Gabriel Sanches's Handbook

Gabriel Sanches's Handbook

This handbook

About

This book is a compilation of tech-related stuff I consider important and somewhat hard to keep in memory for ever. Also, part of this knowledge comes from trial and error, therefore it might be useful for some folks around looking for specific solutions.

How

This book is built with mdBook. You can check the source code here.

Void Linux

Void is a stable rolling release, systemd-free Linux distribution with a powerful package manager called XBPS. I use as the Linux distro for my main programming workstation. You can check more information about Void on the official website.

Why

The first thing some people ask is why use Void and not Ubuntu (for its supposed simplicity), or even Arch (for rolling release).

Well, I use Void because I value *minimalism* and *control* over as much features of my machine it is possible. Ubuntu is far from being minimalist and Arch abstracts too many functionalities to the systemd suite.

This chapter

Having my reasonings behind picking Void being clarified, take note that configuring and maintaining it requires some effort, but at the end of the day it totally pays off because you end up learning many aspects of the system and how Linux and other useful tools work in general.

Since such knowledge doesn't stick to the brain forever, this chapter is an ever-changing compilation about everything I struggled through regarding Void, in order to help people that are interested in using it.

Installing Void

In this guide you will learn how to get a working installation of Void containing:

- btrfs with Zstandard compression
- LUKS-encrypted root and swapfile
- GRUB with UEFI

Things you will *not* find in this handbook:

- Instructions for file systems other than btrfs
- Full disk encryption (there's an official guide here)
- Reasoning for all choices I've made

> **Note:** Sometimes I don't know the true reason for all my choices. 😬

Live ISO

For this tutorial, you'll need to download the *base* image from Void's official download page, which can be found here. You'll have to choose between the *glibc* one and the *musl* one.

One question people make often is why the live ISO image is "not up-to-date". Void is a *rolling release* distro. It's different from Ubuntu, which has release cycles. Software in Void is constantly getting updated, so regardless of how old the image is, you'll end up downloading and installing the most recent packages there exists in the Void repos.

> **Note:** A possible issue in old live ISO images is lack of support for most recent hardware. This is uncommon to happen, but if it does, resort to official channels for help.

Logging in

There are two available users, `root` (superuser) and `anon`. The password of both is `voidlinux` (it might vary from image to image, but it's always explict in the starting screen). I prefer to log in as superuser so I don't have to type `sudo` all the time. Also, I highly recommend that you run `exec bash` since it is a little more comfortable to work with than `dash`, the default shell for the live ISO image.

Configuring keyboard layout

If you need a different layout other than `en_US` during the install process, you can change it using `loadkeys`, for example:

```
# loadkeys $(ls /usr/share/kbd/keymaps/i386/**/*.map.gz | grep br-abnt2)
```

Connecting to the internet

If you're connect via ethernet, you're good to go, since the live ISO image already runs the DHCP service by default. However, if you're on Wi-Fi, you'll need additional setup. For example, to configure it for interface `wlan0`:

```
# cp /etc/wpa_supplicant/wpa_supplicant.conf /etc/wpa_supplicant/wpa_supplicant-wl
# wpa_passphrase $SSID $PASS >> /etc/wpa_supplicant/wpa_supplicant-wlan0.conf
# ip link set up wlan0
# sv restart dhcpcd
```

Formatting disks

Partitioning

The minimum number of partitions is three:

- The EFI partition, where the GRUB bootloader resides (`/efi`)
- The boot partition, where kernels are stored (`/boot`)
- The LUKS-encrypted btrfs root partition

First, we need to generate the partition tables. Let's set the disk up:

```
# fdisk /dev/sda
```

> **Note:** Check which device is the one you want to install Void into. For this guide I'll simply use `/dev/sda`, but it can change depending on your setup, so *watch out!*

After running `fdisk`, it will prompt you with a menu, so follow these steps:

1. Press `g` and then `Enter` to generate a GPT table
2. Press `n` and then `Enter` to create the EFI partition with size of 200 MiB (`+200M`)
3. After creating the partition, change its type by pressing `t` and then `Enter` and select the option that represents the EFI partition (usually 1)
4. Press `n` and then `Enter` to create the boot partition with size of at least 500 MiB (`+500M`)
5. If the default partition type selected for the root partition is not already `Linux Filesystem`, change it by pressing `t` and then selecting it (usually 20)

Creating the file systems

EFI partition

```
# mkfs.vfat -nBOOT -F32 /dev/sda1
```

Boot partition

```
# mkfs.ext2 -L grub /dev/sda2
```

LUKS-encrypted root partition

```
# cryptsetup luksFormat --type=luks -s=512 /dev/sda3
# cryptsetup open /dev/sda3 cryptroot
# mkfs.btrfs -L void /dev/mapper/cryptroot
```

Mounting

Now that the file systems have been created, it's time to mount them.

Root partition

For the root partition, it is necessary to first create *top-level subvolumes*:

```
# BTRFS_OPTS="rw,noatime,ssd,compress=zstd,space_cache,commit=120"
# mount -o $BTRFS_OPTS /dev/mapper/cryptroot /mnt
# btrfs subvolume create /mnt/@
# btrfs subvolume create /mnt/@home
# btrfs subvolume create /mnt/@snapshots
# umount /mnt
```

Then, proceed to mount them:

```
# mount -o $BTRFS_OPTS,subvol=@ /dev/mapper/cryptroot /mnt
# mkdir /mnt/home && mount -o $BTRFS_OPTS,subvol=@home /dev/mapper/cryptroot /mnt/
# mkdir /mnt/.snapshots && mount -o $BTRFS_OPTS,subvol=@snapshots /dev/mapper/cryp
```

> **Note:** Those are the options and names I like to use for btrfs. Feel free to change them according to your needs!

Extra subvolumes

Since btrfs doesn't take snapshots of nested subvolumes, it is a good practice to create some extra, nested subvolumes for directories that are not meant to be in the snapshots:

```
# mkdir -p /mnt/var/cache
# btrfs subvolume create /mnt/var/cache/xbps
# btrfs subvolume create /mnt/var/tmp
# btrfs subvolume create /mnt/srv
```

Also, if you're going for the swapfile, don't forget to create a nested subvolume for it:

```
# btrfs subvolume create /mnt/var/swap
```

> **Note:** Nested subvolumes are mounted automatically when their parent subvolume is mounted.

EFI and boot partitions

Once the root partition is ready, it is time to mount the remaining ones:

```
# mkdir /mnt/efi && mount -o rw,noatime /dev/sda1 /mnt/efi
# mkdir /mnt/boot && mount -o rw,noatime /dev/sda2 /mnt/boot
```

Installing the system

Base installation

First, set some helper variables:

```
# REPO=https://alpha.us.repo.voidlinux.org/current
# ARCH=x86_64
```

If going for the *musl* variant, just reset the variables accordingly:

```
# REPO="$REPO/musl"
# ARCH="$ARCH-musl"
```

Then proceed to install the base system:

```
# XBPS_ARCH=$ARCH xbps-install -S -R "$REPO" -r /mnt base-system btrfs-progs crypt
```

> **Note:** The command above installs the base system (`base-system`), along with btrfs utilities (`btrfs-progs`) and dm-crypt utilities (`cryptsetup`).

Running `chroot`

Mount the pseudo file systems needed for chroot:

```
# for dir in dev proc sys run; do mount --rbind /$dir /mnt/$dir; mount --make-rsla
```

Copy the DNS configuration into the new root so that XBPS can still download new packages inside the chroot:

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

Finally, `chroot` into the new installation:

```
# BTRFS_OPTS=$BTRFS_OPTS PS1='chroot# ' chroot /mnt/ /bin/bash
```

> **Note:** The variable `BTRFS_OPTS` is passed into the chroot so we can use it later when setting fstab (`/etc/fstab`).

Inside the chroot jail

Basic configuration

Hostname

Write the desired hostname to `/etc/hostname`.

System configuration information

Refer to this documentation in order to configure your `rc.conf` file.

Configuring locales

For *glibc* installations, edit `/etc/default/libc-locales`, then run:

```
chroot# xbps-reconfigure -f glibc-locales
```

Root password

```
chroot# passwd
```

Configuring fstab

If you store the appropriate data before creating the fstab file, you don't even need an editor:

```
chroot# UEFI_UUID=$(blkid -s UUID -o value /dev/sda1)
chroot# GRUB_UUID=$(blkid -s UUID -o value /dev/sda2)
chroot# ROOT_UUID=$(blkid -s UUID -o value /dev/mapper/cryptoroot)
chroot# cat <<EOF > /etc/fstab
UUID=$ROOT_UUID / btrfs $BTRFS_OPTS,subvol=@ 0 1
UUID=$UEFI_UUID /efi vfat defaults,noatime 0 2
UUID=$GRUB_UUID /boot ext2 defaults,noatime 0 2
UUID=$ROOT_UUID /home btrfs $BTRFS_OPTS,subvol=@home 0 2
UUID=$ROOT_UUID /.snapshots btrfs $BTRFS_OPTS,subvol=@snapshots 0 2
tmpfs /tmp tmpfs defaults,nosuid,nodev 0 0
EOF
```

Setting up Dracut

I suggest doing a *hostonly* install, that is, Dracut will generate a lean initramfs with everything you might need:

```
chroot# echo hostonly=yes >> /etc/dracut.conf
```

> **Note:** For example, with *hostonly* activated, Dracut will include i915 DRM drivers if you have an Intel CPU with integrated graphics.

Finishing installation

Intel microcode

```
chroot# xbps-install -Su void-repo-nonfree intel-ucode
```

> **Note:** If you're using an AMD CPU, install `linux-firmware-amd` instead.

GRUB

```
chroot# xbps-install grub-x86_64-efi
chroot# grub-install --target=x86_64-efi --efi-directory=/efi --bootloader-id="Voi
```

Swapfile

In order to have an encrypted swap, let's use a more clean approach by using a *swapfile* for, *duh*, swap. For this example, I'll create a swapfile of 16 GiB, but you can choose the best size for your installation:

```
chroot# btrfs subvolume create /var/swap
chroot# truncate -s 0 /var/swap/swapfile
chroot# chattr +C /var/swap/swapfile
chroot# btrfs property set /var/swap/swapfile compression none
chroot# chmod 600 /var/swap/swapfile
chroot# dd if=/dev/zero of=/var/swap/swapfile bs=1G count=16 status=progress
chroot# mkswap /var/swap/swapfile
chroot# swapon /var/swap/swapfile
```

Now, you'll need a tool to calculate the `resume_offset` kernel parameter for btrfs. You can either follow this guide on Arch Linux's wiki or you can use XBPS to build a template I maintain myself of the same code cited in Arch's wiki.

After calculating the value, set it to a variable, let's say, `RESUME_OFFSET`, and append the following to GRUB's config:

```
chroot# echo GRUB_CMDLINE_LINUX="resume=UUID=$ROOT_UUID resume_offset=$RESUME_OFFS
```

> **Note:** This won't be an issue with Void, but be aware that you need Linux kernel version 5.0+ in order to use a swapfile with btrfs.

Regenerating configurations

Finally, regenerate configurations and reboot the machine:

```
chroot# xbps-reconfigure -fa
chroot# exit
# umount -R /mnt
# shutdown -r now
```

Post-installation

If everything went well in the previous chapter, your Void system has successfully booted and you have unlocked your disk in order to be prompted with the login screen in a terminal. Since you only have root available, the first thing to be done is *not* to be root anymore, so let's create a user account.

Creating the main user

Log in as root, set your username to a variable, let's say, `MY_USERNAME`, and run:

```
# xbps-install -S zsh
# useradd -m -G wheel,input,video -s /bin/zsh $MY_USERNAME
# passwd $MY_USERNAME
# visudo
```

After running `visudo`, uncomment the line that contains `%wheel`. Log out and then log in with the newly created user.

> **Note:** If you want to lock down the root account, you can run `sudo passwd -dl root`. Be careful though, as you won't be able to log in using the root account anymore, so make sure your account has root

privileges with `sudo`!