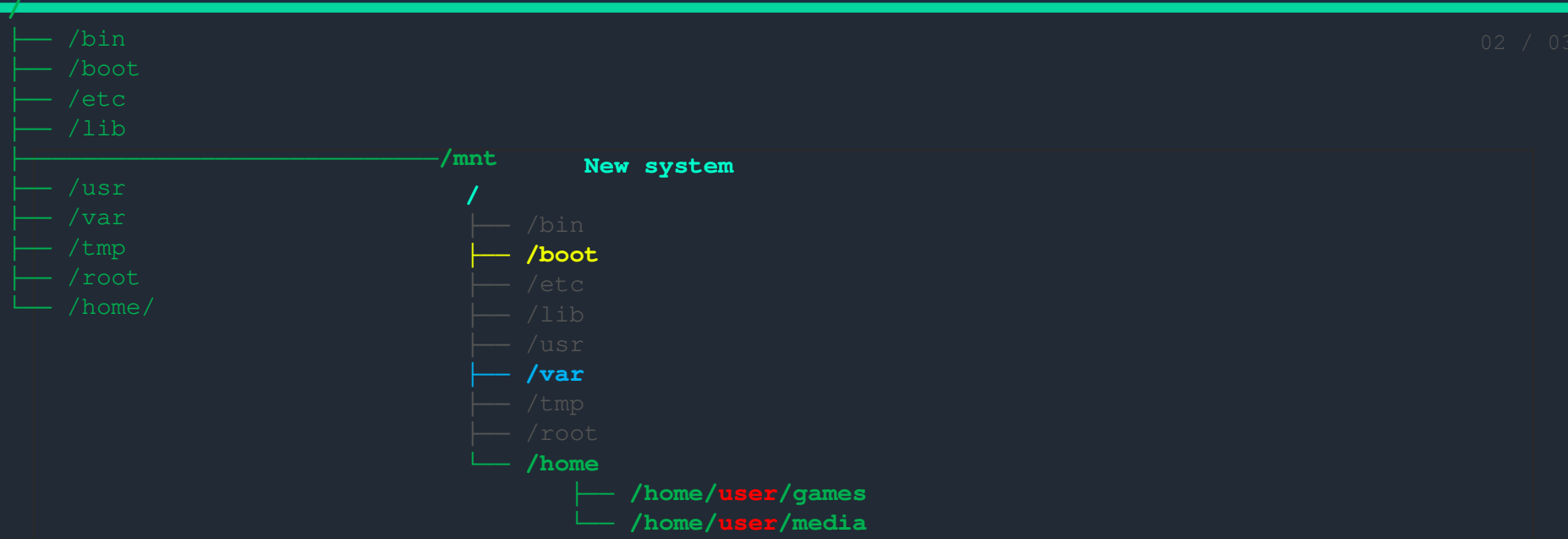
Booted Live system

Live or installed Linux

```
/
├── /bin
├── /boot
├── /etc
├── /lib
├──/mnt
├── /usr
├── /var
├── /tmp
├── /root
└── /home/
```



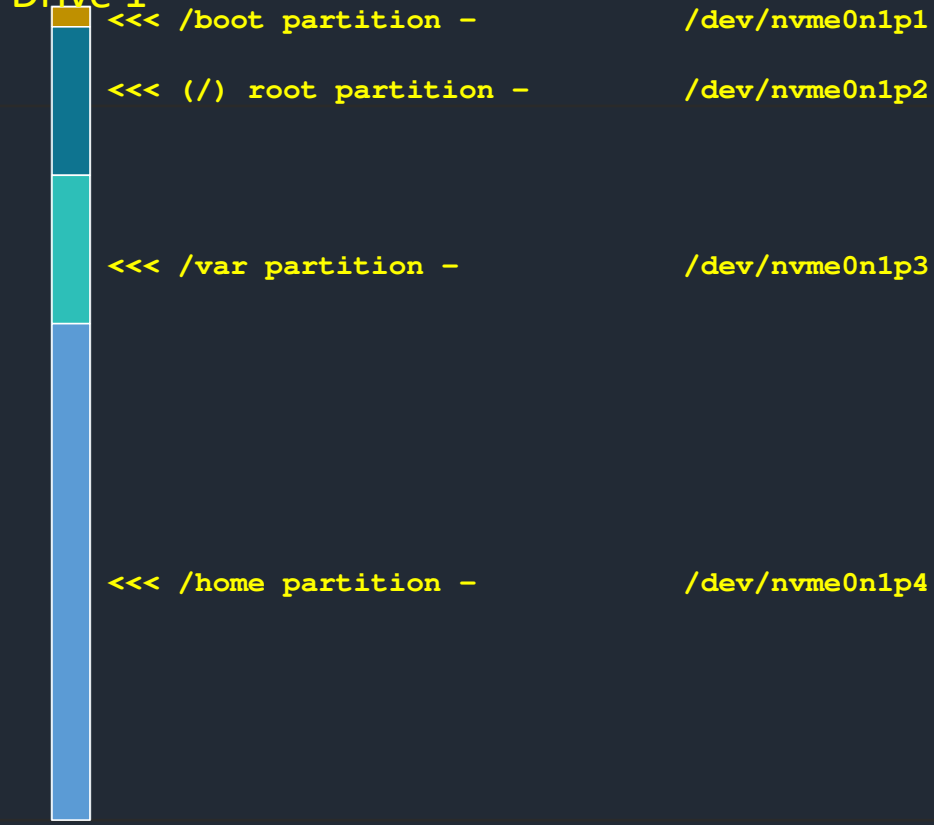
Two Trees — Live System vs New System



Step 1 — Mount Root Partition

Live or installed Linux

```
/
├── /bin
├── /boot
├── /etc
├── /lib
├──────────────/mnt        New system
├── /usr                    /
├── /var                    ├── /bin
├── /tmp                    ├── /boot
├── /root                   ├── /etc
└── /home/                  ├── /lib
                            ├── /usr
                            ├── /var
                            ├── /tmp
                            ├── /root
                            └── /home
                                ├── /home/user/games
                                └── /home/user/media
```



mount — Command Anatomy

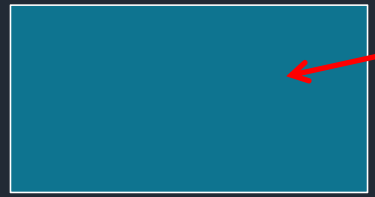
command:	device:	mountpoint:
mount	/dev/nvme0n1p2	/mnt

After Mounting Root — Empty /mnt

Live or installed Linux

/
- /bin
- /boot
- /etc
- /lib
- /mnt
- /usr
- /var
- /tmp
- /root
- /home/

New system



<<< /boot partition – /dev/nvme0n1p1

<<< (/) root partition – /dev/nvme0n1p2

<<< /var partition – /dev/nvme0n1p3

<<< /home partition – /dev/nvme0n1p3

Create Mount Point Directories

Before mounting other partitions, create the necessary directories inside the already-mounted root at /mnt.

command:	where:
mkdir	/mnt/boot
mkdir	/mnt/var
mkdir	/mnt/home

Mount Point Directories Created

Live or installed Linux

```
/
├── /bin
├── /boot
├── /etc
├── /lib
├──────────/mnt
├── /usr
├── /var
├── /tmp
├── /root
└── /home/
```

New system

```
/mnt
├── /boot
├── /var
└── /home
```

<<< /boot partition - /dev/nvme0n1p1

<<< (/) root partition - /dev/nvme0n1p2

<<< /var partition - /dev/nvme0n1p3

<<< /home partition - /dev/nvme0n1p3

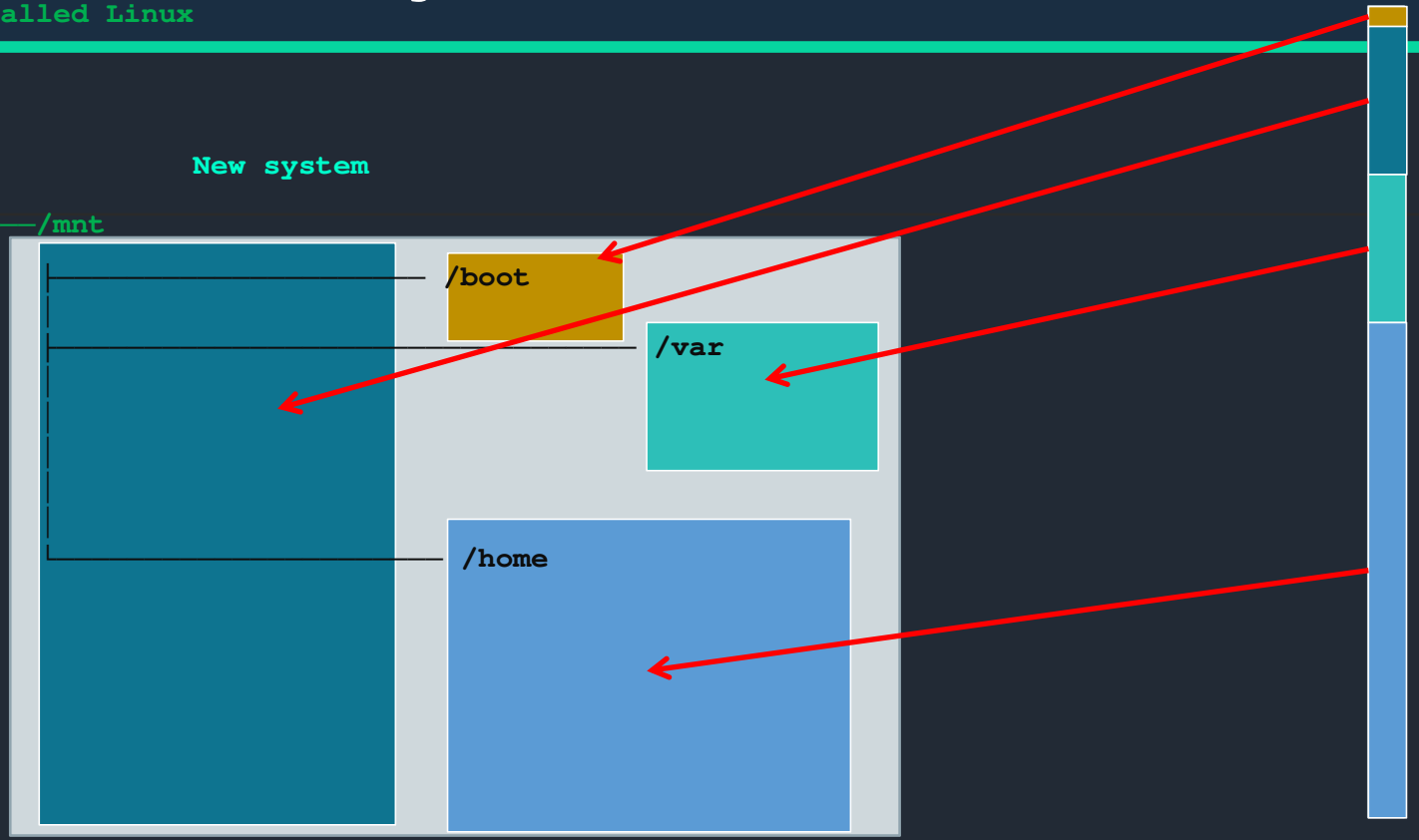
Mount boot, var, home Partitions

command:	device:	mountpoint:
mount	/dev/nvme0n1p1	/mnt/boot
mount	/dev/nvme0n1p3	/mnt/var
mount	/dev/nvme0n1p4	/mnt/home

Drive 1 Fully Mounted

Live or installed Linux

/
- /bin
- /boot
- /etc
- /lib
- /mnt
- /usr
- /var
- /tmp
- /root
- /home/

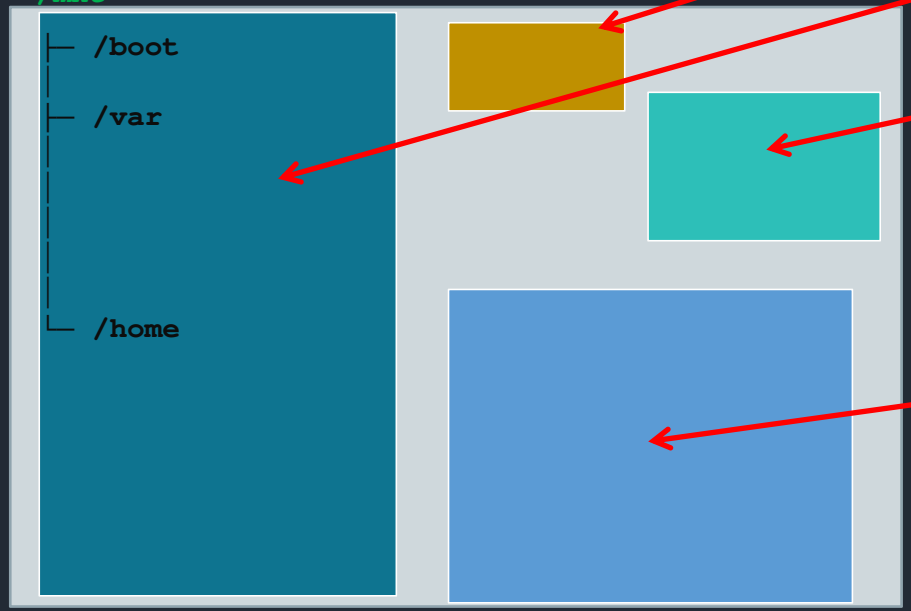


Drive 1 Not Mounted

Live or installed Linux

/
- /bin
- /boot
- /etc
- /lib
- /mnt
- /usr
- /var
- /tmp
- /root
- /home/

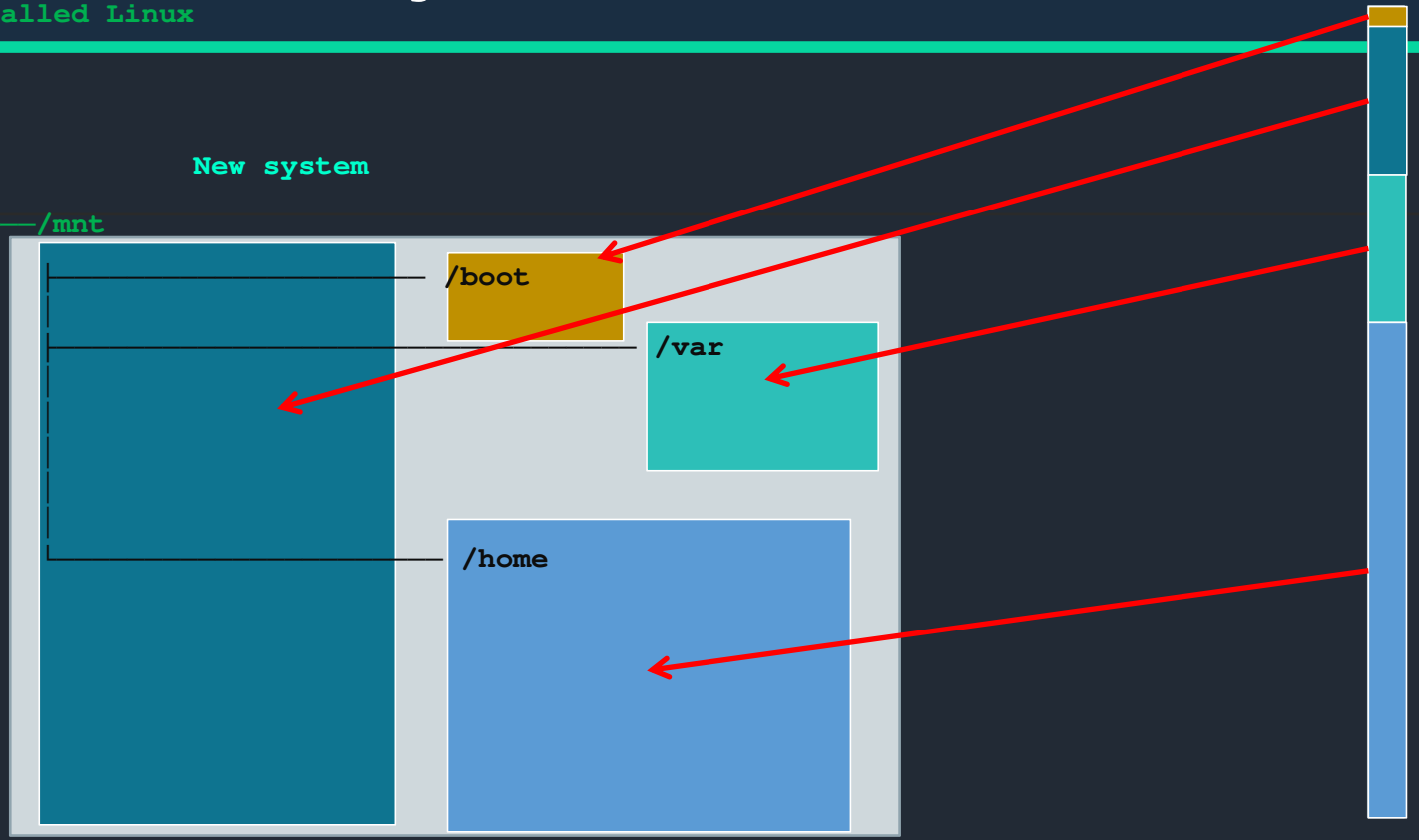
New system



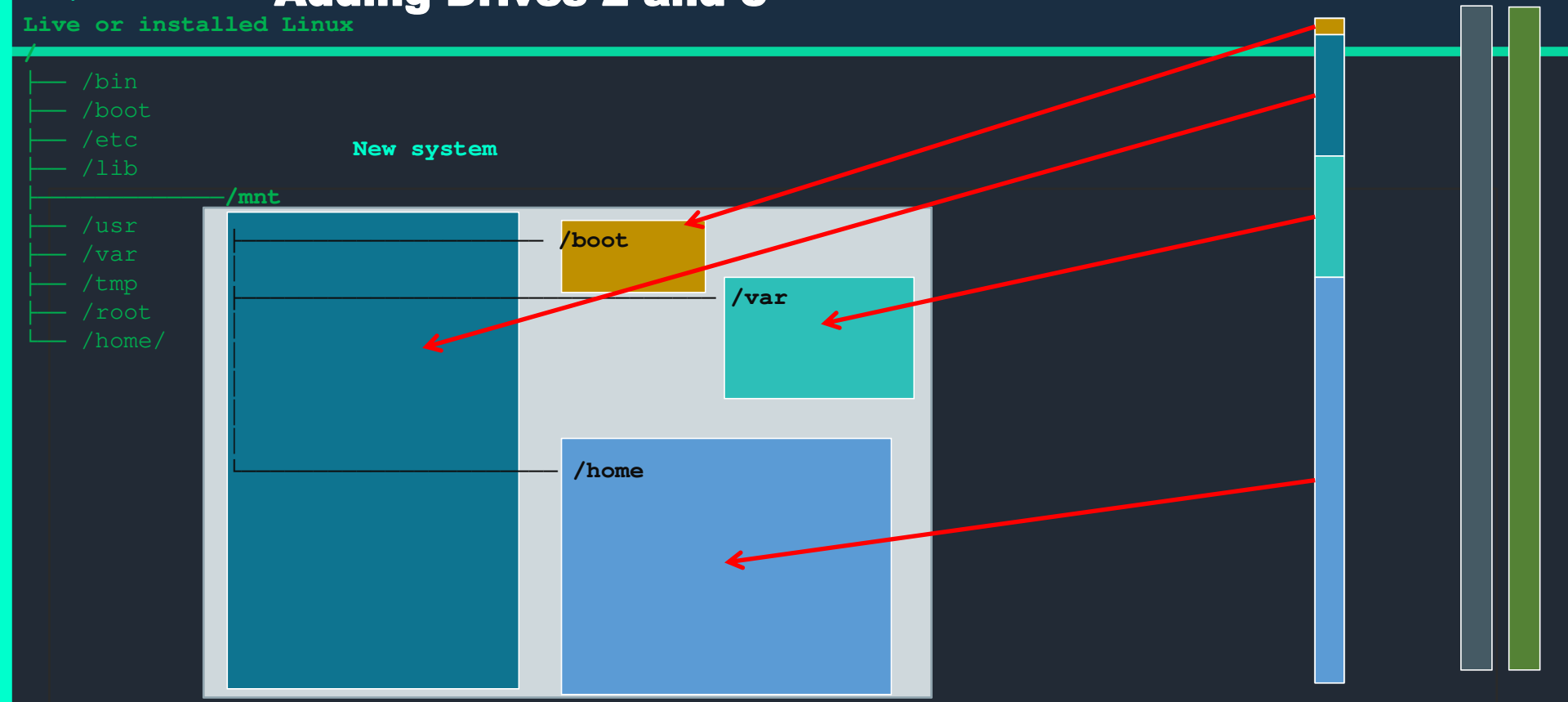
Drive 1 Fully Mounted

Live or installed Linux

/
- /bin
- /boot
- /etc
- /lib
- /mnt
- /usr
- /var
- /tmp
- /root
- /home/



Adding Drives 2 and 3



We also have two additional drives to mount inside /home — one fast NVMe for games, one large SATA for media.

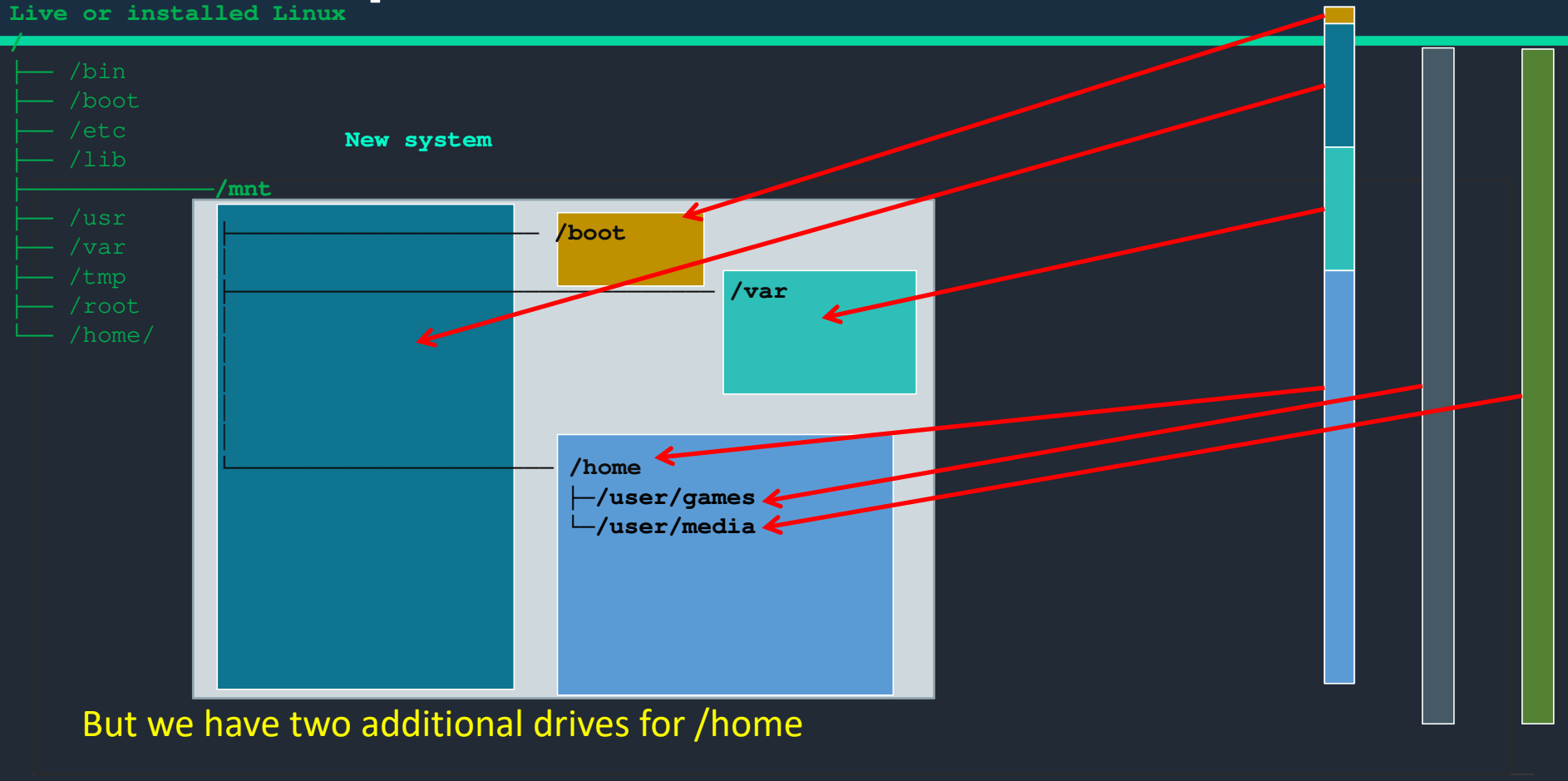
Mount /home/games and /home/media

Create mount point directories inside /home, then mount the two extra drives into them.

```
mkdir     -p      /mnt/home/user_name/games
mkdir     -p      /mnt/home/user_name/media

mount     /dev/nvme1n1p1          /mnt/home/user_name/games   <<< SSD drive for games
mount     /dev/sda1          /mnt/home/user_name/media   <<< SATA drive for media
```

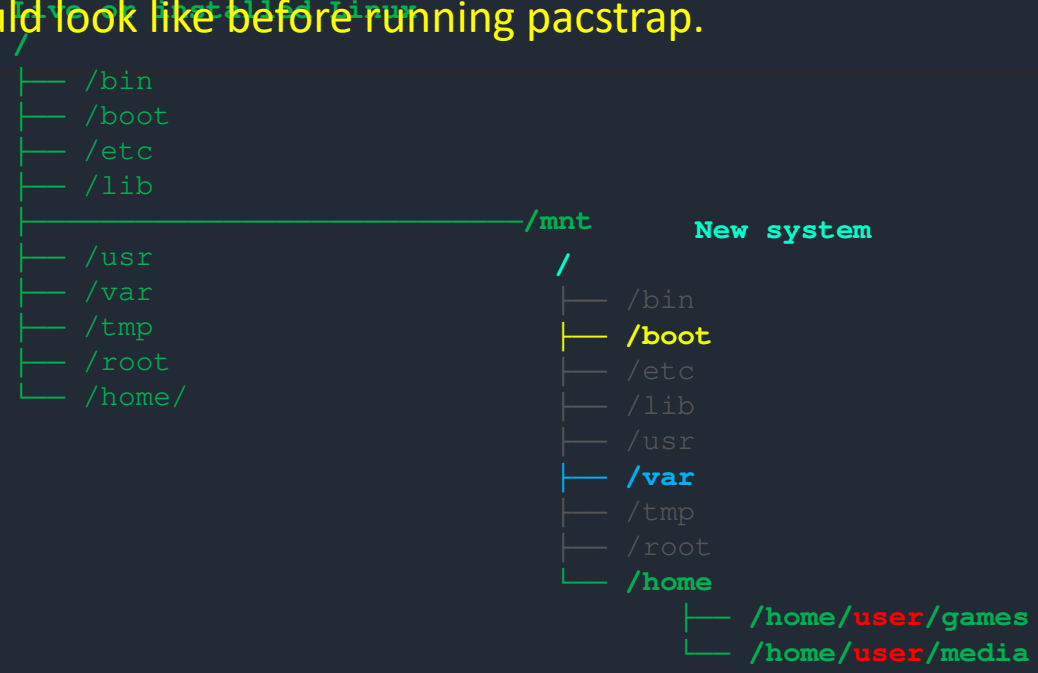
Complete Mount Tree — All Drives



But we have two additional drives for /home

Ready for pacstrap

All partitions mounted — ready for pacstrap. This is what your mount tree should look like before running pacstrap.



PART 4

Arch Linux
Install from Scratch

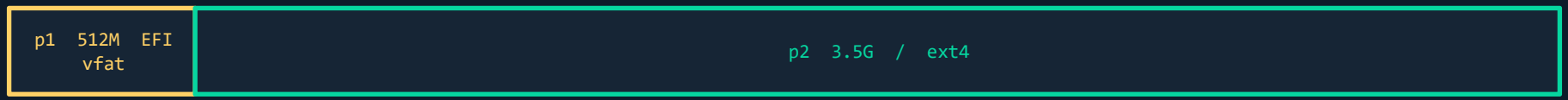
loop file · partitions · filesystems · pacstrap · chroot · GRUB

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 4.2b

Mounting a partition inside a user home directory & /etc/fstab

Mounting Inside User Home & /etc/fstab

How to permanently mount a partition into /home/username/directory — and configure fstab mount options

Step 1 — Create the mount point directory

The directory must exist before mounting:

```
$ mkdir -p /home/alice/data
```

Step 2 — Test-mount manually first

Always verify the partition works before editing fstab:

```
$ sudo mount /dev/sdb1 /home/alice/data
$ ls /home/alice/data   # verify it works
$ sudo umount /home/alice/data
```

Step 3 — Get the partition UUID

Always use UUID in fstab — device names (/dev/sdb1) can change after reboot:

```
$ blkid /dev/sdb1
  /dev/sdb1: UUID="a1b2c3d4-..." TYPE="ext4"
```

Step 4 — Add an entry to /etc/fstab

```
$ sudo nano /etc/fstab
```

fstab entry format — 6 fields:

```
UUID=a1b2...  /home/alice/data  ext4  defaults,nofail  0  2
  [device]        [mountpoint]      [fs]  [options]      [dump]
[pass]
```

Key mount options (field 4):

`defaults` rw, suid, dev, exec, auto, nouser, async

`nofail` boot succeeds even if drive is missing

`noatime` do not update access timestamps (faster)

`uid=1000,gid=1000` force ownership (FAT32/NTFS only)

`x-systemd.automount` mount on first access (lazy)

dump / pass fields:

`dump` = 0 (always 0 — legacy backup flag, ignored today)

`pass` = 0 skip fsck on boot (use for extra drives)

`pass` = 1 fsck root (/) first

`pass` = 2 fsck other partitions after root

Step 5 — Test the fstab entry without rebooting

```
$ sudo mount -a          # mount all fstab entries
$ findmnt /home/alice/data   # verify mounted
```

⚠ **WARNING: a typo in fstab can prevent the system from booting!**

Always run `mount -a` to catch errors before reboot.

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
```

Minimum: base linux linux-firmware · Also install: grub efibootmgr networkmanager vim

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
```

arch-chroot is smarter than plain chroot — auto-mounts /proc /sys /dev /run inside the new root

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
```

--removable is essential for loop device installs — without it GRUB registers with the HOST UEFI and won't be portable

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
```

Unmount order: reverse of mount order. umount /mnt/boot then umount /mnt — never the other way!

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
grub-install --target=x86_64-efi --efi-directory=/boot --removable
grub-mkconfig -o /boot/grub/grub.cfg
```

Complete command reference

Arch install

```
pacstrap -K /mnt base linux linux-firmware grub efibootmgr net-tools dhclient nano mc
```

Complete command reference

Arch install

```
pacstrap -K /mnt ntfs-3b firejail debugedit spotify-launcher fakeroot k3b git ktorrent cargo patch sof-firmware vlc steam signal-desktop nextcloud-client
extra/kde-accessibility-meta extra/kde-applications-meta extra/kde-development-environment-meta extra/kde-education-meta extra/kde-games-meta extra/kde-
graphics-meta extra/kde-multimedia-meta extra/kde-network-meta extra/kde-office-meta extra/kde-pim-meta extra/kde-sdk-meta extra/kde-system-meta
extra/kde-utilities-meta extra/kdevelop-meta extra/plasma-meta extra/plasma-meta plasma-nm
```

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