Gentoo Linux

Building a Custom Rescue USB System with Gentoo

From zero to bootable rescue drive — step by step

01

Why Gentoo?

Philosophy · History · Renaissance

The Brioni Suit of Linux Distributions

Generic Distribution

Compiled once for every CPU on the planet.

Like buying a suit off the rack — it fits, but not perfectly.

Ubuntu, Fedora, Arch:
Same binary runs on Pentium 4 and Threadripper and other CPU's.

Gentoo

Every package compiled specifically for YOUR CPU, with only the features YOU need.

Like a Brioni suit — cut for your measurements.

Binaries compiled for your specific CPU:
-march=znver3 -O2 CPU_FLAGS_X86="avx2 fma"

~25 Years of Gentoo — Then, Lost & Back Again

~2000–2008	Peak popularity. RAM was expensive, CPUs were slow. Every % of performance mattered. Gentoo users squeezed every cycle.
2008–2020	Lost momentum. CPU performance grew faster than software needs. Why spend hours compiling when a binary is fast enough?
2020–now	Renaissance! GPU/RAM prices soared. LLM inference demands every FLOP. -march=native + AVX2/FMA flags matter again.

Portage & The Handbook

Portage — arguably the best package manager

- Source-based: compile with YOUR flags
- USE flags: per-package feature control
- Dependency resolver handles complex graphs
- Supports binary packages too (--getbinpkg)
- Atomic installs, preserved-libs, slot support
- @world, @system, @selected sets

The Gentoo Handbook

- One of the best-maintained Linux install docs
- Covers every architecture
- Community-verified, constantly updated
- https://wiki.gentoo.org/wiki/Handbook

Gentoo community: IRC, forums, mailing lists — very active and helpful even in 2024.

02

Stages & Profiles

Your first architectural decision

Stage1, Stage2, Stage3 — The Bootstrap History

Originally Gentoo had three bootstrap stages. Today only stage3 is used, but understanding the history explains the naming.

Stage 1

Minimal cross-compiler seed

What: A tiny tarball (~10 MB) containing only a C compiler and basic tools — just enough to build stage2 from scratch.

How: Compile a cross-compiler → use it to compile the entire stage2 toolchain from source.

✕ **Deprecated ~2006**

Used by Gentoo developers to bootstrap new CPU architectures. Took 6–24h on typical hardware of the era.

Stage 2

Partial system — toolchain only

What: A tarball (~50 MB) with a working native GCC, libc and basic utilities — no kernel, no init, no package manager.

How: Compile everything else (Portage, init, libraries, userspace) on top of this toolchain.

✕ **Deprecated ~2006**

Skipped stage1 but still required bootstrapping all userspace tools manually. 2–8h before first emerge.

Stage 3

Complete minimal base system ☆

What: A tarball (~275 MB): GCC, glibc, Portage, OpenRC/systemd, coreutils, bash. Ready to chroot and emerge immediately.

How: Unpack → configure make.conf → chroot → sync Portage → emerge. No base compilation needed.

✓ **Current standard since ~2006**

What everyone uses today. Name '3' preserved for historical continuity. Devs still use stage1 for new architectures (RISC-V, LoongArch).

Stage3 Variants — Detailed Comparison

Variant	Init / Profile	Key characteristics	Best for
stage3-amd64-openrc ☆ Our choice	OpenRC init Minimal base Full control	Everything manual. Max learning. Fastest boot.	This course Embedded · Servers
stage3-amd64-systemd	systemd init Minimal base Same packages	Familiar to Ubuntu/Fedora users. Journald logging.	Desktop from Ubuntu/Fedora
stage3-amd64-desktop-openrc	OpenRC init +60 USE flags pre-configured	X11, audio, dbus, polkit enabled by default.	Quick desktop, less config
stage3-amd64-desktop-systemd	systemd init +60 USE flags pre-configured	Most similar to binary distros. GNOME-ready.	Easiest path to full desktop
stage3-amd64-hardened	OpenRC · SSP PIE, RELRO compiler flags	Extra security flags. Compatible with most packages.	Production servers Security research
stage3-amd64-nomultilib	No 32-bit libs 64-bit only Smaller footprint	No lib32, no Wine, no Steam. Purist 64-bit.	Containers Minimal servers
stage3-amd64-musl	musl libc not glibc Alternative ABI	Not all pkgs compile with musl. Better for embedded.	Containers · Embedded Security-focused
stage3-amd64-hardened+selinux	SELinux MAC + hardened Strictest option	Mandatory Access Control. Complex configuration.	High-security servers Gov / military

All variants produce the same end result if configured identically — the difference is only in default USE flags and init system.

What is Stage3?

Stage3 is a ~275 MB compressed tarball containing a minimal pre-compiled base system — your starting seed.

	Arch Linux	Gentoo
Base install	pacstrap /mnt base	tar xpvf stage3-*.tar.xz -C /mnt
What you get	Compiled binaries	Compiled binaries (seed only)
Customisation	After install	From the very first package
Init system	systemd (default)	Choose: OpenRC or systemd

Download methods:

1. lynx https://distfiles.gentoo.org/releases/amd64/autobuilds/current-stage3-amd64-openrc/
2. Browser → gentoo.org/downloads → right-click Stage3 link → copy URL → curl -O <url>

Stage3 Variants — Your First Decision

stage3-amd64-openrc ☆ Our choice

Minimal base + OpenRC init. Full control, maximum learning.

stage3-amd64-desktop-openrc Desktop shortcut

Pre-configured USE flags for GUI apps. Less to configure.

stage3-amd64-systemd Systemd fans

Minimal base with systemd instead of OpenRC.

stage3-amd64-hardened Security focus

Compiler hardening flags (PIE, SSP, RELRO). Production servers.

stage3-amd64-nomultilib 64-bit only

No 32-bit compatibility. Smaller, cleaner.

stage3-amd64-musl Alternative libc

musl instead of glibc. Embedded, containers, security.

03

Disk, Mount & Chroot

Partitioning · fstab · Virtual filesystems

Partitioning — GPT Layout (like in Arch)

Device	Size	Filesystem	Mount	Purpose
/dev/vda1	1 GB	vfat (FAT32)	/boot	EFI System Partition — required for UEFI boot
/dev/vda2	8 GB	swap	[SWAP]	Virtual memory — see next slide for why
/dev/vda3	~49 GB	ext4	/	Root filesystem — everything lives here

Note: In QEMU the disk appears as /dev/vda (virtio driver), not /dev/sda. On real hardware USB drives appear as /dev/sdb, /dev/sdc etc. — always verify with lsblk first!

```
fdisk /dev/vda → g (GPT) → n +1G → t 1 (EFI) → n +8G → t 2 19 (swap) → n (rest) → w

mkfs.fat -F32 /dev/vda1 | mkswap /dev/vda2 && swapon /dev/vda2 | mkfs.ext4 /dev/vda3

mount /dev/vda3 /mnt/gentoo && mkdir -p /mnt/gentoo/boot && mount /dev/vda1 /mnt/gentoo/boot
```

Why Create a Swap Partition?

Compilation safety net

Building LLVM, Rust, Firefox requires 4–8 GB RAM per thread. Without swap, the kernel OOM-killer terminates the compiler. We experienced this live — LLVM killed cc1plus.

Hibernation support

To hibernate (suspend-to-disk) the system needs a swap partition at least as large as installed RAM. A rescue system that can hibernate is useful on laptops.

RAM extension

On machines with 4 GB RAM, swap allows running GParted + Firefox + file manager simultaneously without crashes.

Kernel behaviour

Linux uses swap proactively (swappiness). Even with 16 GB RAM, having swap prevents pathological OOM situations during memory spikes.

The mount Command — Far Beyond /dev/sda1

Type	Example	Use case
Block device	`mount /dev/sdb1 /mnt/usb`	Classic USB/disk mount
Loop device	`mount -o loop image.iso /mnt/iso`	Mount ISO/IMG files as block devices
Device mapper	`mount /dev/mapper/cryptroot /mnt`	LVM volumes, LUKS encrypted disks
NFS	`mount -t nfs 192.168.1.10:/export /mnt/nfs`	Network File System — remote directory
SSHFS	`sshfs user@host:/path /mnt/remote`	Mount remote dir over SSH (FUSE)
SMB/CIFS	`mount -t cifs //server/share /mnt/smb`	Windows shares, Samba
WebDAV	`mount -t davfs https://cloud.example.com/dav /mnt/cloud`	Nextcloud, pCloud, Box, Yandex
Bind mount	`mount --bind /source /target`	☆ Same dir at two locations — used in chroot!
Remount	`mount -o remount,rw /`	Change options on mounted fs (e.g. read-only → rw)
tmpfs	`mount -t tmpfs -o size=2G tmpfs /tmp`	RAM filesystem — no disk I/O

/etc/fstab — Persistent Mount Configuration

```
# <device>                                <mount>   <type>  <options>          <dump>  <pass>
UUID=AAAA-BBBB                            /boot     vfat    defaults,noatime   0       2
UUID=12345678-1234-1234-1234-123456789abc /         ext4    defaults,noatime   0       1
UUID=12345678-1234-1234-1234-abcdefabcdef none      swap    sw                 0       0
```

Column 1 — Device

UUID= preferred (stable)
/dev/vda1 (unstable!)
LABEL=boot
//server/share (SMB)
server:/path (NFS)
https://... (WebDAV)

Column 2 — Mountpoint

Absolute path
none for swap
none for bind mounts
without mount point

Column 3 — Type

ext4, xfs, btrfs, vfat
swap, tmpfs
nfs, cifs, davfs
auto (autodetect)

Column 4 — Options

defaults, noatime
ro/rw, noexec, nosuid
user (allow non-root)
nofail (boot if missing)
compress=zstd (btrfs)

Col 5 — Dump

0 = no backup
1 = dump backup
(almost always 0)

Col 6 — Pass (fsck)

0 = skip check
1 = root only
2 = other partitions

Entering the Chroot

chroot = change root. The shell believes /mnt/gentoo IS /. Identical concept to Arch's arch-chroot — but done manually.

1. Mount virtual filesystems

```
mount --types proc /proc /mnt/gentoo/proc
mount --rbind /sys /mnt/gentoo/sys && mount --make-rslave /mnt/gentoo/sys
mount --rbind /dev /mnt/gentoo/dev && mount --make-rslave /mnt/gentoo/dev
mount --bind /run /mnt/gentoo/run && mount --make-slave /mnt/gentoo/run
```

2. Copy DNS (BEFORE chroot!)

```
cp --dereference /etc/resolv.conf /mnt/gentoo/etc/resolv.conf
# --dereference: follow symlinks (needed on systemd-resolved systems)
```

3. Enter chroot

```
chroot /mnt/gentoo /bin/bash
source /etc/profile
export PS1="(chroot) $PS1"   # Always know where you are!
```

What arch-chroot Actually Does

arch-chroot is a convenience wrapper. Under the hood it runs exactly these mount commands — then drops you into chroot.

arch-chroot /mnt/gentoo

```
# 1. mount --bind virtual filesystems
mount -t proc proc      /mnt/gentoo/proc
mount --rbind /sys      /mnt/gentoo/sys
mount --make-rslave     /mnt/gentoo/sys
mount --rbind /dev      /mnt/gentoo/dev
mount --make-rslave     /mnt/gentoo/dev
mount --rbind /run      /mnt/gentoo/run
mount --make-slave      /mnt/gentoo/run

# 2. copy DNS resolution
cp --dereference /etc/resolv.conf \
   /mnt/gentoo/etc/resolv.conf

# 3. enter chroot
chroot /mnt/gentoo /bin/bash
```

Why each step is needed

/proc — process info filesystem

The kernel exposes running processes and kernel parameters via /proc. Without it, ps, top, kill and most system tools break inside the chroot.

/sys — kernel device tree

Hardware info, driver parameters, udev events. Package installs that touch hardware config (e.g. udev rules) need /sys to be visible.

--make-rslave — propagation isolation

Changes to mounts inside the chroot must NOT propagate back to the host. rslave = chroot sees host mounts, but host never sees chroot mounts.

/dev — device nodes

Without /dev the chroot has no /dev/null, no /dev/random, no disk devices. Compilers, package managers and scripts all need /dev.

--dereference resolv.conf

On systemd-resolved hosts resolv.conf is a symlink → dangling inside chroot. --dereference copies the real file so DNS works for package downloads.

The chroot shares the HOST kernel — /proc, /sys, /dev are kernel interfaces, not files. They must be bind-mounted from the host because there is no second kernel inside the chroot.

04

make.conf

The heart of Gentoo — where Arch ends and Gentoo begins

Compiler Flags — Tuning for Your CPU

Detect your architecture: `gcc -march=native -Q --help=target | grep "\-march"`

Platform	−march=	Notes	CPU_FLAGS_X86
QEMU default	k8-sse3	AMD K8 era (Athlon 64). QEMU without -cpu host.	mmx mmxext sse sse2 sse3
AMD Ryzen 5800X	znver3	Zen 3 desktop. AVX2, FMA, SHA-NI, AES.	mmx mmxext pclmul popcnt rdrand sha sse sse2 sse3 sse4_1 sse4_2 sse4a ssse3 avx avx2 f16c fma
AMD TR PRO 5995WX	znver3	Zen 3 workstation (64c/128t). Same -march as 5800X!	mmx mmxext pclmul popcnt rdrand sha sse sse2 sse3 sse4_1 sse4_2 sse4a ssse3 avx avx2 f16c fma
Intel Core 12th gen	alderlake	AVX-512 available (AMD Zen3 does NOT have it).	mmx mmxext pclmul popcnt sha sse sse2 sse3 sse4_1 sse4_2 ssse3 avx avx2 avx512f f16c fma
ARM Cortex-A72 (RPi4)	cortex-a72	aarch64 — different make.conf entirely.	edsp neon thumb vfp vfpv3 vfpv4 aarch64

MAKEOPTS formula: -j(threads+1) | 5800X → -j17 | 5995WX → -j65 | QEMU 4c → -j5

make.conf — Advanced Options Reference

Localisation

```
LANG="pl_PL.UTF-8"
LC_MESSAGES="C"  # English error messages → easier to Google!
L10N="pl en"
LINGUAS="pl en"
```

CHOST / Keywords

```
CHOST="x86_64-pc-linux-gnu"
ACCEPT_KEYWORDS="~amd64"  # testing branch
```

Features

```
FEATURES="buildpkg parallel-fetch parallel-install"
BINPKG_COMPRESS="zstd"  # compress local binary pkgs
```

GPU & Input

```
VIDEO_CARDS="amdgpu radeonsi"  # AMD dGPU
VIDEO_CARDS="intel iris"  # Intel iGPU
VIDEO_CARDS="qxl virtio"  # QEMU/VM
INPUT_DEVICES="libinput synaptics"
```

Mirrors & Binary

```
GENTOO_MIRRORS="http://ftp.vectranet.pl/gentoo/
http://gentoo.prz.rzeszow.pl/"
PORTAGE_BINHOST="https://distfiles.gentoo.org/releases/amd64/binpack
ages/23.0/x86-64/"
```

LLVM / Bootloader

```
LLVM_TARGETS="X86"
CLANG_TARGETS="X86"
GRUB_PLATFORMS="efi-64"  # or: pc, qemu, xen
```

05

USE Flags & Portage

The unique power of source-based package management

USE Flags — Feature Switches for Every Package

USE flags are global or per-package switches that control which features get compiled in. No other major distro has this.

1. Profile defaults

```
/etc/portage/make.profile
```

Inherited from eselect profile. ~20–60 flags depending on profile.

2. Global (make.conf)

```
/etc/portage/make.conf
USE="-systemd -pulseaudio openrc alsa wifi X gtk"
```

Applies to every package on the system. Use sparingly on desktops.

3. Per-package

```
/etc/portage/package.use  (single file)
/etc/portage/package.use/  (directory — preferred for large systems)
```

Most precise control. Recommended for desktops with 500+ packages.

USE Flags in Practice — parted Example

```
# Check available flags for a package
equery uses sys-block/parted

# Add flags per-package
echo "sys-block/parted device-mapper readline" >> /etc/portage/package.use/rescue-tools
echo "app-admin/testdisk ntfs reiserfs" >> /etc/portage/package.use/rescue-tools

# Or use flaggie (modern tool)
flaggie sys-block/parted +device-mapper +readline
flaggie -g +X +alsa     # -g = global in make.conf
```

Resolving USE flag conflicts — the iterative process:

1. emerge -pv <package> → find circul deps
2. Add fix to /etc/portage/package.use/
3. Repeat until emerge -pv shows no errors
4. emerge <package>

Live example: resolving 185 packages → 5 packages for GRUB by removing 'gtk' from polkit, 'sdl' from GRUB, 'webp' from tiff, 'truetype' from pillow.

Server: global USE flags OK (30 pkgs). Desktop: always use package.use (500–2000+ pkgs — global flags cause cascading dependency explosions).

Portage vs pacman — Command Reference

Action	Arch (pacman)	Gentoo (emerge/equery)
Sync package tree	`pacman -Sy`	`emerge-webrsync` / `emerge --sync`
Install package	`pacman -S pkg`	`emerge pkg` / `emerge -a pkg`
Remove package	`pacman -R pkg`	`emerge --depclean pkg` / `emerge -C pkg`
Full system update	`pacman -Syu`	`emerge -uDN @world` / `emerge --update --deep --newuse @world`
Search by name	`pacman -Ss name`	`emerge -s name` / `eix name`
Search in descriptions	`pacman -Ss desc`	`emerge -S desc` / `eix -S desc`
Show package info	`pacman -Si pkg`	`emerge -pv pkg` / `equery meta pkg`
List installed files	`pacman -Ql pkg`	`equery files pkg`
Which pkg owns file	`pacman -Qo /path`	`equery belongs /path` / `e-file /path`
Pretend install	`pacman -Sp pkg`	`emerge -pv pkg`
Binary packages	`pacman -U pkg.pkg.tar`	`emerge --getbinpkg pkg`
Clean old sources	`paccache -r`	`eclean-dist` / `eclean-pkg`

eselect — System Configuration Switcher

Module	Purpose	Example
profile	☆ Choose system profile (desktop/server/hardened)	`eselect profile list / set 3`
kernel	☆ Manage /usr/src/linux symlink	`eselect kernel list / set 1`
locale	System language (LANG)	`eselect locale list / set 4`
gcc	Switch active GCC version	`eselect gcc list / set 2`
editor	Default EDITOR variable	`eselect editor list / set nano`
rc	Manage OpenRC runlevel scripts	`eselect rc add sshd default`
news	☆ Read Gentoo project news (read before updates!)	`eselect news read`
opengl	Switch OpenGL provider (NVIDIA/AMD/Mesa)	`eselect opengl set nvidia`
repository	Manage overlays (install separately)	`eselect repository add guru`
php / python	Switch active language version	`eselect php set cli php8.2`

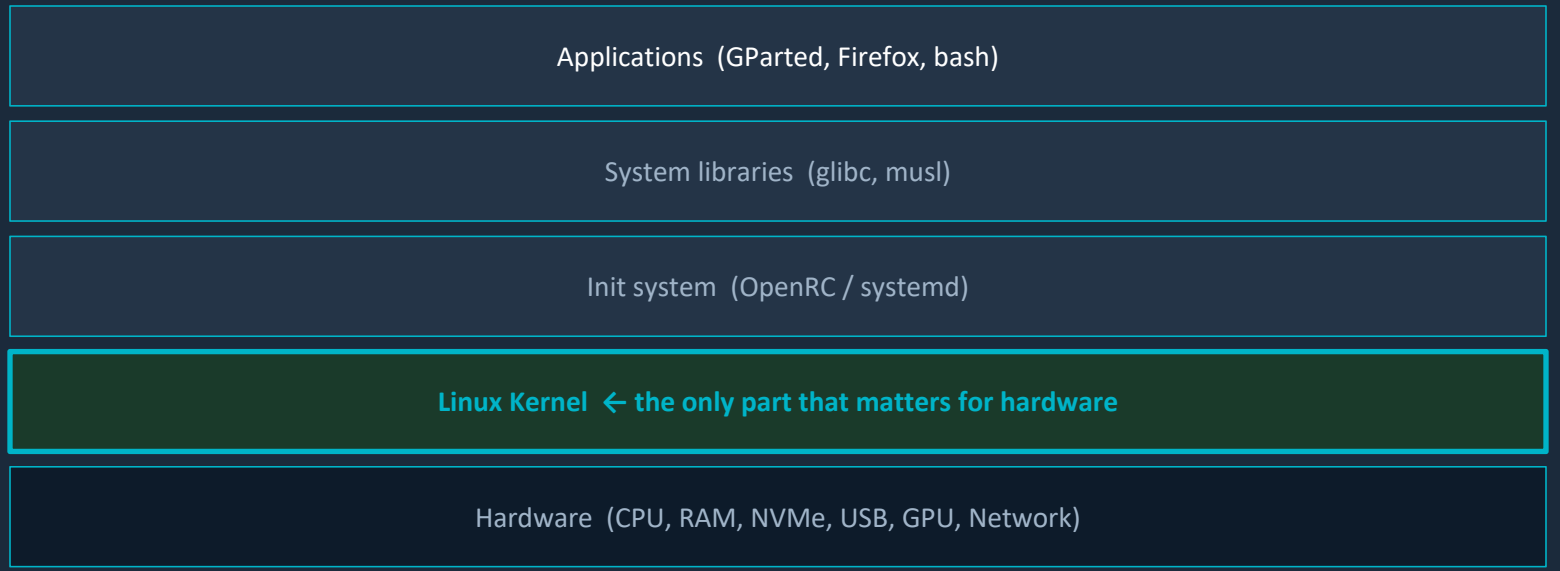
06

The Kernel

The only program that talks to hardware

The Secret: The Distribution Doesn't Matter

The kernel is the ONLY program that communicates with hardware. Everything above it — Gentoo, Arch, Ubuntu, Android — is just userspace running ON TOP of the kernel.



Master kernel configuration → you are independent of any distribution.
The same kernel config boots Gentoo, Arch, Ubuntu, or a custom minimal system.

Three Approaches to the Kernel

Method 1

Copy from Arch ISO

5 minutes

- ✗ No optimisation
- ✗ Arch initramfs hooks (archiso!) won't boot Gentoo
- ✓ Good for testing system is bootable
- ✓ No compilation needed

Must generate NEW initramfs with dracut!

Method 2

Compile with /proc/config.gz

1–2 hours

- ✓ Uses proven working config
- ✓ Adds Gentoo optimisation flags
- ✓ Correct initramfs
- ☆ Best balance for learning

```
zcat /proc/config.gz >
/usr/src/linux/.config
make olddefconfig && make -j5
```

Method 3

Full make menuconfig

3–8 hours

- ✓ Maximum optimisation
- ✓ Only drivers YOU need
- ✓ Smallest possible kernel
- ✗ Requires deep hardware knowledge

Distribution kernels include ALL drivers → hours to compile. Custom kernel = minutes.

SECTION 6 · The Kernel

Driver Dependency Chains — Why Kernel Config is Critical

Boot from USB

- xhci_hcd / ehci_hcd (USB controller)
- ↓
- usb_storage / uas (mass storage)
- ↓
- GPT partition table support
- ↓
- vfat (EFI /boot partition)
- ↓
- ext4 / xfs / btrfs (root filesystem)
- ↓
- rootfs mounted ✓ system boots

Router: LAN + 2x WiFi

eth0 (onboard)	PCIe bus → e1000e / r8169 driver → NET_CORE
WLAN1 (PCIe)	PCIe → ath10k_pci / iwlwifi → mac80211 → cfg80211
WLAN2 (USB)	xhci_hcd → usb_storage → rt2800usb → mac80211
All interfaces	bridge / nftables / ip_forward (routing)

Router works ✓

Missing ONE link in the chain = kernel panic / device not found. lspci, lsusb, dmesg are your best friends.

07

GRUB & Bootloaders

BIOS/MBR · GPT · UEFI · Secure Boot

Boot Modes — Four Combinations

BIOS + MBR

Legacy

```
grub-install --target=i386-pc /dev/vda
```

Oldest method. GRUB in first 512 bytes (MBR). No special partition needed.

BIOS + GPT

⚠ Special case

```
grub-install --target=i386-pc --force /dev/vda
# OR create 1MB BIOS Boot Partition (type 0xEF02)
```

GPT leaves no space before partition 1 for GRUB stage1. Need BIOS Boot Partition or --force in VMs.

UEFI + GPT

☆ Modern standard

```
grub-install --target=x86_64-efi \
  --efi-directory=/boot \
  --bootloader-id=gentoo
```

Required for Secure Boot. All hardware post-2012. FAT32 EFI partition required.

UEFI + Removable

☆ Our choice (USB)

```
grub-install --target=x86_64-efi \
  --efi-directory=/boot \
  --no-nvram --removable
```

Installs to EFI/BOOT/BOOTX64.EFI — works on ANY UEFI machine without writing to NVRAM.

GRUB Platforms & USE Flags — Dependency Chains

GRUB_PLATFORMS in make.conf:

Platform	Use case
pc	BIOS/MBR boot
efi-64	UEFI 64-bit (most common)
efi-32	UEFI 32-bit (old tablets)
qemu	QEMU/KVM virtual machines
xen	Xen hypervisor
ieee1275	PowerPC (old Macs)

The USE flag dependency trap:

```
polkit USE="gtk"  (desktop profile default)

→ pulls gtk+ → needs Mesa → needs LLVM

→ LLVM pulls Rust, Clang, compiler-rt...

Result: 5 packages → 185 packages!


Fix: echo "sys-auth/polkit -gtk" >> package.use

     echo "media-libs/tiff -webp" >> package.use

     echo "dev-python/pillow -truetype" >> package.use

Result: back to 5 packages ✅
```

08

System Configuration

OpenRC · Users · Locale · Binary packages

OpenRC vs systemd — and Service Management

Action	Arch (systemd)	Gentoo (OpenRC)
Start service	`systemctl start sshd`	`rc-service sshd start`
Stop service	`systemctl stop sshd`	`rc-service sshd stop`
Enable at boot	`systemctl enable sshd`	`rc-update add sshd default`
Disable at boot	`systemctl disable sshd`	`rc-update del sshd default`
Status	`systemctl status sshd`	`rc-service sshd status`
List enabled	`systemctl list-unit-files`	`rc-update show`
Runlevel equiv.	`systemctl isolate multi-user`	`openrc default`
Display manager	`systemctl enable lightdm`	`rc-update add display-manager default` `+ set DISPLAYMANAGER="lightdm" in /etc/conf.d/display-manager`

OpenRC runlevels: sysinit → boot → default → shutdown | Services live in /etc/init.d/ | Config in /etc/conf.d/

User Setup — Groups, UIDs and Common Pitfalls

```
# Create required groups FIRST (some don't exist in base Gentoo)
groupadd -g 1000 rescue
groupadd netdev
groupadd plugdev

# Create user with correct UID/GID
useradd -m -u 1000 -g 1000 -G wheel,audio,video,plugdev,netdev,usb -s /bin/bash rescue
passwd rescue

# Passwordless sudo (rescue system — convenience over security)
mkdir -p /etc/sudoers.d
echo "rescue ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers.d/rescue
chmod 440 /etc/sudoers.d/rescue    # sudo requires 440 — will refuse 644!

# Verify
id rescue   # Should show uid=1000(rescue) gid=1000(rescue)
```

⚠ **Pitfalls encountered:**

- netdev and plugdev groups do not exist in base Gentoo — create them before useradd
- useradd -g 1000 fails if group 1000 doesn't exist yet — create group first
- /etc/sudoers.d/ directory may not exist — mkdir -p before writing
- chmod 440 is mandatory — sudo silently refuses files with wrong permissions
- uid ≠ gid when groups created before user consume GIDs — create group rescue -g 1000 explicitly

Binary Packages — When to Compile, When to Download

Package	Compile time	Binary time	Recommendation
LLVM	~45 min	~2 min	Use binary always in workshops
Rust	~30 min	~3 min	Use binary — rarely need custom flags
Mesa	~20 min	~2 min	Use binary unless custom VIDEO_CARDS
GCC	~15 min	~1 min	Source recommended for full optimisation
Firefox	~60 min	~5 min	Use firefox-bin (official Mozilla binary)
Chromium	~90 min	~5 min	Source only — no official Chromium binary
Linux kernel	~10 min	N/A	Always compile (or copy from host)
GRUB	~2 min	~30 sec	Either — small enough to compile

```
# Enable official Gentoo binhost
getuto    # Import Gentoo GPG keys (required for binary verification)
echo 'PORTAGE_BINHOST="https://distfiles.gentoo.org/releases/amd64/binpackages/23.0/x86-64/"' >> /etc/portage/make.conf
echo 'FEATURES="${FEATURES} getbinpkg"' >> /etc/portage/make.conf
echo 'BINPKG_RESPECT_USE="y"' >> /etc/portage/make.conf
```

⚠ *Risk: official binaries built with default profile USE flags. BINPKG_RESPECT_USE=y rejects incompatible ones. Binary packages compiled for generic x86-64 — you lose CPU-specific optimisations.*

Cross-Compilation — Build for ARM on x86

Compile on your fast x86_64 desktop → deploy on slow ARM device (RPi5, router, embedded).

Platform	CHOST triplet
x86_64 PC	x86_64-pc-linux-gnu
ARM64 RPi5	aarch64-unknown-linux-gnu
ARM32 RPi3	armv7a-unknown-linux-gnueabihf
RISC-V 64	riscv64-unknown-linux-gnu
x86 32-bit	i686-pc-linux-gnu

make.conf for RPi5 (ARM64):

```
CHOST="aarch64-unknown-linux-gnu"
COMMON_FLAGS="-O2 -pipe \
  -march=armv8-a+crc+crypto \
  -mtune=cortex-a76"
CPU_FLAGS_ARM="edsp neon thumb \
  vfp vfpv3 vfpv4 aarch64 sha1 sha2 aes"
```

```
# Install crossdev toolchain
emerge sys-devel/crossdev
crossdev -t aarch64-unknown-linux-gnu

# Cross-emerge a package for ARM64
aarch64-unknown-linux-gnu-emerge --ask www-client/lynx

# Alternative: QEMU user emulation (slower but simpler)
emerge app-emulation/qemu
```

09

The Rescue System

What we built · Writing to USB · Next steps

What We Built — Complete Tool Stack

Tool	Package	Purpose
GParted	sys-block/gparted	GUI partition editor — resize, create, delete, move
TestDisk + PhotoRec	app-admin/testdisk	Partition recovery + file carving from damaged disks
cryptsetup	sys-fs/cryptsetup	Open/close LUKS encrypted volumes
ntfs3g	sys-fs/ntfs3g	Read/write NTFS (Windows drives)
exfatprogs	sys-fs/exfatprogs	exFAT support (SD cards, large USB drives)
NetworkManager	net-misc/networkmanager	WiFi + Ethernet management with GUI
wpa_supplicant	net-wireless/wpa_supplicant	WPA/WPA2/WPA3 authentication backend
Firefox	www-client/firefox-bin	Browser for documentation lookup
LVM2	sys-fs/lvm2	Logical Volume Manager — with thin, nvme, xfs support
XFCE + LightDM	xfce-base/xfce4-meta	Lightweight desktop environment
OpenSSH	net-misc/openssh	Remote access to the rescue system
dracut	sys-kernel/dracut	Generate initramfs for the Arch kernel

Writing the Image to a Real USB Drive

```
# Step 1: Find your USB drive — ALWAYS verify before dd!
lsblk
# Look for your USB drive size — e.g. /dev/sdb (NOT your system disk!)

# Step 2: Write the QEMU image to USB
dd if=gentoo-rescue.img of=/dev/sdX bs=4M status=progress conv=fsync

# Alternative with pv (prettier progress bar)
pv gentoo-rescue.img | dd of=/dev/sdX bs=4M conv=fsync

# Step 3: Ensure all data is written before removing
sync

# Verify: check if the partition table looks correct
fdisk -l /dev/sdX
```

⚠ dd = 'disk destroyer' if you target the wrong device. Double-check /dev/sdX with lsblk. There is no undo.

What We Covered Today

Gentoo philosophy

Source-based, CPU-optimised, fine-grained control vs. binary distros

Stage3 & profiles

First architectural decision — init system, USE defaults, profile purpose

make.conf

CPU flags, MAKEOPTS, USE flags, GPU drivers, mirrors, CHOST

Portage & USE flags

emerge, equery, flaggie, per-package flags, dependency resolution

eselect

Profile, kernel, locale, gcc, rc, news — the Gentoo config hub

Kernel & initramfs

Distribution = userspace only; kernel is hardware-independent; dracut for initramfs

GRUB

BIOS/UEFI/GPT combinations; --removable for portable USB

OpenRC

Service management, runlevels, display-manager config

Binary packages

Binhost, getbinpkg, BINPKG_RESPECT_USE, compilation time reality

Cross-compilation

CHOST triplet, crossdev, ARM64 for RPi5

Next Steps & Resources

Gentoo Handbook

`https://wiki.gentoo.org/wiki/Handbook:AMD64`

The definitive installation and configuration guide

Gentoo Wiki

`https://wiki.gentoo.org`

Package-specific guides, USE flag explanations

Portage man page

`man portage  /  man emerge`

Complete reference for all emerge options

Kernel config (next session)

`make menuconfig deep-dive`

Dedicated presentation on kernel configuration