VOID Linux Installation mit BtrFS
(basierend auf der Anleitung von Jake@Linux)

Schritt 1 – erstelle das Installations-Medium

Benutze die Void-base-Installation (void-live-x86_64-20250202-base.iso von der Website https://voidlinux.org/download/), schreibe diese iso-Datei bootfähig auf einen USB-Stick mit Hilfe von balena-etcher, Rufus, Ventoi oder ähnlichen Programmen.

Schritt 2 – Erkenne das Ziel-Laufwerk für die Installation

Ich werde als Ziel-Laufwerk die Bezeichnung /dev/sdX benutzen. Auf modernen Computern kann es auch /dev/nvmeYn1, wobei X für einen Buchstaben (a,b, ...) und bei Y für eine Ziffer (0,1, ...) steht. Du findest das korrekte Laufwerk mit dem Linux-Kommando „lsblk“ heraus.

Schritt 3 – Partitionierung

Nach dem Start der live-Disk vom USB-Stick meldest du dich mit dem User ***anon*** und dem Passwort ***voidlinux*** am root-Prompt an. Damit hast du alle Rechte, jedes Kommando auszuführen (sichtbar durch das voranstehende #).
Der Einfachheit halber benutze ich das Programm *cfdisk,* alternativ kannst du auch *fdisk, parted* oder andere benutzen. *cfdisk* eignet sich ausschließlich für GPT-Partitionstabellen, für MSDOS (Lagacy boot) musst du ein anderes Programm, wie fdisk wählen. Ich beschreibe im Folgenden ausschließlich eine Installation für den (U)EFI-Boot:

bash —> wechselt in die bash-Shell, die einige Vorteile gegenüber der Standard-Shell sh hat (ist optional)

cfdisk /dev/sdX

Mit dem Kommando [Löschen] kannst du eventuell vorhandene Partitionen löschen (Achtung, Datenverlust!). Mit [Neue] legst du im freien Bereich eine neue Partition an.
Wir brauchen eine EFI-Partition (350 MiB genügen), eine Swap-Partition für Systeme mit 8GiB RAM oder weniger (4 oder 8GiB) und die Systempartition mit dem gesamten Rest des Speicherplatzes.

Drücke auf [Neue] und gib am Prompt „Partitionsgröße:“ *350M* ein. Wechsle dann mit der Taste <Pfeil nach unten> auf den freien Bereich und drücke wieder

[Neue], gib am Prompt „Partitionsgröße:“ *4G* ein. Wechsle erneut mit der Pfeiltaste auf den noch freien Bereich und bestätige den Wert mit *<ENTER>.*
Nun müssen noch die Partitionstypen festgelegt werden. Wechsle also mit der Pfeiltaste nach oben auf den 1. Eintrag (1. Partition), danach mit der Pfeiltaste nach rechts auf [Typ], drücke *<ENTER>* und wähle den obersten Eintrag ***EFI-System.***
Wechsle nun auf den 2. Eintrag, dann auf [Typ] und wähle ***Linux Swap*** aus.
Die letzte Partition hat bereits den korrekten Typ ***Linux-Dateisystem***.
Wechle also auf [Schreiben] und drücke *<ENTER>.* Nun muss die Einteilung der Festplatte/SSD mit „yes“ (ausgeschrieben) und <ENTER> bestätigt werden.
Danach kannst du das Programm mit [Ende] beenden.

Schritt 4 – Formatierung der Partitionen

```
# mkfs.btrfs –L VOID /dev/sdX3      —> erzeugt das Dateisystem auf der System-Partition
# mkswap –L SWAP /dev/sdX2          —> erzeugt die Swap-Partition
# mkfs.fat –F 32 –n ESP /dev/sdX1   —> erzeugt die EFI-Systempartition
```

Schritt 5 – Erzeuge die BtrFS-Subvolumes

Um die nötigen Boot-Optionen in einer Variablen zu speichern geben wir ein:

```
# BOPS=“rw,noatime,compress=zstd:3,discard=async,space_cache=v2“
```

Jetzt können wir das Top-Level Subvolume mounten:

```
# mount –o $BOPS /dev/sdX3 /mnt  —> die neue Root-Partition wird in /mnt eingehängt
# btrfs su cr /mnt/@             —> erzeugt das root-Subvolume
# btrfs su cr /mnt/@home         —> erzeugt das home-Subvolume
# btrfs su cr /mnt/@snapshots    —> erzeugt das Subvolume für Snapshots
```

Jetzt muss die Root-Partition vorübergehend wieder ausgehängt werden, um danach die Mount-Optionen nutzen zu können.

```
# umount /mnt
# mount –o $BOPS,subvol=@ /dev/sdX3 /mnt
```

Um für die übrigen Subvolumes eine „Heimat“ zu schaffen, müssen zuerst noch einige Mountpoints geschaffen werden:

```
# mkdir –p /mnt/{boot/efi,home,.snapshots}
# swapon /dev/sdX2                                —> schaltet den Swap aktiv
# mount –o rw,noatime /dev/sdX1 /mnt/boot/efi
# mount –o $BOPS,subvol=@home /dev/sdX3 /mnt/home
# mount –o $BOPS,subvol=@snapshots /dev/sdX3 /mnt/.snapshots
```

Eine Schlüsselfunktion und großer Vorteil von BtrFS sind Snapshots. Nun gibt es einige Verzeichnisse im Linux-Dateisystem, die nicht gesichert werden müssen, deshalb erstellen wir ein paar Subvolumes, die von den Snapshots ausgenommen werden:

```
# mkdir –p /mnt/var/cache
# btrfs su cr /mnt/var/cache/xbps
# btrfs su cr /mnt/var/tmp
# btrfs su cr /mnt/srv
```

Überprüfe, ob alles korrekt eingehängt wurde:

```
# lsblk oder    # df -h
```

Schritt 6 - Installation

Es ist eine gute Idee, einen Software-Mirror (Server mit dem Repository von Void) in der Nähe des eigenen Standortes zu suchen, bevor man installiert, weil die Dauer der Installation maßgeblich von der Download-Geschwindigkeit des gewählten Mirrors abhängt. Die mögliche URL erfährt man auf der Website https://xmirror.voidlinux.org
Für Deutschland kommt https://repo-de.voidlinux.org/ infrage.
Ich gehe von der Architektur ***x86_64*** aus, die vermutlich für die meisten User infrage kommt? Außerdem installiere ich hier mit der glibc library, zu der es auch die Alternative musl gäbe (siehe Dokumentation auf https://voidlinux.org).

Wir erstellen uns also zunächst 2 Variablen, um uns die Installation zu erleichtern:
REPO=https://repo-de.voidlinux.org/current
ARCH=x86_64

Um die Integrität der Pakete zu verifizieren, müssen wir die RSA-Keys vom Installationsmedium auf die eigene Festplatte übertragen. Dazu erstellen wir das nötige Verzeichnis:
mkdir –p /mnt/var/db/xbps/keys und kopieren die Schlüssel
cp /var/db/xbps/keys/* /mnt/var/db/xbps/keys/

Jetzt kann das Basissystem und einige zusätzliche Tools installiert werden. Als Editoren installiere ich vim, micro und nano. Wer den aktuellsten Kernel möchte (neueste Hardware) nimmt linux-mainline, ich installiere den stabilen Kernel linux.

XBPS_ARCH=$ARCH xbps-install –S –R „$REPO“ –r /mnt base-system base-devel bash-completion linux btrfs-progs cryptsetup vim nano micro

Schritt 7 - Konfiguration

Jetzt können wir in das neue Dateisystem wechseln, um abschließende Arbeiten vor einem Neustart vorzunehmen. Dazu muss zunächst nach Jake@Linux ein Pseudo-Dateisystem erstellt werden, das chroot angeblich braucht (siehe englische Anleitung von Jake@Linux, ich nehme an, das ist veraltet?). Es kann auch gleich der Befehl xchroot benutzt werden:
xchroot /mnt /bin/bash
Jetzt wird ein wenig konfiguriert, nutze vim, nano oder meinen Favoriten micro:
micro /etc/rc.conf —> entferne den Hashtag vor Zeilen, die du aktivieren willst und ändere
(etwa HOSTNAME, TIMEZONE, HARDWARECLOCK oder KEYMAP)

ln –s /usr/share/zoneinfo/Europe/Berlin /etc/localtime (für Deutschland)
Jetzt muss der richtige Zeichensatz aktiviert werden:, für Deutschland ist das de_DE.UTF-8. Dazu öffnest du die Datei
micro /etc/default/libc-locales —> entferne # vor de_DE.UTF-8 und speichere die Datei
Jetzt wird rekonfiguriert:
xbps-reconfigure –f glibc-locales
und der Hostname deiner Wahl wird vergeben:
echo „Void“ > /etc/hostname

Jetzt die Datei hosts anpassen:
cat <<EOF > /etc/hosts
#
/etc/hosts: statische Tabelle für Hostnamen
#
127.0.0.1 localhost
::1 localhost
127.0.1.1 Void.localdomain Void
EOF

Schritt 8 - Usermanagement

passwd —> vergibt das Passwort für den User ***root***
Jetzt legen wir den eigenen User an (zum Beispiel „paulchen“, nur Kleinbuchstaben)
useradd paulchen
passwd **paulchen** —> vergibt das Passwort für Paulchen
usermod –aG wheel **paulchen** —> Paulchen bekommt root-Rechte

Um die Root-Rechte freizuschalten, wechseln wir mit dem root-User in die Bash-Shell und editieren die Datei *visudo*
chsh –s /bin/bash root
EDITOR=micro visudo
Suche die Zeile mit dem Eintrag „# %wheel ALL=(ALL:ALL) ALL“ und entferne das #-Zeichen.
Optional kannst du auch die Zeile „paulchen ALL=(ALL:ALL) ALL“ anfügen. Speichere die Datei.

Schritt 9 – Repos

xbps-install –S —> synchronisiert die Repositories
Die folgenden Schritte sind optional, ich empfehle sie:
xbps-install void-repo-nonfree
xbps-install -S
xbps-install void-repo-multilib
xbps-install -S

Schritt 10 – fstab kreieren

# ED=$(blkid –s UUID –o value /dev/sdX1)
# RD=$(blkid –s UUID –o value /dev/sdX3)
# cat <<EOF > /etc/fstab
UUID=$RD / btrfs $BOPS,subvol=@ 0 1
UUID=$RD /home btrfs $BOPS,subvol=@home 0 2
UUID=$RD /.snapshot btrfs $BOPS,subvol=@snapshots 0 2
UUID=$ED /boot/efi vfat defaults,noatime 0 2
tmpfs /tmp tmpfs defaults,nosuid,nodev 0 0
EOF

Schritt 11 – Bootloader installieren und konfigurieren

GRUB für EFI installieren:
xbps-install grub-x86_64-efi
grub-install --target=x86_64-efi --efi-directory=/boot/efi --bootloader-id=VOID

Schritt 12 – Installation von Software, Programmen und Tools

xbps-install duf git wget curl feh rsync openvpn bat eza btrfs-progs btrfs-assistant grub-btrfs ntfs-3g xkill time alacritty flameshot okular timeshift variety vlc grsync sweeper firefox (und was dir sonst noch so brauchbar erscheint)

Schritt 13 – Link-Services

Mit dem Kommando ***ip a*** kannst du ermitteln, wie deine aktive Netzwerkschnittstelle heißt, etwa ***enp4s0*** (in meinem Fall).
Wenn sie also **NICHT** eth0 heißt, muss das angepasst werden:
cp /etc/sv/dhcpcd-**etho** /etc/sv/dhcpcd-**enp4s0**
sed –i ´s/eth0/enp4s0/g´ /etc/sv/dhcpcd-enp4s0/run
Danach die Links setzen:
ln –s /etc/sv/dhcpcd-enp4s0 /var/service
ln –s /etc/sv/dhcpcd /var/service
xbps-install NetworkManager
ln –s /etc/sv/NetworkManager /var/service
Zum Abschluss noch mal alles rekonfigurieren:
xbps-reconfigure –fa

Schritt 14 – Exit und Reboot

Unsere Installation des Basissystems ist abgeschlossen. Wir verlassen die Chroot-Umgebung und starten das neue Betriebssystem das erste Mal:
exit
reboot