Void Handbook

27 July 2020

Contents

1 About

Welcome to the Void Handbook! Please be sure to read the "About this Handbook" section to learn how to use this documentation effectively.

Void is an independent, rolling release Linux distribution, developed from scratch rather than as a fork, with a focus on stability over bleeding-edge. In addition, there are several features that make Void unique:

- The XBPS package manager, which is extremely fast, developed in-house, and performs checks when installing updates to ensure that libraries are not changed to incompatible versions which can break dependencies.
- The musl libc, which focuses on standards compliance and correctness, has first class support. This allows us to build certain components for musl systems statically, which would not be practical on glibc systems.
- The LibreSSL fork is used instead of the mainline OpenSSL library. Developed as part of the OpenBSD project, LibreSSL is dedicated to the security, quality, and maintainability of this critical library.
- runit is used for init(8) and service supervision. This allows Void to support musl as a second libc choice, which would not be possible with systemd. A side effect of this decision is a core system with clean and efficient operation, and a small code base.

Void is developed in the spare time of a handful of developers, and is generally considered stable enough for daily use. We do this for fun and hope that our work will be useful to others.

The name "Void" comes from the C literal `void`. It was chosen rather randomly, and has no deeper meaning.

2 History

Knowledge of the ancients, grepped from the git logs themselves:

- Sep 26 2008: first git import of void-packages
- Aug 17 2009: first git import of xbps
- Mar 1 2013: first musl toolchains added
- Jul 14 2014: begin switching to LibreSSL
- Jul 28 2014: switch from systemd to runit

3 About this Handbook

This handbook is not an extensive guide on how to use and configure common Linux software. The purpose of this document is to explain how to install, configure, and maintain Void Linux systems, and to highlight the differences between common Linux distributions and Void.

To search for a particular term within the Handbook, select the 'magnifying glass' icon, or press 's'.

Those looking for tips and tricks on how to configure a Linux system in general should consult upstream software documentation. Additionally, the Arch Wiki provides a fairly comprehensive outline of common Linux software configuration, and a variety of internet search engines are available for further assistance.

3.1 Reading The Manuals

While this handbook does not provide a large amount of copy and paste configuration instructions, it does provide links to the man pages for the referenced software wherever possible.

To learn how to use the man(1) man page viewer, run the command `man man`. It can be configured by editing `/etc/man.conf`; read man.conf(5) for details.

Void uses the mandoc toolset for man pages. mandoc was formerly known as "mdocml", and is provided by the `mdocml` package.

3.2 Example Commands

Examples in this guide may have snippets of commands to be run in your shell. When you see these, any line beginning with `$` is run as your normal user. Lines beginning with `#` are run as `root`. After either of these lines, there may be example output from the command.

3.2.1 Placeholders

Some examples include text with placeholders. Placeholders indicate where you should substitute the appropriate information. For example:

```
# ln -s /etc/sv/<service_name> /var/service/
```

This means you need to substitute the text `<service_name>` with the actual service name.

4 InfraDocs

InfraDocs is the meta-manual for the Void project systems management.

5 Installation

This section includes general information about the process of installing Void. For specific guides, see the "Advanced Installation" section.

5.1 Base system requirements

Void can be installed on very minimalist hardware, though we recommend the following minimums for most installations:

Architecture	CPU	RAM	Storage
x86_64-glibc	x86_64	96MB	700MB
x86_64-musl	x86_64	96MB	600MB
i686-glibc	Pentium 4 (SSE2)	96MB	700MB

Note that flavor installations require more resources; how much more depends on the flavor.

Void is not available for the i386, i486, or i586 architectures.

Before installing musl Void, please read the "musl" section of this Handbook, so that you are aware of software incompatibilities.

It is highly recommended to have a network connection available during install to download updates, but this is not required. ISO images contain installation data on-disk and can be installed without network connectivity.

5.2 Downloading installation media

The most recent live images and rootfs tarballs can be downloaded from https://alpha.de.repo.voidlinux.org/live/current/. They can also be downloaded from other mirrors. Previous releases can be found under https://alpha.de.repo.voidlinux.org/live/, organized by date.

5.2.1 Verifying images

Each image release's directory contains two files used to verify the image(s) you download. First, there is a `sha256.txt` file containing image checksums to verify the integrity of the downloaded images. Second is the `sha256.sig` file, used to verify the authenticity of the checksums.

It is necessary to verify both the image's integrity and authenticity. It is, therefore, recommended that you download both files.

Verifying image integrity You can verify the integrity of a downloaded file using sha256sum(1) with the `sha256.txt` file downloaded above. The following command will check the integrity of only the image(s) you have downloaded:

```
$ sha256sum -c --ignore-missing sha256.txt
void-live-x86_64-musl-20170220.iso: OK
```

This verifies that the image is not corrupt.

5.2.2 Verifying digital signature

Prior to using any image you're strongly encouraged to validate the signatures on the image to ensure they haven't been tampered with.

Current images are signed using a signify key that is specific to the release. If you're on Void already, you can obtain the keys from the `void-release-keys` package, which will be downloaded using your existing XBPS trust relationship with your mirror. You will also need a copy of signify(1); on Void this is provided by the `outils` package.

To obtain `signify` when using a Linux distribution or operating system other than Void Linux:

- Install the `signify` package in Arch Linux and Arch-based distros.
- Install the `signify-openbsd` package in Debian and Debian-based distros.
- Install the package listed here for your distribution.
- Install `signify-osx` with homebrew in macOS.

If you can't obtain `signify` for some reason (e.g. you are on Windows and can't use WSL or MinGW), you can use minisign(1) to verify the file.

If you are not currently using Void Linux, it will also be necessary to obtain the appropriate signing key from our Git repository here.

Once you've obtained the key, you can verify your image with the `sha256.sig` file. The following example demonstrates the verification of the GCP musl filesystem from the 20191109 release:

```
$ signify -C -p /etc/signify/void-release-20191109.pub -x
    sha256.sig void-GCP-musl-PLATFORMFS-20191109.tar.xz
Signature Verified
void-GCP-musl-PLATFORMFS-20191109.tar.xz: OK
```

If the verification process does not produce the expected "OK" status, do not use it! Please alert the Void Linux team of where you got the image and how you verified it, and we will follow up on it.

For verification with `minisign`, it is necessary to rename the `sha256.sig` file to `sha256.txt.minisig` and remove the first line from the `.pub` release key. The following example demonstrates the verification of the `sha256.txt` file from the 20191109 release:

```
$ minisign -Vm sha256.txt -f -p void-release-20191109.pub
void-release-20191109.pub: Success
```

The same warning as above applies. If the verification process isn't successful, do not use the file - warn the Void Linux team about it.

6 Live Installers

Void provides live installer images containing a base set of utilities, an installer program, and package files to install a new Void system. These live images are also useful for repairing a system that is not able to boot or function properly.

There are `x86_64` images for both `glibc` and `musl` based systems. There are also images for `i686`, but only `glibc` is supported for this architecture. Live installers are not provided for other architectures. Users of other architectures will need to use rootfs tarballs, or perform an installation manually.

6.1 Installer images

Void releases two types of images: base images and "flavor" images. Linux beginners are encouraged to try one of the more full-featured flavor images, but more advanced users may often prefer to start from a base image to install only the packages they need.

6.1.1 Base images

The base images provide only a minimal set of packages to install a usable Void system. These base packages are only those needed to configure a new machine, update the system, and install additional packages from repositories.

6.1.2 Flavor images

Each of the Void "flavor" images includes a full desktop environment, web browser, and basic applications configured for that environment. The only difference from the base images is the additional packages and services installed.

The install process for each of the flavor images is the same as the base images, except that you **must** select the `Local` source when installing. If you select `Network` instead, the installer will download and install the latest version of the base system, without any additional packages included on the live image.

Comparison of flavor images Here's a quick overview of the main components and applications included with each flavor:

	Enlightenment	Cinnamon	LXDE	LXQT	MATE	XFCE
Window Manager	Enlightenment Window Manager	Mutter (Muffin)	Openbox	Openbox	Metacity (Marco)	xfwm4
File Manager	Enlightenment File Manager	Nemo	PCManFM	PCManFM-Qt	Caja	Thunar
Web Browser	Firefox ESR	Firefox ESR	Firefox ESR	QupZilla	Firefox ESR	Firefox ESR
Terminal	Terminology	gnome-terminal	LXTerminal	QTerminal	MATE terminal	xfce4-Terminal
Document Viewer	-	-	-	-	Atril (PS/PDF)	-
Plain text viewer	-	-	-	-	Pluma	Mousepad
Image viewer	-	-	GPicView	LXImage	Eye of MATE	Ristretto
Archive unpacker	-	-	-	-	Engrampa	-
Other	Mixer, EConnMan (connection manager), Elementary Test	-	LXTask (task manager), MIME type editor	Screen grabber	Screen grabber, file finder, MATE color picker, MATE font viewer, Disk usage analyzer, Power statistics, System monitor (task manager), Dictionary, Log file viewer	Bulk rename, Orage Globaltime, Orage Calendar, Task Manager, Parole Media Player, Audio Mixer, MIME type editor, Application finder

7 Prepare Installation Media

After downloading a live image, it must be written to bootable media, such as a USB drive, SD card, or CD/DVD.

7.1 Create a bootable USB drive or SD card on Linux

7.1.1 Identify the Device

Before writing the image, identify the device you'll write it to. You can do this using fdisk(8). After connecting the storage device, identify the device path by running:

```
# fdisk -l
Disk /dev/sda: 7.5 GiB, 8036286464 bytes, 15695872 sectors
Disk model: Your USB Device's Model
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

In the example above, the output shows the USB device as `/dev/sda`. On Linux, the path to the device will typically be in the form of `/dev/sdX` (where X is a number) for USB devices, `/dev/mmcblkX` for SD cards, or other variations depending on the device. You can use the model and size ('7.5GiB' above, after the path) to identify the device if you're not sure what path it will have.

Once you've identified the device you'll use, ensure it's not mounted by unmounting it with umount(8):

```
# umount /dev/sdX
umount: /dev/sdX: not mounted.
```

7.1.2 Write the live image

The dd(1) command can be used to copy a live image to a storage device. Using dd, write the live image to the device:

> **Warning**: this will destroy any data currently on the referenced device.
>
> Exercise caution.

```
# dd bs=4M if=/path/to/void-live-ARCH-DATE-VARIANT.iso of=/
   dev/sdX
90+0 records in
90+0 records out
377487360 bytes (377 MB, 360 MiB) copied, 0.461442 s, 818 MB/
   s
```

dd won't print anything until it's completed (or if it failed), so depending on the device, this can take a few minutes or longer.

Finally, ensure all data is flushed before disconnecting the device:

```
$ sync
```

The number of records, amount copied, and rates will all vary depending on the device and the live image you chose.

7.2 Burning to a CD or DVD

Any disk burning application should be capable of writing the `.iso` file to a CD or DVD. The following free software applications are available (cross-platform support may vary):

- Brasero
- K3B
- Xfburn

  Note: with a CD or DVD, live sessions will be less responsive than with a USB

  or hard drive.

8 Partitioning notes

Partitioning for a modern Linux distribution is generally very simple, however the introduction of GPT and UEFI booting does bring new complexity to the process. When creating your new partition table you will need a partition for the root filesystem, along with a swap partition and possibly another partition or two to facilitate booting, if required.

Note that if the disk has already been initialized, the top of the `cfdisk` screen will show the partition layout already present: `Label: dos` for the MBR scheme, `Label: gpt` for the GPT scheme. If you just want to erase the partition table before starting the installer, use `wipefs(8)`. Otherwise, you can run `cfdisk(8)` manually with the `-z` option to start with an uninitialized disk layout; `cfdisk` will prompt you for the label type before continuing to the main screen.

The following sections will detail the options for partition configuration.

8.1 BIOS system notes

It is recommended that you create an MBR partition table if you are using a BIOS boot system. This will limit the number of partitions you create to four. It is possible to install a GPT partition table on a BIOS system, but grub will need a special partition to boot properly.

8.2 UEFI system notes

UEFI users are recommended to create a GPT partition table. UEFI booting with grub also requires a special partition of the type `EFI System` with a `vfat` filesystem mounted at `/boot/efi`. A reasonable size for this partition could be between 200MB and 1GB. With this partition setup during the live image installation, the installer should successfully set up the bootloader automatically.

8.3 Swap partitions

A swap partition is not strictly required, but recommended for systems with low RAM. If you want to use hibernation, you will need a swap partition. The following table has recommendations for swap partition size.

System RAM	Recommended swap space	Swap space if using hibernation
$<$ 2GB	2x the amount of RAM	3x the amount of RAM
2-8GB	Equal to amount of RAM	2x the amount of RAM
8-64GB	At least 4GB	1.5x the amount of RAM
64GB	At least 4GB	Hibernation not recommended

8.4 Boot partition (optional)

On most modern systems, a separate `/boot` partition is no longer necessary to boot properly. If you choose to use one, note that Void does not remove old kernels after updates by default and also that the kernel tends to increase in size with each new

version, so plan accordingly (e.g. `/boot` with one Linux 5.x `x86_64` kernel and grub occupies about 60MB).

8.5 Other partitions

It is fine to install your system with only a large root partition, but you may create other partitions if you want. One helpful addition could be a separate partition for your `/home` directory. This way if you need to reinstall Void (or another distribution) you can save the data and configuration files in your home directory for your new system.

9 Installation Guide

Once you have downloaded a Void image to install and prepared your install media, you are ready to install Void Linux.

Before you begin installation, you should determine whether your machine boots using BIOS or UEFI. This will affect how you plan partitions. See Partitioning Notes for more detail.

The following features are not supported by the installer script:

- LVM
- LUKS
- ZFS

9.1 Booting

Boot your machine from the install media you created. If you have enough RAM, there is an option on the boot screen to load the entire image into ram, which will take some time but speed up the rest of the install process.

Once the live image has booted, log in as `root` with password `voidlinux` and run:

```
# void-installer
```

The following sections will detail each screen of the installer.

9.2 Keyboard

Select the keymap for your keyboard; standard "qwerty" keyboards will generally use the "us" keymap.

9.3 Network

Select your primary network interface. If you do not choose to use DHCP, you will be prompted to provide an IP address, gateway, and DNS servers.

If you intend to use a wireless connection during the installation, you may need to configure it manually using wpa_supplicant and dhcpcd manually before running `void-installer`.

9.4 Source

To install packages provided on the install image, select `Local`. Otherwise, you may select `Network` to download the latest packages from the Void repository.

Warning!: If you are installing a desktop environment from a ”flavor” image, you MUST choose `Local` for the source!

9.5 Hostname

Select a hostname for your computer (that is all lowercase, with no spaces.)

9.6 Locale

Select your default locale settings. This option is for glibc only, as musl does not currently support locales.

9.7 Timezone

Select your timezone based on standard timezone options.

9.8 Root password

Enter and confirm your `root` password for the new installation. The password will not be shown on screen.

9.9 User account

Choose a login (default `void`) and a descriptive name for that login. Then enter and confirm the password for the new user. You will then be prompted to verify the groups for this new user. They are added to the `wheel` group by default and will have `sudo` access.

9.10 Bootloader

Select the disk to install a bootloader on when Void is installed. You may select `none` to skip this step and install a bootloader manually after completing the installation process. If installing a bootloader, you will also be asked whether or not you want a graphical terminal for the GRUB menu.

9.11 Partition

Next, you will need to partition your disks. Void does not provide a preset partition scheme, so you will need to create your partitions manually with cfdisk(8). You will be prompted with a list of disks. Select the disk you want to partition and the installer will launch `cfdisk` for that disk. Remember you must write the partition table to the drive before you exit the partition editor.

If using UEFI, it is recommended you select GPT for the partition table and create a partition (typically between 200MB-1GB) of type `EFI System`, which will be mounted at `/boot/efi`.

If using BIOS, it is recommended you select MBR for the partition table. Advanced users may use GPT but will need to create a special BIOS partition for `grub` to boot.

See the Partitioning Notes for more details about partitioning your disk.

9.12 Filesystems

Create the filesystems for each partition you have created. For each partition you will be prompted to choose a filesystem type, whether you want to create a new filesystem on the partition, and a mount point, if applicable. When you are finished, select `Done` to return to the main menu.

If using UEFI, create a `vfat` filesystem and mount it at `/boot/efi`.

9.13 Review settings

It is a good idea to review your settings before proceeding. Use the right arrow key to select the settings button and hit `<enter>`. All your selections will be shown for review.

9.14 Install

Selecting `Install` from the menu will start the installer. The installer will create all the filesystems selected, and install the base system packages. It will then generate an initramfs and install a GRUB2 bootloader to the bootable partition.

These steps will all run automatically, and after the installation is completed successfully, you can reboot into your new Void Linux install!

9.15 Post installation

After booting into your Void installation for the first time, perform a system update.

10 Advanced Installation Guides

This section contains guides for more specific or complex use-cases.

10.1 Section Contents

- Installing Void via chroot (x86 or x86_64)
- Installing Void with Full Disk Encryption

11 Installation via chroot (x86/x86_64)

This guide details the process of manually installing Void via a chroot on an x86 or x86_64 PC architecture. It is assumed that you have a familiarity with Linux, but not necessarily with installing a Linux system via a chroot. This guide can be used to create a a "typical" setup, using a single partition on a single SATA/IDE/USB disk. Each step may be modified to create less typical setups, such as full disk encryption.

Void provides two options for bootstrapping the new installation. The **XBPS method** uses the XBPS Package Manager running on a host operating system to install the base system. The **ROOTFS method** installs the base system by unpacking a ROOTFS tarball.

The **XBPS method** requires that the host operating system have XBPS installed. This may be an existing installation of Void, an official live image, or any Linux installation running a statically linked XBPS.

The **ROOTFS method** requires only a host operating system that can enter a Linux chroot and that has both tar(1) and xz(1) installed. This method may be preferable if you wish to install Void using a different Linux distribution.

11.1 Prepare Filesystems

Partition your disks and format them using mke2fs(8), mkfs.xfs(8), mkfs.btrfs(8) or whatever tools are necessary for your filesystem(s) of choice.

mkfs.vfat(8) is also available to create FAT32 partitions. However, due to restrictions associated with FAT filesystems, it should only be used when no other filesystem is suitable (such as for the EFI System Partition).

cfdisk(8) and fdisk(8) are available on the live images for partitioning, but you may wish to use gdisk(8) (from the package `gptfdisk`) or parted(8) instead.

For a UEFI booting system, make sure to create an EFI System Partition (ESP). The ESP should have the partition type "EFI System" (code `EF00`) and be formatted as FAT32 using mkfs.vfat(8).

If you're unsure what partitions to create, create a 1GB partition of type "EFI System" (code `EF00`), then create a second partition of type "Linux Filesystem" (code `8300`) using the remainder of the drive.

Format these partitions as FAT32 and ext4, respectively:

```
# mkfs.vfat /dev/sda1
# mkfs.ext4 /dev/sda2
```

11.1.1 Create a New Root and Mount Filesystems

This guide will assume the new root filesystem is mounted on `/mnt`. You may wish to mount it elsewhere.

If using UEFI, mount the EFI System Partition as `/mnt/boot/efi`.

For example, if `/dev/sda2` is to be mounted as `/` and `dev/sda1` is the EFI System Partition:

```
# mount /dev/sda2 /mnt/
# mkdir -p /mnt/boot/efi/
# mount /dev/sda1 /mnt/boot/efi/
```

Initialize swap space, if desired, using mkswap(8).

11.2 Base Installation

Follow only one of the two following subsections.

11.2.1 The XBPS Method

Select a mirror and use the appropriate URL for the type of system you wish to install. For simplicity, save this URL to a shell variable for later use, e.g.:

```
# REPO=https://alpha.de.repo.voidlinux.org/current
```

XBPS also needs to know what architecture is being installed. Available options are `x86_64`, `x86_64-musl` and `i686` for PC architecture computers. For example:

```
# ARCH=x86_64
```

This architecture must be compatible with your current operating system, but does not need to be the same. If your host is running an x86_64 operating system, any of the three architectures can be installed (whether the host is musl or glibc), but an i686 host can only install i686 distributions.

Use xbps-install(1) to bootstrap the installation by installing the `base-system` meta-package:

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

`xbps-install` might ask you to verify the RSA keys for the packages you are installing.

11.2.2 The ROOTFS Method

Download a ROOTFS tarball matching your architecture.

Unpack the tarball into the newly configured filesystems:

```
# tar xvf void-<...>-ROOTFS.tar.xz -C /mnt
```

11.3 Configuration

With the exception of the section "Install base-system (ROOTFS method only)", the remainder of this guide is common to both the XBPS and ROOTFS installation methods.

11.3.1 Entering the Chroot

Mount the pseudo-filesystems needed for a chroot:

```
# mount --rbind /sys /mnt/sys && mount --make-rslave /mnt/sys
# mount --rbind /dev /mnt/dev && mount --make-rslave /mnt/dev
# mount --rbind /proc /mnt/proc && mount --make-rslave /mnt/
   proc
```

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/
```

Chroot into the new installation:

```
# PS1='(chroot) # ' chroot /mnt/ /bin/bash
```

11.3.2 Install base-system (ROOTFS method only)

ROOTFS images generally contain out of date software, due to being a snapshot of the time when they were built, and do not come with a complete `base-system`. Update the package manager and install `base-system`:

```
# xbps-install -Su xbps
# xbps-install -u
# xbps-install base-system
# xbps-remove base-voidstrap
```

11.3.3 Installation Configuration

Specify the hostname in `/etc/hostname`. Go through the options in '/etc/rc.conf'. If installing a glibc distribution, edit `/etc/default/libc-locales`, uncommenting desired locales.

nvi(1) is available in the chroot, but you may wish to install your preferred text editor at this time.

For glibc builds, generate locale files with:

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

11.3.4 Set a Root Password

Configure at least one super user account. Other user accounts can be configured later, but there should either be a root password, or a new user account with sudo(8) privileges.

To set a root password, run:

```
(chroot) # passwd
```

11.3.5 Configure fstab

The fstab(5) file can be automatically generated from currently mounted filesystems by copying the file `/proc/mounts`:

```
(chroot) # cp /proc/mounts /etc/fstab
```

Remove lines in `/etc/fstab` that refer to `proc`, `sys`, `devtmpfs` and `pts`.

Replace references to `/dev/sdXX`, `/dev/nvmeXnYpZ`, etc. with their respective UUID, which can be found by running blkid(8). Referring to filesystems by their UUID guarantees they will be found even if they are assigned a different name at a later time. In some situations, such as booting from USB, this is absolutely essential. In other situations, disks will always have the same name unless drives are physically added or removed. Therefore, this step may not be strictly necessary, but is almost always recommended.

Change the last zero of the entry for `/` to `1`, and the last zero of every other line to `2`. These values configure the behaviour of fsck(8).

For example, the partition scheme used throughout previous examples yields the following `fstab`:

```
/dev/sda1       /boot/EFI   vfat    rw,relatime,[...]       0
    0
/dev/sda2       /           ext4    rw,relatime             0
    0
```

The information from `blkid` results in the following `/etc/fstab`:

```
UUID=6914[...]  /boot/EFI   vfat    rw,relatime,[...]       0
    2
UUID=dc1b[...]  /           ext4    rw,relatime             0
    1
```

Note: The output of `/proc/mounts` will have a single space between each field. The columns are aligned here for readability.

Add an entry to mount `/tmp` in RAM:

```
tmpfs           /tmp        tmpfs   defaults,nosuid,nodev   0
    0
```

If using swap space, add an entry for any swap partitions:

```
UUID=1cb4[...]  swap        swap    rw,noatime,discard      0
    0
```

11.4 Installing GRUB

Use grub-install to install GRUB onto your boot disk.

On a BIOS computer, install the package `grub`, then run `grub-install /dev/sdX`, where `/dev/sdX` is the drive (not partition) that you wish to install GRUB to. For example:

```
(chroot) # xbps-install grub
(chroot) # grub-install /dev/sda
```

On a UEFI computer, install either `grub-x86_64-efi` or `grub-i386-efi`, depending on your architecture, then run `grub-install`, optionally specifying a bootloader label (this label may be used by your computer's firmware when manually selecting a boot device):

```
(chroot) # xbps-install grub-x86_64-efi
(chroot) # grub-install --bootloader-id="Void"
```

If installing onto a removable disk (such as USB), add the option `-removable` to the `grub-install` command.

11.5 Finalization

Use xbps-reconfigure(1) to ensure all installed packages are configured properly:

```
(chroot) # xbps-reconfigure -fa
```

This will make dracut(8) generate an initramfs, and will make GRUB generate a working configuration.

At this point, the installation is complete. Exit the chroot and reboot your computer:

```
(chroot) # exit
# shutdown -r now
```

After booting into your Void installation for the first time, perform a system update.

12 Full Disk Encryption

Your drive's block device and other information may be different, so make sure it is correct.

Boot the live image and login.

Create a single physical partition on the disk using `cfdisk`, marking it bootable. For an MBR system, the partition layout should look like the following.

```
# fdisk -l /dev/sda
Disk /dev/sda: 48 GiB, 51539607552 bytes, 100663296 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x4d532059

Device     Boot Start       End   Sectors Size Id Type
/dev/sda1  *     2048 100663295 100661248  48G 83 Linux
```

UEFI systems will need the disk to have a GPT disklabel and an EFI system partition. The required size for this may vary depending on needs, but 100M should be enough for most cases. For an EFI system, the partition layout should look like the following.

```
# fdisk -l /dev/sda
Disk /dev/sda: 48 GiB, 51539607552 bytes, 100663296 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: EE4F2A1A-8E7F-48CA-B3D0-BD7A01F6D8A0

Device      Start       End   Sectors  Size Type
/dev/sda1    2048    264191    262144  128M EFI System
/dev/sda2  264192 100663262 100399071 47.9G Linux filesystem
```

Configure the encrypted volume. `cryptsetup` defaults to LUKS2, yet `grub` currently only has support for LUKS1, so it is critical to force LUKS1. Keep in mind this will be `/dev/sda2` on EFI systems.

```
# cryptsetup luksFormat --type luks1 /dev/sda1

WARNING!
========
This will overwrite data on /dev/sda1 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter passphrase:
Verify passphrase:
```

Once the volume is created, it needs to be opened. Replace voidvm with an appropriate name. Again, this will be `/dev/sda2` on EFI systems.

```
# cryptsetup luksOpen /dev/sda1 voidvm
Enter passphrase for /dev/sda1:
```

Once the LUKS container is opened, create the LVM volume group using that partition.

```
# vgcreate voidvm /dev/mapper/voidvm
  Volume group "voidvm" successfully created
```

There should now be an empty volume group named `voidvm`.

Next, logical volumes need to be created for the volume group. For this example, I chose 10G for `/`, 2G for `swap`, and will assign the rest to `/home`.

```
# lvcreate --name root -L 10G voidvm
  Logical volume "root" created.
# lvcreate --name swap -L 2G voidvm
  Logical volume "swap" created.
# lvcreate --name home -l 100%FREE voidvm
  Logical volume "home" created.
```

Next, create the filesystems. The example below uses XFS as a personal preference of the author. Any filesystem supported by GRUB will work.

```
# mkfs.xfs -L root /dev/voidvm/root
meta-data=/dev/voidvm/root       isize=512    agcount=4,
    agsize=655360 blks
...
# mkfs.xfs -L home /dev/voidvm/home
meta-data=/dev/voidvm/home       isize=512    agcount=4,
    agsize=2359040 blks
...
mkswap /dev/voidvm/swap
Setting up swapspace version 1, size = 2 GiB (2147479552
    bytes)
```

Next, setup the chroot and install the base system.

```
# mount /dev/voidvm/root /mnt
# for dir in dev proc sys run; do mkdir -p /mnt/$dir ; mount
    --rbind /$dir /mnt/$dir ; mount --make-rslave /mnt/$dir ;
    done
# mkdir -p /mnt/home
# mount /dev/voidvm/home /mnt/home
```

On a UEFI system, the EFI system partition also needs to be mounted.

```
# mkfs.vfat /dev/sda1
# mkdir -p /mnt/boot/efi
# mount /dev/sda1 /mnt/boot/efi
```

Before we enter the chroot to finish up configuration, we do the actual install. `xbps-install` might ask you to verify the RSA keys for the packages you are installing.

```
# xbps-install -Sy -R https://alpha.de.repo.voidlinux.org/
   current -r /mnt base-system lvm2 cryptsetup grub
[*] Updating 'https://alpha.de.repo.voidlinux.org/current/
   x86_64-repodata' ...
x86_64-repodata: 1661KB [avg rate: 2257KB/s]
'https://alpha.de.repo.voidlinux.org/current' repository has
   been RSA signed by "Void Linux"
Fingerprint: 60:ae:0c:d6:f0:95:17:80:bc:93:46:7a:89:af:a3:2d
Do you want to import this public key? [Y/n] y
130 packages will be downloaded:
...
```

UEFI systems will have a slightly different package selection. The installation command for a UEFI system will be as follows.

```
# xbps-install -Sy -R https://alpha.de.repo.voidlinux.org/
   current -r /mnt base-system cryptsetup grub-x86_64-efi
   lvm2
```

When it's done, we can enter the `chroot` and finish up the configuration.

```
# chroot /mnt
# chown root:root /
# chmod 755 /
# passwd root
# echo voidvm > /etc/hostname
# echo "LANG=en_US.UTF-8" > /etc/locale.conf
# echo "en_US.UTF-8 UTF-8" >> /etc/default/libc-locales
# xbps-reconfigure -f glibc-locales
```

The next step is editing `/etc/fstab`, which will depend on how you configured and named your filesystems. For this example, the file should look like this:

```
# <file system>    <dir> <type>  <options>             <dump>
     <pass>
tmpfs             /tmp  tmpfs   defaults,nosuid,nodev 0
          0
/dev/voidvm/root  /     xfs     defaults              0
          0
/dev/voidvm/home  /home xfs     defaults              0
          0
```

```
/dev/voidvm/swap   swap   swap     defaults                0
          0
```

UEFI systems will also have an entry for the EFI system partition.

```
/dev/sda1        /boot/efi        vfat     defaults          0
           0
```

Next, configure GRUB to be able to unlock the filesystem. Add the following line to `/etc/default/grub`:

```
GRUB_ENABLE_CRYPTODISK=y
```

Next, the kernel needs to be configured to find the encrypted device. First, find the UUID of the device.

```
# lsblk -l -o NAME,UUID
NAME  UUID
sda
sda1  135f3c06-26a0-437f-a05e-287b036440a4
...
```

Edit the `GRUB_CMDLINE_LINUX_DEFAULT=` line in `/etc/default/grub` and add `rd.lvm.vg=voidvm rd.luks.uuid=<UUID>` to it. Make sure the UUID matches the one for the `sda1` device found in the output of the `lsblk` command above.

And now to avoid having to enter the password twice on boot, a key will be configured to automatically unlock the encrypted volume on boot. First, generate a random key.

```
# dd bs=512 count=4 if=/dev/urandom of=/boot/volume.key
4+0 records in
4+0 records out
2048 bytes (2.0 kB, 2.0 KiB) copied, 0.000421265 s, 4.9 MB/s
```

Next, add the key to the encrypted volume.

```
# cryptsetup luksAddKey /dev/sda1 /boot/volume.key
Enter any existing passphrase:
```

Change the permissions to protect generated the key.

```
# chmod 000 /boot/volume.key
# chmod -R g-rwx,o-rwx /boot
```

This keyfile also needs to be added to `/etc/crypttab`. Again, this will be `/dev/sda2` on EFI systems.

```
voidvm   /dev/sda1   /boot/volume.key   luks
```

And then the keyfile and `crypttab` need to be included in the initramfs. Create a new file at `/etc/dracut.conf.d/10-crypt.conf` with the following line:

```
install_items+=" /boot/volume.key /etc/crypttab "
```

Next, install the boot loader to the disk.

```
# grub-install /dev/sda
```

Ensure an initramfs is generated. Replace `X.X` in the following command line with the installed kernel version.

```
# xbps-reconfigure -f linuxX.X
```

Exit the `chroot`, unmount the filesystems, and reboot the system.

```
# exit
# umount -R /mnt
# reboot
```

13 musl

musl is a libc implementation which strives to be lightweight, fast, simple, and correct.

Void officially supports musl by using it in its codebase for all target platforms (although binary packages are not available for i686). Additionally, all compatible packages in our official repositories are available with musl-linked binaries in addition to their glibc counterparts.

Currently, there are nonfree and debug sub-repositories for musl, but no multilib subrepo.

13.1 Incompatible software

Musl practices very strict and minimal standard compliance. Many commonly used platform-specific extensions are not present. Because of this, it is common for software to need modification to compile and/or function properly. Void developers work to patch such software and hopefully get portability/correctness changes accepted into the upstream projects.

Proprietary software rarely supports non-glibc libc implementations, although sometimes these applications are available as flatpaks, which provide their own libc in the image.

13.1.1 glibc chroot

Software requiring glibc can be run in a glibc chroot.

Create a directory that will contain the chroot, and install a base system in it via the `base-voidstrap` package. If network access is required, copy `/etc/resolv.conf` into the chroot; `/etc/hosts` may need to be copied as well.

Several directories then need to be mounted as follows:

```
# mount -t proc none <chroot_dir>/proc
# mount -t sysfs none <chroot_dir>/sys
# mount --rbind /dev <chroot_dir>/dev
# mount --rbind /run <chroot_dir>/run
```

Use chroot(1) to change to the new root, then run glibc programs as usual. Once you've finished using it, unmount the chroot using umount(8).

PRoot An alternative to the above is proot(1), a user-space implementation of chroot, mount –bind, and binfmt_misc. By installing the `proot` package, unprivileged users can utilize a chroot environment.

14 Configuration

This section and its subsections provide information about configuring your Void system.

15 Firmware

Void provides a number of firmware packages in the repositories. Some firmware is only available if you have enabled the nonfree repository.

15.1 Microcode

Microcode is loaded onto the CPU or GPU at boot by the BIOS, but can be replaced later by the OS itself. An update to microcode can allow a CPU's or GPU's behavior to be modified to work around certain yet to be discovered bugs, without the need to replace the hardware.

15.1.1 Intel

Install the Intel microcode package, `intel-ucode`. This package is in the nonfree repo, which has to be enabled. After installing this package, it is necessary to regenerate your initramfs. For subsequent updates, the microcode will be added to the initramfs automatically.

15.1.2 AMD

Install the AMD package, `linux-firmware-amd`, which contains microcode for both AMD CPUs and GPUs. AMD CPUs and GPUs will automatically load the microcode, no further configuration required.

15.1.3 Verification

The `/proc/cpuinfo` file has some information under `microcode` that can be used to verify the microcode update.

15.2 Removing firmware

By default, `linuxX.Y` packages and the `base-system` package install a number of firmware packages. It is not necessary to remove unused firmware packages, but if you wish to do so, you can configure XBPS to ignore those packages, then remove them.

16 Locales

For a list of currently enabled locales, run

```
$ locale -a
```

16.1 Enabling locales

To enable a certain locale, un-comment or add the relevant lines in `/etc/default/libc-locales` and reconfigure the `glibc-locales` package.

16.2 Setting the system locale

Set `LANG=xxxx` in `/etc/locale.conf`.

16.3 Application locale

Some programs have their translations in a separate package that must be installed in order to use them. You can search for the desired language (e.g. "german" or "portuguese") in the package repositories and install the packages relevant to the applications you use.

17 Users and Groups

The useradd(8), userdel(8) and usermod(8) commands are used to add, delete and modify users respectively. The passwd(1) command is used to change passwords.

The groupadd(8), groupdel(8) and groupmod(8) commands are used to add, delete and modify groups respectively. The groups(1) command lists all groups a user belongs to.

17.1 Default shell

The default shell for a user can be changed with chsh(1):

```
$ chsh -s <shell> <user_name>
```

`<shell>` must be the path to the shell as specified by `/etc/shells` or the output of `chsh -l`, which provides a list of installed shells.

17.2 sudo

Note: sudo(8) is installed by default but

may not be configured. It is only necessary to configure sudo if you wish to

use it.

Use visudo(8) as root to edit the sudoers(5) file.
To create a superuser, uncomment the line

```
#%wheel ALL=(ALL) ALL
```

and add users to the `wheel` group.

17.3 Default Groups

Void Linux defines a number of groups by default.

Group	Description
root	Complete access to the system.
bin	Unused - present for historical reasons.
sys	Unused - present for historical reasons.
kmem	Ability to read from /dev/mem and /dev/port.
wheel	Elevated privileges for specific system administration tasks.
tty	Access to TTY-like devices:
	/dev/tty*, /dev/pts*, /dev/vcs*.
tape	Access to tape devices.
daemon	System daemons that need to write to files on disk.
floppy	Access to floppy drives.
disk	Raw access to /dev/sd* and /dev/loop*.
lp	Access to printers.
dialout	Access to serial ports.
audio	Access to audio devices.
video	Access to video devices.
utmp	Ability to write to /var/run/utmp, /var/log/wtmp
	and /var/log/btmp.
adm	Unused - present for historical reasons. This group was
	traditionally used for system monitoring, such as viewing
	files in /var/log.
cdrom	Access to CD devices.
optical	Access to DVD/CD-RW devices.
mail	Used by some mail packages, e.g. dma.
storage	Access to removable storage devices.
scanner	Ability to access scanners.
network	Unused - present for historical reasons.
kvm	Ability to use KVM for virtual machines, e.g. via QEMU.
input	Access to input devices: /dev/mouse*, /dev/event*.
nogroup	System daemons that don't need to own any files.
users	Ordinary users.
xbuilder	To use xbps-uchroot(1) with xbps-src.

18 Services and Daemons - runit

Void uses the runit(8) supervision suite to run system services and daemons.

Some advantages of using runit include:

- a small code base, making it easier to audit for bugs and security issues.
- each service is given a clean process state, regardless of how the service was started or restarted: it will be started with the same environment, resource limits, open file descriptors, and controlling terminals.
- a reliable logging facility for services, where the log service stays up as long as the relevant service is running and possibly writing to the log.

18.1 Section Contents

18.2 Service Directories

Each service managed by runit has an associated *service directory*.

A service directory requires only one file: an executable named `run`, which is expected to exec a process in the foreground.

Optionally, a service directory may contain:

- an executable named `check`, which will be run to check whether the service is up and available; it's considered available if `check` exits with 0.
- an executable named `finish`, which will be run on shutdown/process stop.
- a `conf` file; this can contain environment variables to be sourced and referenced in `run`.
- a directory named `log`; a pipe will be opened from the output of the `run` process in the service directory to the input of the `run` process in the `log` directory.

When a new service is created, a `supervise` folder will be automatically created on the first run.

18.2.1 Configuring Services

Most services can take configuration options set by a `conf` file in the service directory. This allows service customization without modifying the service directory provided by the relevant package.

Check the service file for how to pass configuration parameters. A few services have a field like `OPTS="-value ..."` in their `conf` file.

To make more complex customizations, you should edit the service.

18.2.2 Editing Services

To edit a service, first copy its service directory to a different directory name. Otherwise, xbps-install(1) can overwrite the service directory. Then, edit the new service file as needed. Finally, the old service should be stopped and disabled, and the new one should be started.

18.3 Managing Services

18.3.1 Runsvdirs

A **runsvdir** is a directory in `/etc/runit/runsvdir` containing enabled services in the form of symlinks to service directories. On a running system, the current runsvdir is accessible via the `/var/service` symlink.

The `runit-void` package comes with two runsvdirs, `single` and `default`:

- `single` just runs sulogin(8) and the necessary steps to rescue your system.
- `default` is the default runsvdir on a running system, unless specified otherwise by the kernel command line.

Additional runsvdirs can be created in `/etc/runit/runsvdir/`.

See runsvdir(8) and runsvchdir(8) for further information.

Booting A Different runsvdir To boot a runsvdir other than `default`, the name of the desired runsvdir can be added to the kernel command-line. As an example, adding `single` to the kernel command line will boot the `single` runsvdir.

18.3.2 Basic Usage

To start, stop, restart or get the status of a service:

```
# sv up <services>
# sv down <services>
# sv restart <services>
# sv status <services>
```

The `<services>` placeholder can be:

- Service names (service directory names) inside the `/var/service` directory.
- The full paths to the services.

For example, the following commands show the status of a specific service and of all enabled services:

```
# sv status dhcpcd
# sv status /var/service/*
```

See sv(8) for further information.

Enabling Services Void Linux provides service directories for most daemons in `/etc/sv/`.

To enable a service on a booted system, create a symlink to the service directory in `/var/service`:

```
# ln -s /etc/sv/<service> /var/service/
```

If the system is not currently running, the service can be linked directly into the `default` runsvdir:

```
# ln -s /etc/sv/<service> /etc/runit/runsvdir/default/
```

This will automatically start the service. Once a service is linked it will always start on boot and restart if it stops, unless administratively downed.

To prevent a service from starting at boot while allowing runit to manage it, create a file named `down` in its service directory:

```
# touch /etc/sv/<service>/down
```

The `down` file mechanism also makes it possible to disable services that are enabled by default, such as the agetty(8) services for ttys 1 to 6. This way, package updates which affect these services - in this case, the `runit-void` package - won't re-enable them.

Disabling Services To disable a service, remove the symlink from the running runsvdir:

```
# rm /var/service/<service>
```

Or, for example, from the `default` runsvdir, if either the specific runsvdir, or the system, is not currently running:

```
# rm /etc/runit/runsvdir/default/<service>
```

Testing Services To check if a service is working correctly when started by the service supervisor, run it once before fully enabling it:

```
# touch /etc/sv/<service>/down
# ln -s /etc/sv/<service> /var/service
# sv once <service>
```

If everything works, remove the `down` file to enable the service.

19 Per-User Services

Sometimes it can be nice to have user-specific runit services. For example, you might want to open an ssh tunnel as the current user, run a virtual machine, or regularly run daemons on your behalf. The most common way to do this is to create a system-level service that runs runsvdir(8) as your user, in order to start and monitor the services in a personal services directory.

For example, you could create a service called `/etc/sv/runsvdir-<username>` with the following `run` script:

```
#!/bin/sh

export USER="<username>"
export HOME="/home/<username>"

groups="$(id -Gn "$USER" | tr ' ' ':')"
svdir="$HOME/service"

exec chpst -u "$USER:$groups" runsvdir "$svdir"
```

In this example chpst(8) is used to start a new runsvdir(8) process as the specified user. chpst(8) does not read groups on its own, but expects the user to list all required groups separated by a `:`. The `id` and `tr` pipe is used to create a list of all the user's groups in a way chpst(8) understands it. Note that we export `$USER` and `$HOME` because some user services may not work without them.

The user can then create new services or symlinks to them in the `/home/<username>/service` directory. To control the services using the sv(8) command, the user can specify the services by path, or by name if the `SVDIR` environment variable is set to the user's services directory. This is shown in the following examples:

```
$ sv status ~/service/*
run: /home/duncan/service/gpg-agent: (pid 901) 33102s
run: /home/duncan/service/ssh-agent: (pid 900) 33102s
$ SVDIR=~/service sv restart gpg-agent
ok: run: gpg-agent: (pid 19818) 0s
```

20 Logging

20.1 Syslog

The default installation comes with no syslog daemon. However, there are syslog implementations available in the Void repositories.

20.1.1 Socklog

socklog(8) is a syslog implementation from the author of runit(8). Use socklog if you're not sure which syslog implementation to use. To use it, install the `socklog-void` package, and enable the `socklog-unix` and `nanoklogd` services.

The logs are saved in sub-directories of `/var/log/socklog/`, and `svlogtail` can be used to help access them conveniently.

The ability to read logs is limited to `root` and users who are part of the `socklog` group.

20.1.2 Other syslog daemons

The Void repositories also include packages for `rsyslog` and `metalog`.

21 rc.conf, rc.local and rc.shutdown

The files `/etc/rc.conf`, `/etc/rc.local` and `/etc/rc.shutdown` can be used to configure certain parts of your Void system. `rc.conf` is often configured by `void-installer`.

21.1 rc.conf

Sourced in runit stages 1 and 3. This file can be used to set variables, including the following:

21.1.1 KEYMAP

Specifies which keymap to use for the Linux console. Available keymaps are listed in `/usr/share/kbd/keymaps`. For example:

```
KEYMAP=fr
```

For further details, refer to loadkeys(1).

21.1.2 HARDWARECLOCK

Specifies whether the hardware clock is set to UTC or local time.

By default this is set to `utc`. However, Windows sets the hardware clock to local time, so if you are dual-booting with Windows, you need to either configure Windows to use UTC, or set this variable to `localtime`.

For further details, refer to hwclock(8).

21.1.3 FONT

Specifies which font to use for the Linux console. Available fonts are listed in `/usr/share/kbd/consolefonts`. For example:

```
FONT=eurlatgr
```

For further details, refer to setfont(1).

21.2 rc.local

Sourced in runit stage 2. A shell script which can be used to specify configuration to be done prior to login.

21.3 rc.shutdown

Sourced in runit stage 3. A shell script which can be used to specify tasks to be done during shutdown.

22 Cron

Void Linux comes without a default cron daemon, you can choose one of multiple cron implementations, by installing the package and enabling the system service.

Available choices include cronie, dcron, fcron and more.

As alternative to the standard cron implementations you can use something like snooze or runwhen which go hand in hand with service supervision.

23 Solid State Drives

Post installation, you will need to enable TRIM for solid state drives. You can check which devices allow TRIM by running:

```
$ lsblk --discard
```

If the DISC-GRAN (discard granularity) and DISC-MAX (discard maximum bytes) columns are non-zero, that means the block device has TRIM support. If your solid state drive partition does not show TRIM support, please verify that you chose a file system with TRIM support (ext4, Btrfs, F2FS, etc.). Note that F2FS requires kernel 4.19 or above to support TRIM.

To run TRIM one-shot, you can run 'fstrim(8)' manually. For example, if your / directory is on an SSD:

```
# fstrim /
```

To automate running TRIM, use cron or add the `discard` option to `/etc/fstab`.

23.1 Periodic TRIM with cron

Add the following lines to `/etc/cron.daily/fstrim`:

```
#!/bin/sh

fstrim /
```

Finally, make the script executable:

```
# chmod u+x /etc/cron.daily/fstrim
```

23.2 Continuous TRIM with fstab discard

Note: You can use either continuous or periodic TRIM, but usage of continuous TRIM is discouraged if you have an SSD that doesn't handle NCQ correctly. Refer to the blacklist.

Edit `/etc/fstab` and add the `discard` option to block devices that need TRIM.

For example, if `/dev/sda1` was an SSD partition, formatted as ext4, and mounted at /:

```
/dev/sda1  /           ext4  defaults,discard   0  1
```

23.3 LVM

To enable TRIM for LVM's commands ('lvremove', `lvreduce`, etc.), open `/etc/lvm/lvm.conf`, uncomment the `issue_discards` option, and set it to `1`:

```
issue_discards=1
```

23.4 LUKS

Warning: Before enabling discard for your LUKS partition, please be aware of the security implications).

To open an encrypted LUKS device and allow discards to pass through, open the device with the `-allow-discards` option:

```
# cryptsetup luksOpen --allow-discards /dev/sdaX luks
```

23.4.1 Non-root devices

Edit `/etc/crypttab` and set the `discard` option for devices on the SSD. For example, if you have a LUKS device with the name `externaldrive1`, device `/dev/sdb2`, and password `none`:

```
externaldrive1  /dev/sdb2   none    luks,discard
```

23.4.2 Root devices

If your root device is on LUKS, add `rd.luks.allow-discards` to `CMDLINE_LINUX_DEFAULT`. In the case of GRUB, edit `/etc/default/grub`:

```
GRUB_CMDLINE_LINUX_DEFAULT="rd.luks.allow-discards"
```

23.4.3 Verifying configuration

To verify that you have configured TRIM correctly for LUKS, run:

```
# dmsetup table /dev/mapper/crypt_dev --showkeys
```

If this command output contains the string `allow_discards`, you have successfully enabled TRIM on your LUKS device.

23.5 ZFS

Before running `trim` on a ZFS pool, ensure that all devices in the pool support it:

```
# zpool get all | grep trim
```

If the pool allows `autotrim` (set `off` by default), you can `trim` the pool periodically or automatically. To one-shot `trim` `yourpoolname`:

```
# zpool trim yourpoolname
```

23.5.1 Periodic TRIM

Add the following lines to `/etc/cron.daily/ztrim`:

```
#!/bin/sh
zpool trim yourpoolname
```

Finally, make the script executable:

```
# chmod u+x /etc/cron.daily/ztrim
```

23.5.2 Autotrim

To set autotrim for `yourpoolname`, run:

```
# zpool set autotrim=on yourpoolname
```

24 Security

There are several ways you can make your installation more secure. This section explores some of them.

24.1 Section Contents

- AppArmor
- Hashboot

25 Hashboot

`hashboot` hashes all files in `/boot` and the MBR to check them during early boot. It is intended for when the root partition is encrypted but not the boot partition. The checksums and a backup of the contents of `/boot` are stored in `/var/lib/hashboot` by default. If a checksum doesn't match, there is the option to restore the file from backup.

If there is a core- or libreboot BIOS, `hashboot` can also check the BIOS for modifications.

25.1 Installation

Install the `hashboot` package. To verify the BIOS, `flashrom` needs to be installed as well.

25.2 Configuration

After installation it is important to run

```
# hashboot index
```

to create the configuration file and generate the index of the chosen options.
If this is not run after installation, next boot will stop with an emergency shell.
Possible options as KEY=VALUE in `/etc/hashboot.cfg`:

- `SAVEDIR` The checksums and the backup are stored here.
- `CKMODES` 001=MBR, 010=files, 100=BIOS. (e.g. 101 to verify MBR and BIOS)
- `MBR_DEVICE` Device with the MBR on it.
- `PROGRAMMER` Use this programmer instead of "internal". Will be passed to flashrom.

25.2.1 Flashrom

For a special programmer for flashrom (e.g. `internal:laptop=force_I_want_a_brick`), the following must be set in `/etc/hashboot.cfg`:

```
PROGRAMMER="internal:laptop=force_I_want_a_brick"
```

25.3 Usage

- Run `hashboot index` to generate checksums and a backup for /boot and MBR
- Run `hashboot check` to check /boot and MBR
- Run `hashboot recover` to replace corrupted files with the backup

26 AppArmor

AppArmor is a mandatory access control mechanism (like SELinux). It can constrain programs based on pre-defined or generated policy definitions.

Void ships with some default profiles for several services, such as `dhcpcd` and `wpa_supplicant`. Container runtimes such as LXC and podman integrate with AppArmor for better security for container payloads.

To use AppArmor on a system, one must:

1. Install the `apparmor` package.
2. Set the `APPARMOR` variable in `/etc/default/apparmor` to `enforce` or `complain`.
3. Set `apparmor=1 security=apparmor` on the kernel commandline.

To accomplish the third step, consult the documentation on how to modify the kernel cmdline.

27 Date and Time

To view your system's current date and time information, as well as make direct changes to it, use date(1).

27.1 Timezone

The default system timezone can be set by linking the timezone file to `/etc/localtime`:

```
# ln -sf /usr/share/zoneinfo/<timezone> /etc/localtime
```

To change the timezone on a per user basis, the `TZ` variable can be exported from your shell's profile:

```
export TZ=<timezone>
```

27.2 Hardware clock

By default, the hardware clock in Void is stored as UTC. Windows does not use UTC by default, and if you are dual-booting, this will conflict with Void. You can either change Windows to use UTC, or change Void Linux to use `localtime` by setting the `HARDWARECLOCK` variable in `/etc/rc.conf`:

```
export HARDWARECLOCK=localtime
```

For more details, see hwclock(8).

27.3 NTP

To maintain accuracy of your system's clock, you can use the Network Time Protocol (NTP).

Void provides packages for three NTP daemons: NTP, OpenNTPD and Chrony.

Once you have installed an NTP daemon, you can enable the service.

27.3.1 NTP

NTP is the official reference implementation of the Network Time Protocol.

The `ntp` package provides NTP and the `isc-ntpd` service.

For further information, visit the NTP site.

27.3.2 OpenNTPD

OpenNTPD focuses on providing a secure, lean NTP implementation which "just works" with reasonable accuracy for a majority of use-cases.

The `openntpd` package provides OpenNTPD and the `openntpd` service.

For further information, visit the OpenNTPD site.

27.3.3 Chrony

Chrony is designed to work well in a variety of conditions; it can synchronize faster and with greater accuracy than NTP.

The `chrony` package provides Chrony and the `chronyd` service.

The Chrony site provides a brief overview of its advantages over NTP, as well as a detailed feature comparison between Chrony, NTP and OpenNTPD.

28 Kernel

28.1 Kernel series

Void Linux provides many kernel series in the default repository. These are named `linuxX.Y`: for example, `linux4.19`. You can query for all available kernel series by running:

```
$ xbps-query --regex -Rs '^linux[0-9.]+-[0-9._]+'
```

The `linux` meta package, installed by default, depends on one of the kernel packages, usually the package containing the latest mainline kernel that works with all DKMS modules. Newer kernels might be available in the repository, but are not necessarily considered stable enough to be the default; use these at your own risk. If you wish to use a more recent kernel and have DKMS modules that you need to build, install the relevant `linuxX.Y-headers` package, then use xbps-reconfigure(1) to reconfigure the `linuxX.Y` package you installed. This will build the DKMS modules.

28.2 Removing old kernels

When updating the kernel, old versions are left behind in case it is necessary to roll back to an older version. Over time, old kernel versions can accumulate, consuming disk space and increasing the time taken by dkms module updates. Furthermore, if `/boot` is a separate partition and fills up with old kernels, updating can fail or result in incomplete initramfs filesystems to be generated and result in kernel panics if they are being booted. Thus, it may be advisable to clean old kernels from time to time.

Removing old kernels is done using the vkpurge(8) utility. `vkpurge` comes pre-installed on every Void Linux system. This utility runs the necessary hooks when removing old kernels. Note that `vkpurge` does not remove kernel *packages*, only particular *kernels*.

28.3 Kernel modules

Kernel modules are typically drivers for devices or filesystems.

28.3.1 Loading kernel modules during boot

Normally the kernel automatically loads required modules, but sometimes it may be necessary to explicitly specify modules to be loaded during boot.

To load kernel modules during boot, a `.conf` file like `/etc/modules-load.d/virtio.conf` needs to be created with the contents:

```
# load virtio-net
virtio-net
```

28.3.2 Blacklisting kernel modules

Blacklisting kernel modules is a method for preventing modules from being loaded by the kernel. There are two different methods for blacklisting kernel modules, one for modules loaded by the initramfs and one for modules loaded after the initramfs process is done. Modules loaded by the initramfs have to be blacklisted in the initramfs configuration.

To blacklist modules loaded after the initramfs process, create a `.conf` file, like `/etc/modprobe.d/radeon.conf`, with the contents:

```
blacklist radeon
```

Blacklisting modules in the initramfs After making the necessary changes to the configuration files, the initramfs needs to be regenerated for the changes to take effect on the next boot.

dracut Dracut can be configured to not include kernel modules through a configuration file. To blacklist modules from being included in a dracut initramfs, create a `.conf` file, like `/etc/dracut.conf.d/radeon.conf`, with the contents:

```
omit_drivers+=" radeon "
```

mkinitcpio To blacklist modules from being included in a mkinitcpio initramfs a `.conf` file needs to be created like `/etc/modprobe.d/radeon.conf` with the contents:

```
blacklist radeon
```

28.4 Kernel hooks

Void Linux provides directories for kernel hooks in `/etc/kernel.d/pre-install,post-install,pre-remove,post-remove`.

These hooks are used to update the boot menus for bootloaders like `grub`, `gummiboot` and `lilo`.

28.4.1 Install hooks

The `pre,post-install` hooks are executed by xbps-reconfigure(1) when configuring a Linux kernel, such as building its initramfs. This happens when a kernel series is installed for the first time or updated, but can also be run manually:

```
# xbps-reconfigure --force linuxX.Y
```

If run manually, they serve to apply initramfs configuration changes to the next boot.

28.4.2 Remove hooks

The `pre,post-remove` hooks are executed by vkpurge(8) when removing old kernels.

28.5 Dynamic Kernel Module Support (dkms)

There are kernel modules that are not part of the Linux source tree that are built at install time using dkms and kernel hooks. The available modules can be listed by searching for `dkms` in the package repositories.

28.6 cmdline

28.6.1 GRUB

Kernel command line arguments can be added through the GRUB bootloader by editing `/etc/default/grub`, changing the `GRUB_CMDLINE_LINUX_DEFAULT` variable and then running `update-grub`.

28.6.2 dracut

Dracut can be configured to add additional command line arguments to the kernel through a configuration file. The documentation for dracut's configuration files can be found in dracut.conf(5). To apply these changes, it is necessary to regenerate the initramfs.

29 Power Management

29.1 acpid

The `acpid` service for acpid(8) is installed and, if Void was installed from a live image using the local source, will be enabled by default. ACPI events are handled by `/etc/acpi/handler.sh`, which uses zzz(8) for suspend-to-RAM events.

29.2 elogind

The `elogind` service is provided by the `elogind` package. By default, elogind(8) listens for, and processes, ACPI events related to lid-switch activation and the *power*, *suspend* and *hibernate* keys. This will conflict with the `acpid` service if it is installed and enabled. Either disable `acpid` when enabling `elogind`, or configure `elogind` to `ignore` ACPI events in logind.conf(5). There are several configuration options, all starting with the keyword `Handle`, that should be set to `ignore` to avoid interfering with `acpid`.

29.3 Power Saving - tlp

Laptop battery life can be extended by using tlp(8). To use it, install the `tlp` package, and enable the `tlp` service. Refer to the TLP documentation for details.

30 Network

Network configuration in Void Linux can be done in several ways. The default installation comes with the dhcpcd(8) service enabled.

30.1 Interface Names

Newer versions of udev(7) no longer use the traditional Linux naming scheme for interfaces (‘eth0‘, `eth0`, `wlan0`, ...).

This behavior can be reverted by adding `net.ifnames=0` to the kernel cmdline.

30.2 Static Configuration

A simple way to configure a static network at boot is to add the necessary ip(8) commands to the `/etc/rc.local` file:

```
ip link set dev eth0 up
ip addr add 192.168.1.2/24 brd + dev eth0
ip route add default via 192.168.1.1
```

30.3 dhcpcd

To run dhcpcd(8) on all interfaces, enable the `dhcpcd` service.

To run `dhcpcd` only on a specific interface, copy the `dhcpcd-eth0` service and modify it to match your interface:

```
$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
   UNKNOWN mode DEFAULT group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
   pfifo_fast state UP mode DEFAULT group default qlen 1000
        link/ether ff:ff:ff:ff:ff:ff brd ff:ff:ff:f
# cp -R /etc/sv/dhcpcd-eth0 /etc/sv/dhcpcd-enp3s0
# sed -i 's/eth0/enp3s0/' /etc/sv/dhcpcd-enp3s0/run
# ln -s /etc/sv/dhcpcd-enp3s0 /var/service/
```

For more information on configuring `dhcpcd`, refer to dhcpcd.conf(5)

30.4 Wireless

Before using wireless networking, use rfkill(8) to check whether the relevant interfaces are soft- or hard-blocked.

Void provides several ways to connect to wireless networks:

- wpa_supplicant
- iwd

- NetworkManager
- ConnMan

31 Firewalls

31.1 iptables

By default, the `iptables` package is installed on the base system. It provides iptables(8)/ip6tables(8). The related services use the `/etc/iptables/iptables.rules` and `/etc/iptables/ip6tables.rules` ruleset files, which must be created by the system administrator.

Two example rulesets are provided in the `/etc/iptables` directory: `empty.rules` and `simple_firewall.rules`.

31.1.1 Applying the rules at boot

To apply iptables rules at runit stage 1, install the `runit-iptables` package. This adds a core-service which restores the `iptables.rules` and `ip6tables.rules` rulesets.

Alternatively, to apply these rules at stage 2, add the following to `/etc/rc.local`:

```
if [ -e /etc/iptables/iptables.rules ]; then
  iptables-restore /etc/iptables/iptables.rules
fi

if [ -e /etc/iptables/ip6tables.rules ]; then
  ip6tables-restore /etc/iptables/ip6tables.rules
fi
```

After rebooting, check the active firewall rules:

```
# iptables -L
# ip6tables -L
```

31.1.2 Applying the rules at runtime

`iptables` comes with two runit services, `iptables` and `ip6tables`, to quickly flush or restore the `iptables.rules` and `ip6tables.rules` rulesets. Once these services are enabled, you can flush the rulesets by downing the relevant service, e.g.:

```
# sv down iptables
```

and restore them by upping the relevant service, e.g.:

```
# sv up ip6tables
```

31.2 nftables

`nftables` replaces `iptables`, `ip6tables`, `arptables` and `ebtables` (collectively referred to as `xtables`). The nftables wiki describes the main differences from the `iptables` toolset.

To use `nftables`, install the `nftables` package, which provides nft(8). It also provides iptables-translate(8)/ip6tables-translate(8) and iptables-restore-translate(8)/ip6tables-restore-translate(8), which convert `iptables` rules to `nftables` rules.

31.2.1 Applying the rules at boot

To apply nftables rules at runit stage 1, install the `runit-nftables` package. This adds a core-service which restores the ruleset in `/etc/nftables.conf`.

31.2.2 Applying the rules at runtime

The `nftables` package provides the `nftables` service, which uses rules from `/etc/nftables.conf`. To load the rules, run:

```
# sv up nftables
```

To flush the rules, run:

```
# sv down nftables
```

32 wpa_supplicant

The `wpa_supplicant` package is installed by default on the base system. It includes utilities to configure wireless interfaces and handle wireless security protocols. To use wpa_supplicant, you will need to enable the wpa_supplicant service.

wpa_supplicant(8) is a daemon that manages wireless interfaces based on wpa_supplicant.conf(5) configuration files. An extensive overview of configuration options, including examples, can be found in `/usr/share/examples/wpa_supplicant/wpa_supplicant.conf`.

wpa_passphrase(8) helps create pre-shared keys for use in configuration files. wpa_cli(8) provides a CLI for managing the `wpa_supplicant` daemon.

32.1 WPA-PSK

To use WPA-PSK, generate a pre-shared key with wpa_passphrase(8) and append the output to the relevant `wpa_supplicant.conf` file:

```
# wpa_passphrase <MYSSID> <passphrase> >> /etc/wpa_supplicant
   /wpa_supplicant-<device_name>.conf
```

32.2 WPA-EAP

WPA-EAP is often used for institutional logins, notably eduroam. This does not use PSK, but a password hash can be generated like this:

```
$ echo -n <passphrase> | iconv -t utf16le | openssl md4
```

32.3 WEP

For WEP configuration, add the following lines to your device's `wpa-supplicant.conf`:

```
network={
    ssid="MYSSID"
    key_mgmt=NONE
    wep_key0="YOUR AP WEP KEY"
    wep_tx_keyidx=0
    auth_alg=SHARED
}
```

32.3.1 The wpa_supplicant service

The `wpa_supplicant` service checks the following options in `/etc/sv/wpa_supplicant/conf`:

- `OPTS`: Options to be passed to the service. Overrides any other options.

- `CONF_FILE`: Path to file to be used for configuration. Defaults to `/etc/wpa_supplicant/wpa_supplicant.conf`.
- `WPA_INTERFACE`: Interface to be matched. May contain a wildcard; defaults to all interfaces.
- `DRIVER`: Driver to use. See `wpa_supplicant -h` for available drivers.

If no `conf` file is found, the service searches for the following files in `/etc/wpa_supplicant`:

- `wpa_supplicant-<interface>.conf`: If found, these files are bound to the named interface.
- `wpa_supplicant.conf`: If found, this file is loaded and binds to all other interfaces found.

Once you are satisfied with your configuration, enable the `wpa_supplicant` service.

32.3.2 Using wpa_cli

When using `wpa_cli` to manage `wpa_supplicant` from the command line, be sure to specify which network interface to use via the `-i` option, e.g.:

```
# wpa_cli -i wlp2s0
```

Not doing so can result in various `wpa_cli` commands (for example, `scan` and `scan_results`) not producing the expected output.

33 IWD

IWD (iNet Wireless Daemon) is a wireless daemon for Linux that aims to replace WPA supplicant.

33.1 Installation

Install the `iwd` package and enable the `dbus` and `iwd` services.

33.2 Usage

The command-line client iwctl(1) can be used to add, remove, and configure network connections. Commands can be passed as arguments; when run without arguments, it provides an interactive session. To list available commands, run `iwctl help`, or run `iwctl` and enter `help` at the interactive prompt.

By default, only the root user and those in the `wheel` group have permission to operate `iwctl`.

33.3 Configuration

Configuration options and examples are described below. Consult the relevant manual pages and the upstream documentation for further information.

33.3.1 Daemon configuration

The main configuration file is located in `/etc/iwd/main.conf`. If it does not exist, you may create it. It is documented in iwd.config(5).

33.3.2 Network configuration

Network configuration, including examples, is documented in iwd.network(5). IWD stores information on known networks, and reads information on pre-provisioned networks from network configuration files located in `/var/lib/iwd`; IWD monitors the directory for changes. Network configuration filenames consist of the encoding of the SSID followed by `.open`, `.psk`, or `.8021x` as determined by the security type.

As an example, a basic configuration file for a WPA2/PSK secured network would be called `<ssid>.psk`, and it would contain the plain text password:

```
[Security]
Passphrase=<password>
```

33.4 Troubleshooting

By default, IWD will create and destroy the wireless interfaces (e.g. `wlan0`) that it manages. This can interfere with `udevd`, which may attempt to rename the interface using its rules for persistent network interface names. The following messages may be printed to your screen as a symptom of this interference:

```
[   39.441723] udevd[1100]: Error changing net interface name
    wlan0 to wlp59s0: Device or resource busy
[   39.442472] udevd[1100]: could not rename interface '3'
   from 'wlan0' to 'wlp59s0': Device or resource busy
```

A simple fix is to prevent IWD from manipulating the network interfaces in this way by adding `UseDefaultInterface=true` to the `[General]` section of `/etc/iwd/main.conf`.

An alternative approach is to disable the use of persistent network interface names by `udevd`. This may be accomplished either by adding `net.ifnames=0` to your kernel cmdline or by creating a symbolic link to `/dev/null` at `/etc/udev/rules.d/80-net-name-slot.rules` to mask the renaming rule. This alternative approach will affect the naming of all network devices.

34 NetworkManager

NetworkManager(8) is a daemon that manages Ethernet, Wi-Fi, and mobile broadband network connections. Install the `NetworkManager` package for the basic NetworkManager utilities.

34.1 Starting NetworkManager

Before enabling the NetworkManager daemon, disable any other network management services, such as dhcpcd, wpa_supplicant, or `wicd`. These services all control network interface configuration, and will interfere with NetworkManager.

Also ensure that the `dbus` service is enabled and running. NetworkManager uses `dbus` to expose networking information and a control interface to clients, and will fail to start without it.

Finally, enable the `NetworkManager` service.

34.2 Configuring NetworkManager

Users of NetworkManager must belong to the `network` group.

The `NetworkManager` package includes a command line tool, nmcli(1), and a text-based user interface, nmtui(1), to control network settings.

There are many other front-ends to NetworkManager, including `nm-applet` for system trays, `nm-plasma` for KDE Plasma, and a built-in network configuration tool in GNOME.

34.3 Eduroam with NetworkManager

Eduroam is a roaming service providing international, secure Internet access at universities and other academic institutions. More information can be found here.

34.3.1 Dependencies

Install the `python3-dbus` package.

34.3.2 Installation

Download the correct eduroam_cat installer for your institution from here and execute it. It will provide a user interface guiding you through the process.

35 ConnMan

ConnMan(8) is a daemon that manages network connections, is designed to be slim and to use as few resources as possible. The `connman` package contains the basic utilities to run ConnMan.

35.1 Starting ConnMan

To enable the ConnMan daemon, first disable any other network managing services like dhcpcd, wpa_supplicant, or `wicd`. These services all control network interface configuration, and interfere with each other.

Finally, enable the `connmand` service.

35.2 Configuring ConnMan

The `connman` package includes a command line tool, connmanctl(1) to control network settings. If you do not provide any commands, `connmanctl` starts as an interactive shell.

There are many other front-ends to ConnMan, including `connman-ui` for system trays, `connman-gtk` for GTK, `cmst` for QT and `connman-ncurses` for ncurses based UI.

35.3 Preventing DNS overrides by ConnMan

Create `/etc/sv/connmand/conf` with the following content:

```
OPTS="--nodnsproxy"
```

36 Network Filesystems

36.1 NFS

36.1.1 Mounting an NFS Share

To mount an NFS share, start by installing the `nfs-utils` and `sv-netmount` packages.

Before mounting an NFS share, enable the `statd`, `rpcbind`, and `netmount` services. If the server supports `nfs4`, the `statd` service isn't necessary.

To mount an NFS share:

```
# mount -t <mount_type> <host>:/path/to/sourcedir /path/to/
   destdir
```

`<mount_type>` should be `nfs4` if the server supports it, or `nfs` otherwise. `<host>` can be either the hostname or IP address of the server.

Mounting options can be found in mount.nfs(8), while unmounting options can be found in umount.nfs(8).

For example, to connect `/volume` on a server at `192.168.1.99` to an existing `/mnt/volume` directory on your local system:

```
# mount -t nfs 192.168.1.99:/volume /mnt/volume
```

To have the directory mounted when the system boots, add an entry to fstab(5):

```
192.168.1.99:/volume /mnt/volume nfs rw,hard,intr 0 0
```

Refer to nfs(5) for information about the available mounting options.

36.1.2 Setting up a server (NFSv4, Kerberos disabled)

To run an NFS server, start by installing the `nfs-utils` package.

Edit `/etc/exports` to add a shared volume:

```
/storage/foo     *.local(rw,no_subtree_check,no_root_squash)
```

This line exports the `/storage/foo` directory to any host in the local domain, with read/write access. For information about the `no_subtree_check` and `no_root_squash` options, and available options more generally, refer to exports(5).

Finally, enable the `rpcbind`, `statd`, and `nfs-server` services.

This will start your NFS server. To check if the shares are working, use the showmount(8) utility to check the NFS server status:

```
# showmount -e localhost
```

You can use nfs.conf(5) to configure your server.

37 Session and Seat Management

Session and seat management is not necessary for every setup, but it can be used to safely create temporary runtime directories, provide access to hardware devices and multi-seat capabilities, and control system shutdown.

37.1 D-Bus

D-Bus is an IPC (inter-process communication) mechanism used by userspace software in Linux. D-Bus can provide a system bus and/or a session bus, the latter being specific to a user session.

- To provide a system bus, you should enable the `dbus` service. This might require a system reboot to work properly.
- To provide a session bus, you can start a given program (usually a window manager or interactive shell) with dbus-run-session(1). Most desktop environments, if launched through an adequate display manager, will launch a D-Bus session themselves.

Note that some software assumes the presence of a system bus, while other software assumes the presence of a session bus.

37.2 elogind

elogind(8) manages user logins and system power, as a standalone version of `systemd-logind`. elogind provides necessary features for most desktop environments and Wayland compositors. It can also be one of the mechanisms for rootless Xorg.

Please read the "Power Management" section for things to consider before installing elogind.

To make use of its features, install the `elogind` package and make sure the system D-Bus is enabled. You might need to log out and in again.

If you're having any issues with elogind, enable its service, as waiting for a D-Bus activation can lead to issues.

There is an alternative D-Bus configuration which takes advantage of elogind for features such as seat detection. It requires installing the `dbus-elogind`, `dbus-elogind-libs` and `dbus-elogind-x11` packages.

38 Graphical Session

In order to configure a graphical session, you need:

- Graphics drivers
- A basis for your graphical session: Xorg or Wayland

You may also need:

- Session management tools

39 Graphics Drivers

This section covers basic graphics setup depending on the hardware configuration of your system.

39.1 Section Contents

- Intel GPU
- NVIDIA Optimus
- NVIDIA

40 Intel GPU

Intel GPU support requires the `linux-firmware-intel` package. If you have installed the `linux` or `linux-lts` packages, it will be installed as a dependency. If you installed a version-specific kernel package (e.g., `linux5.4`), it may be necessary to manually install `linux-firmware-intel`.

40.1 OpenGL

Install the Mesa DRI package, `mesa-dri`.

> Note: This is already included in the `xorg` meta-package, but it is needed
>
> when installing xorg via `xorg-minimal` or for running a Wayland compositor.

40.2 Vulkan

Install the Khronos Vulkan Loader and the Mesa Intel Vulkan driver packages, respectively `vulkan-loader` and `mesa-vulkan-intel`.

40.3 Video acceleration

Install the `intel-video-accel` meta-package:

This will install all the Intel VA-API drivers. `intel-media-driver` will be used by default, but this choice can be overridden at runtime via the environment variable `LIBVA_DRIVER_NAME`:

Driver Package	Supported GPU Gen	Explicit selection
`libva-intel-driver`	up to Coffee Lake	`LIBVA_DRIVER_NAME=i965`
`intel-media-driver`	from Broadwell	`LIBVA_DRIVER_NAME=iHD`

40.4 Troubleshooting

The kernels packaged by Void are configured with `CONFIG_INTEL_IOMMU_DEFAULT_ON=y`, which can lead to issues with their graphics drivers, as reported by the kernel documentation. To fix this, it is necessary to disable IOMMU for the integrated GPU. This can be done by adding `intel_iommu=igfx_off` to your kernel cmdline. This problem is expected to happen on the Broadwell generation of internal GPUs. If you have another internal GPU and your issues are fixed by this kernel option, you should file a bug reporting the problem to kernel developers.

41 NVIDIA Optimus

NVIDIA Optimus refers to a dual graphics configuration found on laptops consisting of an Intel integrated GPU and a discrete NVIDIA GPU.

There are different methods to take advantage of the NVIDIA GPU, which depend on the driver version supported by your hardware.

In order to determine the correct driver to install, it is not enough to look at the "Supported Products" list on NVIDIA's website, because they are not guaranteed to work in an Optimus configuration. So the only way is to try installing the latest `nvidia`, rebooting, and looking at the kernel log. If your device is not supported, you will see a message like this:

```
NVRM: The NVIDIA GPU xxxx:xx:xx.x (PCI ID: xxxx:xxxx)
NVRM: installed in this system is not supported by the xxx.xx
NVRM: NVIDIA Linux driver release.  Please see 'Appendix
NVRM: A - Supported NVIDIA GPU Products' in this release's
NVRM: README, available on the Linux driver download page
NVRM: at www.nvidia.com.
```

which means you have to uninstall `nvidia` and install the legacy `nvidia390`.
A summary of the methods supported by Void, which are mutually exclusive:
PRIME Render Offload

- only available on `nvidia`
- allows to switch to the NVIDIA GPU on a per-application basis
- more flexible but power saving capabilities depend on the hardware (pre-Turing devices are not shut down completely)

Offloading Graphics Display with RandR 1.4

- available on `nvidia` and `nvidia390`
- allows to choose which GPU to use at the start of the X session
- less flexible, but allows the user to completely shut down the NVIDIA GPU when not in use, thus saving power

Bumblebee

- available on `nvidia` and `nvidia390`
- allows to switch to the NVIDIA GPU on a per-application basis
- unofficial method, offers poor performance

Nouveau PRIME

- uses the open source driver `nouveau`
- allows to switch to the NVIDIA GPU on a per-application basis
- `nouveau` is a reverse-engineered driver and offers poor performance

You can check the currently used GPU by searching for `renderer string` in the output of the `glxinfo` command. It is necessary to install the `glxinfo` package for this.

41.1 PRIME Render Offload

In this method, GPU switching is done by setting environment variables when executing the application to be rendered on the NVIDIA GPU. The wrapper script `prime-run` is available from the `nvidia` package, and can be used as shown below:

```
$ prime-run <application>
```

For more information, see NVIDIA's README

41.2 Bumblebee

Enable the `bumblebeed` service and add the user to the `bumblebee` group. This requires a re-login to take effect.

Run the application to be rendered on the NVIDIA GPU with `optirun`:

```
$ optirun <application>
```

41.3 Nouveau PRIME

This method uses the open source `nouveau` driver. If the NVIDIA drivers are installed, it is necessary to configure the system to use `nouveau`.

Set `DRI_PRIME=1` to run an application on the NVIDIA GPU:

```
$ DRI_PRIME=1 <application>
```

42 NVIDIA

42.1 nouveau (Open Source Driver)

This driver is developed mostly by the community, with little input from Nvidia, and is not as performant as the proprietary driver. It is required in order to run most Wayland compositors.

Install the `mesa-dri` driver or the `xf86-video-nouveau` driver.

Xorg can make use of either of the above mentioned drivers. The latter is older, more stable and generally the recommended option. However, for newer devices you might get better performance by using the `mesa-dri` provided driver.

Note: `xf86-video-nouveau` is already included in the `xorg` meta-package, but is needed when installing via `xorg-minimal`.

For using Wayland, users should install the `mesa-dri` provided driver.

42.2 nvidia (Proprietary Driver)

The proprietary drivers are available in the nonfree repository.

Check if your graphics card belongs to the legacy branch. If it does not, install the `nvidia` package. Otherwise you should install the appropriate legacy driver, either `nvidia390` or `nvidia340`.

Brand	Type	Model	Driver Package
NVIDIA	Proprietary	500+	`nvidia`
NVIDIA	Proprietary	300/400 Series	`nvidia390`
NVIDIA	Proprietary	GeForce8/9 + 100/200/300 Series	`nvidia340`

The proprietary driver integrates in the kernel through DKMS.

This driver offers better performance and power handling, and is recommended where performance is needed.

42.3 32-bit program support (glibc only)

In order to run 32-bit programs with driver support, you need to install additional packages.

If using the `nouveau` driver, install the `mesa-dri-32bit` package.

If using the `nvidia` driver, install the `nvidia<x>-libs-32bit` package. `<x>` represents the legacy driver version (340 or 390) or can be left empty for the main driver.

42.4 Reverting from nvidia to nouveau

42.4.1 Uninstalling nvidia

In order to revert to the `nouveau` driver, install the 'nouveau' driver (if it was not installed already), then remove the `nvidia`, `nvidia390` or `nvidia340` package, as appropriate.

If you were using the `nvidia340` driver, you will need to install the `libglvnd` package after removing the `nvidia340` package.

42.4.2 Keeping both drivers

It is possible to use the `nouveau` driver while still having the `nvidia` driver installed. To do so, remove the blacklisting of `nouveau` in `/etc/modprobe.d/nouveau_blacklist.conf`, `/usr/lib/modprobe.d/nvidia.conf`, or `/usr/lib/modprobe.d/nvidia-dkms.conf` by commenting it out:

```
#blacklist nouveau
```

For Xorg, specify that it should load the `nouveau` driver rather than the `nvidia` driver by creating the file `/etc/X11/xorg.conf.d/20-nouveau.conf` with the following content:

```
Section "Device"
    Identifier "Nvidia card"
    Driver "nouveau"
EndSection
```

You may need to reboot your system for these changes to take effect.

43 Xorg

This section details the manual installation and configuration of the Xorg display server and common related services and utilities. If you would just like to install a full desktop environment, it is recommended to try one of the flavor images

43.1 Installation

Void provides a comprehensive `xorg` package which installs the server and all of the free video drivers, input drivers, fonts, and base applications. This package is a safe option, and should be adequate for most systems which don't require proprietary video drivers.

If you would like to select only the packages you need, the `xorg-minimal` package contains the base xorg server *only*. If you install only `xorg-minimal`, you will likely need to install a font package (like `xorg-fonts`), a terminal emulator (like `xterm`), and a window manager to have a usable graphics system.

43.2 Video Drivers

Void provides both open-source and proprietary (non-free) video drivers.

43.2.1 Open Source Drivers

Xorg can use two categories of open source drivers: DDX or modesetting.

DDX The DDX drivers are installed with the `xorg` package by default, or may be installed individually if the `xorg-minimal` package was installed. They are provided by the `xf86-video-*` packages.

For advanced configuration, see the man page corresponding to the vendor name, like intel(4).

Modesetting Modesetting requires the `mesa-dri` package, and no additional vendor-specific driver package.

Xorg defaults to DDX drivers if they are present, so in this case modesetting must be explicitly selected: see Forcing the modesetting driver.

For advanced configuration, see modesetting(4).

43.2.2 Proprietary Drivers

Void also provides proprietary NVIDIA drivers, which are available in the nonfree repository.

43.3 Input Drivers

A number of input drivers are available for Xorg. If `xorg-minimal` was installed and a device is not responding, or behaving unexpectedly, a different driver may correct the issue. These drivers can grab everything from power buttons to mice and keyboards. They are provided by the `xf86-input-*` packages.

43.4 Xorg Configuration

Although Xorg normally auto-detects drivers and configuration is not needed, a config for a specific keyboard driver may look something like a file `/etc/X11/xorg.conf.d/30-keyboard.conf` with the contents:

```
Section "InputClass"
  Identifier "keyboard-all"
  Driver "evdev"
  MatchIsKeyboard "on"
EndSection
```

43.4.1 Forcing the modesetting driver

Create the file `/etc/X11/xorg.conf.d/10-modesetting.conf`:

```
Section "Device"
    Identifier "GPU0"
    Driver "modesetting"
EndSection
```

and restart Xorg. Verify that the configuration has been picked up with:

```
$ grep -m1 '(II) modeset([0-9]+):' /var/log/Xorg.0.log
```

If there is a match, modesetting is being used.

43.5 Starting X Sessions

43.5.1 startx

The `xinit` package provides the startx(1) script as a frontend to xinit(1), which can be used to start X sessions from the console. For example, to use i3, edit `~/.xinitrc` to contain:

```
exec /bin/i3
```

Then call `startx` to start an i3 session.
If a D-Bus session bus is required, you can manually start one.

43.5.2 Display Managers

Display managers (DMs) provide a graphical login UI. A number of DMs are available in the Void repositories, including `gdm` (the GNOME DM), `sddm` (the KDE DM) and `lightdm`. When setting up a display manager, be sure to test the service before enabling it.

44 Wayland

This section details the manual installation and configuration of Wayland compositors and related services and utilities.

44.1 Installation

Unlike Xorg, Wayland implementations combine the display server, the window manager and the compositor in a single application.

44.1.1 Desktop Environments

GNOME, KDE Plasma and Enlightenment have Wayland sessions. GNOME uses its Wayland session by default. When using these desktop environments, applications built with GTK+ will automatically choose the Wayland backend, while Qt5 and EFL applications might require setting some environment variables if used outside KDE or Enlightenment, respectively.

44.1.2 Standalone compositors

Void Linux currently packages the following Wayland compositors:

- Weston: reference compositor for Wayland
- Sway: an i3-compatible Wayland compositor
- Wayfire: 3D Wayland compositor
- Hikari: a stacking compositor with some tiling features
- Cage: a Wayland kiosk

44.1.3 Video drivers

Both GNOME and KDE Plasma have EGLStreams backends for Wayland, which means they can use the proprietary NVIDIA drivers. Most other Wayland compositors require drivers that implement the GBM interface. The main driver for this purpose is provided by the `mesa-dri` package. The "Graphics Drivers" section has more details regarding setting up graphics in different systems.

44.1.4 Native applications

Qt5-based applications require installing the `qt5-wayland` package and setting the environment variable `QT_QPA_PLATFORM=wayland-egl` to enable their Wayland backend. Some KDE specific applications also require installing the `kwayland` package. EFL-based applications require setting the environment variable `ELM_DISPLAY=wl`, and can have issues without it, due to not supporting XWayland properly. GTK+-based applications should use the Wayland backend automatically. Information about other toolkits can be found in the Wayland documentation.

Media applications, such as mpv(1), vlc(1) and `imv` work natively on Wayland.

Web browsers Mozilla Firefox ships with a Wayland backend which is disabled by default. To enable the Wayland backend, either set the environment variable `MOZ_ENABLE_WAYLAND=1` before running `firefox` or use the provided `firefox-wayland` script.

Browsers based on GTK+ or Qt5, such as Midori and qutebrowser(1), should work on Wayland natively.

Running X applications inside Wayland If an application doesn't support Wayland, it can still be used in a Wayland context. XWayland is an X server that bridges this gap for most Wayland compositors, and is installed as a dependency for most of them. Its package is `xorg-server-xwayland`. For Weston, the correct package is `weston-xwayland`.

44.2 Configuration

The Wayland API uses the `XDG_RUNTIME_DIR` environment variable to determine the directory for the Wayland socket.

Install `elogind` as your session manager to automatically setup `XDG_RUNTIME_DIR`.

Alternatively, manually set the environment variable through the shell. Make sure to create a dedicated user directory and set its permissions to `700`. A good default location is `/run/user/$(id -u)`.

It is also possible that some applications use the `XDG_SESSION_TYPE` environment variable in some way, which requires that you set it to `wayland`.

45 Fonts

To customize font display in your graphical session, you can use configurations provided in `/usr/share/fontconfig/conf.avail/`. To do so, create a symlink to the relevant `.conf` file in `/etc/fonts/conf.d/`, then use xbps-reconfigure(1) to reconfigure the `fontconfig` package.

For example, to disable use of bitmap fonts:

```
# ln -s /usr/share/fontconfig/conf.avail/70-no-bitmaps.conf /
   etc/fonts/conf.d/
# xbps-reconfigure -f fontconfig
```

46 Icons

46.1 GTK

By default, GTK-based applications try to use the Adwaita icon theme for application icons. Consequently, installation of the `gtk+3` package will also install the `adwaita-icon-theme` package. If you wish to use a different theme, install the relevant package, then specify the theme in `/etc/gtk-3.0/settings.ini` or `~/.config/gtk-3.0/settings.ini`. `adwaita-icon-theme` can be removed after ignoring the package.

For information about how to specify a different GTK icon theme in `settings.ini`, refer to the GtkSettings documentation, in particular the "gtk-icon-theme-name" property.

47 GNOME

47.1 Pre-installation

Install the `dbus` package, ensure the `dbus` service is enabled, and reboot for the changes to take effect.

47.2 Installation

Install the `gnome` package for a GNOME environment which includes GNOME applications.

A minimal GNOME environment can be created by installing the `mesa-dri`, `gnome-session`, `gdm` and `adwaita-icon-theme` packages. (Note, however, that not all GNOME features may be present or functional.)

The `gdm` package provides the `gdm` service for the GNOME Display Manager; test the service before enabling it. GDM defaults to providing a Wayland session via the `mutter` window manager, but an X session can be chosen instead.

If you wish to start an X-based GNOME session from the console, use startx to run `gnome-session`.

GNOME applications can be installed via the `gnome-apps` package.

If you require ZeroConf support, install the `avahi` package and enable the `avahi-daemon` service.

48 KDE

48.1 Installation

Install the `kde5` package, and optionally, the `kde5-baseapps` package.

To use the "Networks" widget, enable the `dbus` and `NetworkManager` services.

Installing the `kde5` package also installs the `sddm` package, which provides the `sddm` service for the Simple Desktop Display Manager; test the service before enabling it. If you are not intending to run SDDM via a remote X server, you will need to install either the `xorg-minimal` package or the `xorg` package. By default, SDDM will start an X-based Plasma session, but you can request a Wayland-based Plasma session instead.

If you wish to start an X-based session from the console, use startx to run `startplasma-x11`. For a Wayland-based session, run `startplasma-wayland` directly.

49 Multimedia

49.1 Audio setup

To setup audio on your Void Linux system you have to decide if you want to use PulseAudio or just ALSA.

Some applications require PulseAudio, especially closed source programs.

50 ALSA

To use ALSA, install the `alsa-utils` package and make sure your user is a member of the `audio` group.

The `alsa-utils` package provides the `alsa` service. When enabled, this service saves and restores the state of ALSA (e.g. volume) at shutdown and boot, respectively.

To allow use of software requiring PulseAudio, install the `apulse` package. `apulse` provides part of the PulseAudio interface expected by applications, translating calls to that interface into calls to ALSA. For details about using `apulse`, consult the project README.

50.1 Configuration

The default sound card can be specified via ALSA configuration files or via kernel module options.

To obtain information about the order of loaded sound card modules:

```
$ cat /proc/asound/modules
 0 snd_hda_intel
 1 snd_hda_intel
 2 snd_usb_audio
```

To set a different card as the default, edit `/etc/asound.conf` or the per-user configuration file `~/.asoundrc`:

```
defaults.ctl.card 2;
defaults.pcm.card 2;
```

or specify sound card module order in `/etc/modprobe.d/alsa.conf`:

```
options snd_usb_audio index=0
```

50.2 Dmix

The `dmix` ALSA plugin allows playing sound from multiple sources. `dmix` is enabled by default for soundcards which do not support hardware mixing. To enable it for digital output, edit `/etc/asound.conf`:

```
pcm.dsp {
    type plug
    slave.pcm "dmix"
}
```

51 PulseAudio

Depending on which applications you use, you might need to provide PulseAudio with a D-BUS session bus (e.g. via `dbus-run-session`) or a D-BUS system bus (via the `dbus` service).

For applications which use ALSA directly and don't support PulseAudio, the `alsa-plugins-pulseaudio` package can make them use PulseAudio through ALSA.

The PulseAudio package comes with a service file, which is not necessary in most setups - the PulseAudio maintainers discourage using a system-wide setup. Instead, PulseAudio will automatically start when needed.

There are several methods of allowing PulseAudio to access to audio devices. The simplest one is to add your user to the `audio` group. Alternatively, you can use a session manager, like `elogind`.

52 sndio

Install the `sndio` package and enable the sndiod(8) service.

52.1 Configuration

The service can be configured by adding sndiod(8) flags to the `OPTS` variable in the service configuration file (‘/etc/sv/sndiod/conf‘).

52.1.1 Default device

sndiod(8) uses the first ALSA device by default. To use another ALSA device for sndio's default device `snd/0` add the flags to use specific devices to the service configuration file.

```
# echo 'OPTS="-f rsnd/Speaker"' >/etc/sv/sndiod/conf
```

Use the `-f` flag to chooses a device by its ALSA device index or its ALSA device name.

52.2 Volume control

The master and per application volume controls are controlled with MIDI messages by hardware or software.

aucatctl(1) is a tool specific to sndio to send MIDI control messages to the sndiod(8) daemon. It can be found in the `aucatctl` package.

52.3 Application specific configurations

52.3.1 Firefox

Firefox is built with sndio support and should work out of the box since version 71 if libsndio is installed and the `snd/0` device is available.

The following `about:config` changes are required for versions prior to 71 and should be removed when using version 71 or later:

```
media.cubeb.backend;sndio
media.cubeb.sandbox;false
security.sandbox.content.read_path_whitelist;/home/<username
   >/.sndio/cookie
security.sandbox.content.write_path_whitelist;/home/<username
   >/.sndio/cookie
```

52.3.2 mpv

MPV comes with sndio support, but to prevent it from using ALSA over sndio if the ALSA device is available, set the `-ao=sndio` command line option. You can also add the option to mpv's configuration file: `~/.config/mpv/mpv.conf` should contain a line specifying `ao=sndio`.

52.3.3 OpenAL

libopenal comes with sndio support, but prioritizes ALSA over sndio by default. You can configure this behaviour per user in `~/.alsoftrc` or system wide in `/etc/openal/alsoft.conf` by adding the following lines:

```
[general]
drivers = sndio
```

53 Bluetooth

Ensure the Bluetooth controller is not blocked. Use `rfkill` to check whether there are any blocks and to remove soft blocks. If there is a hard block, there is likely either a physical hardware switch or an option in the BIOS to enable the Bluetooth controller.

```
$ rfkill
ID TYPE      DEVICE      SOFT      HARD
0 wlan       phy0   unblocked unblocked
1 bluetooth hci0      blocked unblocked

# rfkill unblock bluetooth
```

53.1 Installation

Install the `bluez` package and enable the `bluetoothd` and `dbus` services. Then, add your user to the `bluetooth` group and restart the `dbus` service, or simply reboot the system. Note that restarting the `dbus` service may kill processes making use of it.

> Note: To use an audio device such as a wireless speaker or headset, ALSA users
>
> need to install the `bluez-alsa` package, while
>
> PulseAudio users do not need any additional software.

53.2 Usage

Manage Bluetooth connections and controllers using `bluetoothctl`, which provides a command line interface and also accepts commands on standard input.

Consult the Arch Wiki for an example of how to pair a device.

53.3 Configuration

The main configuration file is `/etc/bluetooth/main.conf`.

54 TeX Live

In Void, the `texlive-bin` package provides a basic TeX installation, including the `tlmgr` program. Use `tlmgr` to install TeX packages and package collections from CTAN mirrors. Install the `gnupg` package to allow `tlmgr` to verify TeX packages.

The `texlive-bin` package contains the latest TeX Live version; however, earlier versions, such as `texlive2018-bin`, are also available.

54.1 Configuring TeX Live

After installing TeX Live, update the value of `PATH`:

```
$ source /etc/profile
```

Check that `/opt/texlive/<year>/bin/x86_64-linux` (or `/opt/texlive/<year>/bin/i386-linux`) is in your `PATH`:

```
$ echo $PATH
```

If required, change the global default paper size:

```
# tlmgr paper <letter|a4>
```

54.2 Installing/Updating TeX packages

To install all available packages:

```
# tlmgr install scheme-full
```

To install specific packages, you can install the collection(s) including them. To list the available collections:

```
$ tlmgr info collections
```

To see the list of files owned by a collection:

```
$ tlmgr info --list collection-<name>
```

To install the collection:

```
# tlmgr install collection-<name>
```

To install a standalone package, first check if the package exists:

```
$ tlmgr search --global <package>
```

and then install it:

```
# tlmgr install <package>
```

To find the package providing a particular file (for example, a font):

```
$ tlmgr search --file <filename> --global
```

To remove a package or a collection:

```
# tlmgr remove <package>
```

To update installed packages:

```
# tlmgr update --all
```

For a full description, check:
https://www.tug.org/texlive/doc/tlmgr.html

55 External Applications

55.1 Programming Languages

The Void repositories have a number of Python and Lua packages. If possible, install packages from the Void repositories or consider packaging the library or application you need. Packaging your application allows for easier system maintenance and can benefit other Void Linux users, so consider making a pull request for it. The contribution instructions can be found here.

To keep packages smaller, Void has separate `devel` packages for header files and development tools. If you install a library or application via a language's package manager (e.g. `pip`, `gem`), or compile one from source, you may need to install the programming language's `-devel` package. This is specially relevant for `musl` libc users, due to pre-built binaries usually targeting `glibc` instead.

Language	Package Manager	Void Package
Python3	pip, anaconda, virtualenv, etc	`python3-devel`
Python2	pip, anaconda, virtualenv, etc	`python2-devel`
Ruby	gem	`ruby-devel`
lua	luarocks	`lua-devel`

55.2 Restricted Packages

Some packages have legal restrictions on their distribution (e.g. Discord), may be too large, or have another condition that makes it difficult for Void to distribute. These packages have build templates, but the packages themselves are not built or distributed. As such, they must be built locally. For more information see the page on restricted packages.

55.3 Non-x86_64 Arch

The Void build system runs on x86_64 servers, both for compiling and cross compiling packages. However, some packages (e.g. `libreoffice`) do not support cross-compilation. These packages have to be built locally on a computer running the same architecture and libc as the system on which the package is to be used. To learn how to build packages, refer to the README for the void-packages repository.

55.4 Flatpak

Flatpak is another method for installing external proprietary applications on Linux. For information on using Flatpak with Void Linux, see the official Flatpak documentation.

> Flatpak's sandboxing will not necessarily protect you from any security and/or
>
> privacy-violating features of proprietary software.

55.4.1 Troubleshooting

Some apps may not function properly (e.g. not being able to access the host system's files). Some of these issues can be fixed by installing one or more of the `xdg-desktop-portal`, `xdg-desktop-portal-gtk`, `xdg-user-dirs`, `xdg-user-dirs-gtk` or `xdg-utils` packages.

Some Flatpaks require D-Bus and/or Pulseaudio.

55.5 Octave Packages

Some Octave packages require external dependencies to compile and run. For example, to build the control package, you must install the `openblas-devel`, `libgomp-devel`, `libgfortran-devel`, `gcc-fortran`, and `gcc` packages.

55.6 MATLAB

To use MATLAB's help browser, live scripts, add-on installer, and simulink, install the `libselinux` package.

56 Printing

CUPS (Common Unix Printing System) is the supported mechanism for connecting to printers on Void Linux.

As prerequisites, install the `cups` package and enable the `cupsd` service. Wait until the service is marked available.

56.1 Installing Printing Drivers

If the printer is being accessed over the network and supports PostScript or PCL, CUPS alone should be sufficient. However, additional driver packages are necessary for local printer support. The `cups-filters` package provides driver support for CUPS.

Depending on the hardware in question, additional drivers may be necessary.

Some CUPS drivers contain proprietary or binary-only extensions. These are available only in the nonfree repository, and sometimes only for specific architectures.

56.1.1 Gutenprint drivers

Gutenprint provides support for many printers. These drivers are contained in the `gutenprint` package.

56.1.2 HP drivers

Printers from Hewlett-Packard require the `hplip` package.

Running the following command will guide you through the driver installation process. The default configuration selections it suggests are typically sufficient.

```
# hp-setup -i
```

56.1.3 Brother drivers

For Brother printer support, install the foomatic drivers, which are contained in the `foomatic-db` and `foomatic-db-nonfree` packages.

56.2 Configuring a New Printer

CUPS provides a web interface and command line tools that can be used to configure printers. Additionally, various native GUI options are available and may be better suited, depending on the use-case.

56.2.1 Web interface

To configure the printer using the CUPS web interface, navigate to http://localhost:631 in a browser. Under the "Administration" tab, select "Printers > Add Printer".

56.2.2 Command line

The lpadmin(8) tool may be used to configure a printer using the command line.

56.2.3 Graphical interface

The `system-config-printer` package offers simple and robust configuration of new printers. Install and invoke it:

```
# system-config-printer
```

Normally this tool requires root privileges. However, if you are using PolicyKit, you can install the `cups-pk-helper` package to allow unprivileged users to use `system-config-printer`.

While `system-config-printer` is shown here, your desktop environment may have a native printer dialog, which may be found by consulting the documentation for your DE.

56.3 Troubleshooting

56.3.1 USB printer not shown

The device URI can be found manually by running:

```
# /usr/lib/cups/backend/usb
```

57 Manual Pages

Void packages come with manual pages and the default installation includes the mandoc manpage toolset.

The man(1) command can be used to show manual pages.

```
$ man 1 chroot
```

The `mandoc` toolset contains apropos(1), which can be used to search for manual pages. `apropos` uses a database that can be updated and generated with the makewhatis(8) command.

```
# makewhatis -a
$ apropos chroot
chroot(1) - run command or interactive shell with special
   root directory
xbps-uchroot(1) - XBPS utility to chroot and bind mount with
   Linux namespaces
xbps-uchroot(1) - XBPS utility to chroot and bind mount with
   Linux namespaces
xbps-uunshare(1) - XBPS utility to chroot and bind mount with
    Linux user namespaces
xbps-uunshare(1) - XBPS utility to chroot and bind mount with
    Linux user namespaces
chroot(2) - change root directory
```

`man-pages-devel` and `man-pages-posix` are extra packages which are not installed by default. They contain development and POSIX manuals, respectively.

58 XBPS Package Manager

The X Binary Package System (XBPS) is a fast package manager that has been designed and implemented from scratch. XBPS is managed by the Void Linux team and developed at https://github.com/void-linux/xbps.

Most general package management is done with the following commands:

- xbps-query(1) searches for and displays information about packages installed locally, or, if used with the `-R` flag, packages contained in repositories.
- xbps-install(1) installs and updates packages, and syncs repository indexes.
- xbps-remove(1) removes installed packages, and can also remove orphaned packages and cached package files.
- xbps-reconfigure(1) runs the configuration steps for installed packages, and can be used to reconfigure certain packages after changes in their configuration files. The latter usually requires the `-force` flag.
- xbps-alternatives(1) lists or sets the alternatives provided by installed packages. Alternatives is a system which allows multiple packages to provide common functionality through otherwise conflicting files, by creating symlinks from the common paths to package-specific versions that are selected by the user.
- xbps-pkgdb(1) can report and fix issues in the package database, as well as modify it.
- xbps-rindex(1) manages local binary package repositories.

Most questions can be answered by consulting the man pages for these tools, together with the xbps.d(5) man page.

To learn how to build packages from source, refer to the README for the void-packages repository.

58.1 Updating

Like any other system, it is important to keep Void up-to-date. Use xbps-install(1) to update:

```
# xbps-install -Su
```

XBPS must use a separate transaction to update itself. If your update includes the `xbps` package, you will need to run the above command a second time to apply the rest of the updates.

58.1.1 Restarting Services

XBPS does not restart services when they are updated. This task is left to the administrator, so they can orchestrate maintenance windows, ensure reasonable backup capacity, and generally be present for service upgrades.

To find processes running different versions than are present on disk, use the `xcheckrestart` tool provided by the `xtools` package:

```
$ xcheckrestart
11339 /opt/google/chrome/chrome (deleted) (google-chrome)
```

`xcheckrestart` will print out the PID, path to the executable, status of the path that was launched (almost always `deleted`) and the process name. `xcheckrestart` can and should be run as an unprivileged user.

58.1.2 Kernel Panic After Update

If you get a kernel panic after an update, it is likely your system ran out of space in `/boot`. Refer to "Removing old kernels" for further information.

58.2 Finding Files and Packages

If you can't find a file or program you expected to find after installing a package, you can use xbps-query(1) to list the files provided by that package:

```
$ xbps-query -f <package_name>
```

The `xtools` package contains the xlocate(1) utility. `xlocate` works like locate(1), but for files in the Void package repositories:

```
$ xlocate -S
Fetching objects: 11688, done.
From https://alpha.de.repo.voidlinux.org/xlocate/xlocate
 + e122c3634...a2659176f master     -> master  (forced update
    )
$ xlocate xlocate
xtools-0.59_1   /usr/bin/xlocate
xtools-0.59_1   /usr/share/man/man1/xlocate.1 -> /usr/share/
   man/man1/xtools.1
```

It is also possible to use xbps-query(1) to find files, though this is strongly discouraged:

```
$ xbps-query -Ro /usr/bin/xlocate
xtools-0.46_1: /usr/bin/xlocate (regular file)
```

This requires `xbps-query` to download parts of every package to find the file. `xlocate`, however, queries a locally cached index of all files, so no network access is required.

To get a list of all installed packages, without their version:

```
$ xbps-query -l | awk '{ print $2 }' | xargs -n1 xbps-uhelper
    getpkgname
```

58.3 Verifying RSA keys

If you are installing Void for the first time or the Void RSA key has changed, you may get a message from `xbps-install` claiming:

```
<REPO> repository has been RSA signed by <RSA-FINGERPRINT>
```

To verify the signature, ensure the `<RSA-FINGERPRINT>` matches one of the fingerprints in both void-packages and void-mklive.

59 Advanced Usage

59.1 Downgrading

XBPS allows you to downgrade a package to a specific package version.

59.1.1 Via xdowngrade

The easiest way to downgrade is to use `xdowngrade` from the `xtools` package, specifying the package version to which you wish to downgrade:

```
# xdowngrade /var/cache/xbps/pkg-1.0_1.xbps
```

59.1.2 Via XBPS

XBPS can be used to downgrade to a package version that is no longer available in the repository index.

If the package version had been installed previously, it will be available in `/var/cache/xbps/`. If not, it will need to be obtained from elsewhere; for the purposes of this example, it will be assumed that the package version has been added to `/var/cache/xbps/`.

First add the package version to your local repository:

```
# xbps-rindex -a /var/cache/xbps/pkg-1.0_1.xbps
```

Then downgrade with `xbps-install`:

```
# xbps-install -R /var/cache/xbps/ -f pkg-1.0_1
```

The `-f` flag is necessary to force downgrade/re-installation of an already installed package.

59.2 Holding packages

To prevent a package from being updated during a system update, use xbps-pkgdb(1):

```
# xbps-pkgdb -m hold <package>
```

The hold can be removed with:

```
# xbps-pkgdb -m unhold <package>
```

59.3 Repository-locking packages

If you've used `xbps-src` to build and install a package from a customized template, or with custom build options, you may wish to prevent system updates from replacing that package with a non-customized version. To ensure that a package is only updated from the same repository used to install it, you can *repolock* it via xbps-pkgdb(1):

```
# xbps-pkgdb -m repolock <package>
```

To remove the repolock:

```
# xbps-pkgdb -m repounlock <package>
```

59.4 Ignoring Packages

Sometimes you may wish to remove a package whose functionality is being provided by another package, but will be unable to do so due to dependency issues. For example, you may wish to use doas(1) instead of sudo(8), but will be unable to remove the `sudo` package due to it being a dependency of the `base-system` package. To remove it, you will need to *ignore* the `sudo` package.

To ignore a package, add an appropriate `ignorepkg` entry in an xbps.d(5) configuration file. For example:

```
ignorepkg=sudo
```

You will then be able to remove the `sudo` package using xbps-remove(1).

59.5 Virtual Packages

Virtual packages can be created with xbps.d(5) `virtualpkg` entries. Any request to the virtual package will be resolved to the real package. For example, to create a `linux` virtual package which will resolve to the `linux5.6` package, create an `xbps.d` configuration file with the contents:

```
virtualpkg=linux:linux5.6
```

60 Repositories

Repositories are the heart of the XBPS package system. Repositories can be local or remote. A repository contains binary package files, which may have signatures, and a data file named `$ARCH-repodata` (e.g. `x86_64-repodata`), which may also be signed.

Note that, while local repositories do not require signatures, remote repositories *must* be signed.

60.1 The main repository

The locations of the main repository in relation to a base mirror URL are:

- glibc: `/current`
- musl: `/current/musl`
- aarch64 and aarch64-musl: `/current/aarch64`

60.2 Subrepositories

In addition to the main repository, which is enabled upon installation, Void provides other official repositories maintained by the Void project, but not enabled by default:

- nonfree: contains software packages with non-free licenses
- multilib: contains 32-bit libraries for 64-bit systems (glibc only)
- multilib/nonfree: contains non-free multilib packages
- debug: contains debugging symbols for packages

These repositories can be enabled via the installation of the relevant package. These packages only install a repository configuration file in `/usr/share/xbps.d`.

60.2.1 nonfree

Void has a `nonfree` repository for packages that don't have free licenses. It can enabled by installing the `void-repo-nonfree` package.

Packages can end up in the `nonfree` repository for a number of reasons:

- Non-free licensed software with released source-code.
- Software released only as redistributable binary packages.
- Patented technology, which may or may not have an (otherwise) open implementation.

60.2.2 multilib

The `multilib` repository provides 32-bit packages as a compatibility layer inside a 64-bit system. It can be enabled by installing the `void-repo-multilib` package.

These repositories are only available for `x86_64` systems running the `glibc` C library.

60.2.3 multilib/nonfree

The `multilib/nonfree` repository provides additional 32-bit packages which have non-free licenses. It can be enabled by installing the `void-repo-multilib-nonfree` package.

60.2.4 debug

Void Linux packages come without debugging symbols. If you want to debug software or look at a core dump you will need the debugging symbols. These packages are contained in the debug repository. It can be enabled by installing the `void-repo-debug` package.

Once enabled, symbols may be obtained for `<package>` by installing `<package>-dbg`.

Finding debug dependencies The `xtools` package contains the `xdbg` utility to retrieve a list of debug packages, including dependencies, for a package:

```
$ xdbg bash
bash-dbg
glibc-dbg
# xbps-install -S $(xdbg bash)
```

61 Mirrors

Void Linux maintains mirrors in several geographic regions for you to use. A fresh install will default to using the master mirror in Europe, but you may also select a different mirror manually.

61.1 Tier 1 mirrors

Tier 1 mirrors are maintained by the Void Linux Infrastructure Team. These mirrors sync directly from the build-master and will always have the latest packages available.

Repository	Location
https://alpha.de.repo.voidlinux.org/	EU: Finland
https://mirrors.servercentral.com/voidlinux/	USA: Chicago
https://alpha.us.repo.voidlinux.org/	USA: Kansas City
https://mirror.clarkson.edu/voidlinux/	USA: New York

61.2 Tier 2 mirrors

Tier 2 mirrors sync from a nearby Tier 1 mirror when possible. These mirrors are not managed by Void and do not have any guarantees of freshness or completeness of packages, nor are they required to sync every available architecture or sub-repository.

61.2.1 Globally-available mirrors

Repository	Location
https://mirror.ps.kz/voidlinux/	Asia: Almaty, KZ
https://mirrors.bfsu.edu.cn/voidlinux/	Asia: China
https://mirrors.cnnic.cn/voidlinux/	Asia: China
https://mirrors.tuna.tsinghua.edu.cn/voidlinux/	Asia: China
https://mirror.maakpain.kro.kr/void/	Asia: Seoul, SK
https://void.webconverger.org/	Asia: Singapore
https://mirror.aarnet.edu.au/pub/voidlinux/	AU: Canberra
https://ftp.swin.edu.au/voidlinux/	AU: Melbourne
https://void.cijber.net/	EU: Amsterdam, NL
http://dk.archive.ubuntu.com/voidlinux/	EU: Denmark
http://ftp.dk.xemacs.org/voidlinux/	EU: Denmark
https://mirrors.dotsrc.org/voidlinux/	EU: Denmark
https://quantum-mirror.hu/mirrors/pub/voidlinux/	EU: Hungary
https://mirror.i-novus.ru/mirrors/voidlinux/	EU: Ireland
http://ftp.debian.ru/mirrors/voidlinux/	EU: Russia
https://mirror.yandex.ru/mirrors/voidlinux/	EU: Russia
https://cdimage.debian.org/mirror/voidlinux/	EU: Sweden
https://ftp.acc.umu.se/mirror/voidlinux/	EU: Sweden
https://ftp.gnome.org/mirror/voidlinux/	EU: Sweden
https://ftp.lysator.liu.se/pub/voidlinux/	EU: Sweden
https://ftp.sunet.se/mirror/voidlinux/	EU: Sweden

61.2.2 Region-locked mirrors

Repository	Location
https://mirrors.hushan.tech:44300/voidlinux	Asia: China

61.3 Tor Mirrors

Void Linux is also mirrored on the Tor network. See Using Tor Mirrors for more information.

61.4 Creating a mirror

If you'd like to set up a mirror, and are confident you can keep it reasonably up-to-date, follow one of the many guides available for mirroring with rsync(1), then submit a pull request to the void-docs repository to add your mirror to the appropriate mirror table on this page.

A full mirror requires around 1TB of storage. It is also possible to mirror only part of the repositories. Excluding debug packages is one way of decreasing the load on the Tier 1 mirrors, with low impact on users.

Please keep in mind that we pay bandwidth for all data sent out from the Tier 1 mirrors. You can respect this by only mirroring if your use case for your mirror will offset the network throughput consumed by your mirror syncing.

62 Changing Mirrors

Each repository has a file defining the URL for the mirror used. For official repositories, these files are installed by the package manager in `/usr/share/xbps.d`, but if duplicate files are found in `/etc/xbps.d`, those values are used instead.

To modify mirror URLs cleanly, copy all the repository configuration files to `/etc/xbps.d` and change the URLs in each copied repository file.

```
# mkdir -p /etc/xbps.d
# cp /usr/share/xbps.d/*-repository-*.conf /etc/xbps.d/
# sed -i 's|https://alpha.de.repo.voidlinux.org|<repository>|
   g' /etc/xbps.d/*-repository-*.conf
```

After changing the URLs, you must synchronize xbps with the new mirrors:

```
# xbps-install -S
```

You should see the new repository URLs while synchronizing. You can also use `xbps-query` to verify the repository URLs, but only after they have been synchronized:

```
$ xbps-query -L
 9970 https://alpha.de.repo.voidlinux.org/current (RSA signed
    )
   27 https://alpha.de.repo.voidlinux.org/current/multilib/
      nonfree (RSA signed)
 4230 https://alpha.de.repo.voidlinux.org/current/multilib (
    RSA signed)
   47 https://alpha.de.repo.voidlinux.org/current/nonfree (
      RSA signed)
 5368 https://alpha.de.repo.voidlinux.org/current/debug (RSA
    signed)
```

Remember that repositories added afterwards will also need to be changed, or they will use the default mirror.

63 Using Tor mirrors

Tor is an anonymizing software that bounces traffic via computers all around the world. It can provide access to regular sites on the internet or to hidden sites only available on the network.

The following Void Linux Mirrors are available on the Tor Network:

Repository	Location
http://lysator7eknrfl47rlyxvgeamrv7ucefgrrlhk7rouv3sna25asetwid.onion/pub/voidlinux/	EU: Sweden

63.1 Using XBPS with Tor

XBPS can be made to connect to mirrors using Tor. These mirrors can be normal mirrors, via exit relays, or, for potentially greater anonymity, hidden service mirrors on the network.

XBPS respects the `SOCKS_PROXY` environment variable, which makes it easy to use via Tor.

63.1.1 Installing Tor

Tor is contained in the `tor` package.

After having installed Tor, you can start it as your own user:

```
$ tor
```

or enable its system service.

By default, Tor will act as a client and open a SOCKS5 proxy on TCP port 9050 on localhost.

63.1.2 Making XBPS connect via the SOCKS proxy

XBPS reads the `SOCKS_PROXY` environment variable and will use any proxy specified in it. By simply setting the variable to the address and port of the proxy opened by the Tor client, all XBPS's connections will go over the Tor network.

An example upgrading your system over Tor:

```
# export SOCKS_PROXY="socks5://127.0.0.1:9050"
# xbps-install -Su
```

63.1.3 Using a hidden service mirror

To use a hidden service mirror, the default mirrors need to be overwritten with configuration files pointing to `.onion`-addresses that are used internally on the Tor network. XBPS allows overriding repository addresses under `/etc/xbps.d`.

Copy your repository files from `/usr/share/xbps.d` to `/etc/xbps.d` and replace the addresses with that of an onion service (Lysator's onion used as an example):

```
# mkdir -p /etc/xbps.d
# cp /usr/share/xbps.d/*-repository-*.conf /etc/xbps.d/
# sed -i 's|https://alpha.de.repo.voidlinux.org|http://
   lysator7eknrfl47rlyxvgeamrv7ucefgrrlhk7rouv3sna25asetwid.
   onion/pub/voidlinux|g' /etc/xbps.d/*-repository-*.conf
```

Tor provides layered end-to-end encryption so HTTPS is not necessary.

When installing packages, with `SOCKS_PROXY` set like the earlier example, XBPS should indicate that it is synchronizing the repositories from the onion address specified in the override:

```
# xbps-install -S
[*] Updating `http://
   lysator7eknrfl47rlyxvgeamrv7ucefgrrlhk7rouv3sna25asetwid.
   onion/pub/voidlinux/current/aarch64/nonfree/aarch64-
   repodata' ...
aarch64-repodata: 4030B [avg rate: 54KB/s]
[*] Updating `http://
   lysator7eknrfl47rlyxvgeamrv7ucefgrrlhk7rouv3sna25asetwid.
   onion/pub/voidlinux/current/aarch64/aarch64-repodata' ...
aarch64-repodata: 1441KB [avg rate: 773KB/s]
```

63.1.4 Security consideration

It is advisable to set `SOCKS_PROXY` automatically in your environment if you are using an onion. If the setting is missing, a DNS query for the name of the hidden service will leak to the configured DNS server.

To automatically set the environment variable, add it to a file in `/etc/profile.d`:

```
# cat - <<EOF > /etc/profile.d/socksproxy.sh
#!/bin/sh
export SOCKS_PROXY="socks5://127.0.0.1:9050"
EOF
```

64 Restricted Packages

Void offers some packages that are officially maintained, but not distributed. These packages are marked as restricted and must be built from their void-packages template locally.

Packages can be restricted from distribution by either the upstream author or Void. Void reserves the right to restrict distribution of any package for effectively any reason, massive size being the most common. Another common reason is restrictive licensing that does not allow third-party redistribution of source or binary packages.

64.1 Building manually

You can use `xbps-src` in the void-packages repository to build the restricted packages from templates. For instructions on building packages from templates, refer to the void-packages documentation, and the "Quick start" section in particular .

Remember that the building of restricted packages must be enabled explicitly by setting `XBPS_ALLOW_RESTRICTED=yes` in your `xbps-src` configuration (in the `etc/conf` file in the repository.)

64.2 Automated building

There is also a tool, xbps-mini-builder which automates the process of building a list of packages. The script can be called periodically and will only rebuild packages if their templates have changed.

65 Custom Repositories

Void supports user-created repositories, both local and remote. This is only recommended for serving custom packages created personally, or packages from another trusted source. The Void project does not support *any* third-party package repositories - the use of third-party software packages poses very serious security concerns, and risks serious damage your system.

65.1 Adding custom repositories

To add a custom repository, create a file in `/etc/xbps.d`, with the contents:

```
repository=<URL>
```

where `<URL>` is either a local directory or a URL to a remote repository.
For example, to define a remote repository:

```
# echo 'repository=http://my.domain.com/repo' > /etc/xbps.d/
    my-remote-repo.conf
```

Remote repositories need to be signed. xbps-install(1) refuses to install packages from remote repositories if they are not signed.

To define a local repository:

```
# echo 'repository=/path/to/repo' > /etc/xbps.d/my-local-repo
    .conf
```

66 Signing repositories

Remote repositories **must** be signed. Local repositories do not need to be signed.

The xbps-rindex(1) tool is used to sign repositories.

The private key for signing packages needs to be a PEM-encoded RSA key. The key can be generated with either ssh-keygen(1) or openssl(1):

```
$ ssh-keygen -t rsa -m PEM -f private.pem
```

```
$ openssl genrsa -out private.pem
```

Once the key is generated, the public part of the private key has to be added to the repository metadata. This step is required only once.

```
$ xbps-rindex --privkey private.pem --sign --signedby "I'm
   Groot" /path/to/repository/dir
```

Then sign one or more packages with the following command:

```
$ xbps-rindex --privkey private.pem --sign-pkg /path/to/
   repository/dir/*.xbps
```

Note that future packages will not be automatically signed.

67 Troubleshooting XBPS

Sometimes the package manager gets in a weird spot and can't fix itself without help. This section documents important fixes and things that can go wrong when working with XBPS.

67.1 Section Contents

- Common Errors
- Static XBPS

68 Common Errors

68.1 Errors while updating or installing packages

If there are any errors while updating or installing a new package, make sure that you are using the latest version of the remote repository index. Running xbps-install(1) with the `-S` option will guarantee that.

68.1.1 "Operation not permitted"

An "Operation not permitted" error, such as:

```
ERROR: [reposync] failed to fetch file https://alpha.de.repo.
    voidlinux.org/current/nonfree/x86_64-repodata': Operation
    not permitted
```

can be caused by your system's date and/or time being incorrect. Ensure your date and time are correct.

68.1.2 "Not Found"

A "Not Found" error, such as:

```
ERROR: [reposync] failed to fetch file `https://alpha.de.repo
    .voidlinux.org/current/musl/x86_64-repodata': Not Found
```

usually means your XBPS configuration is pointing to the wrong repositories for your system. Confirm that your xbps.d(5) files refer to the correct repositories.

68.1.3 shlib errors

An "unresolvable shlib" error, such as:

```
libllvm8-8.0.1_2: broken, unresolvable shlib 'libffi.so.6'
```

is probably due to orphan packages, already removed from the Void repos, still being installed on your system. This can be solved by running xbps-remove(1) with the `-o` option, which removes orphan packages.

If you get an error message saying:

```
Transaction aborted due to unresolved shlibs
```

the repositories are in the staging state, which can happen due to large builds. The solution is to wait for the builds to finish. You can view the builds' progress in the Buildbot's Waterfall Display.

68.1.4 repodata errors

In March 2020, the compression format used for the repository data (repodata) was changed from gzip to zstd. If XBPS wasn't updated to version `0.54` (released June 2019) or newer, it is not possible to update the system with it. Unfortunately, there isn't an error message for this case, but it can be detected by running `xbps-install` with the `-Sd` flags. The debug message for this error is shown below.

```
[DEBUG] [repo] '//var/db/xbps/
   https___alpha_de_repo_voidlinux_org_current/x86_64-
   repodata' failed to open repodata archive Invalid or
   incomplete multibyte or wide character
```

In this situation, it is necessary to follow the steps in xbps-static.

68.2 Broken systems

If your system is for some reason broken and can't perform updates or package installations, using a statically linked version of xbps to update and install packages can help you avoid reinstalling the whole system.

69 Static XBPS

In rare cases, it is possible to break the system sufficiently that XBPS can no longer function. This usually happens while trying to do unsupported things with libc, but can also happen when an update contains a corrupt glibc archive or otherwise fails to unpack and configure fully.

Another issue that can present itself is in systems with a XBPS version before `0.54` (released June 2019). These systems will be impossible to update from the official repositories using the regular update procedure, due a change in the compression format used for repository data, which was made in March 2020.

In these cases it is possible to recover your system with a separate, statically compiled copy of XBPS.

69.1 Obtaining static XBPS

Statically compiled versions of XBPS are available in all mirrors in the static/ directory. The link below points to the static copies on the primary mirror in Germany:

https://alpha.de.repo.voidlinux.org/static

Download and unpack the latest version, or the version that matches the broken copy on your system (with a preference for the latest copy).

69.2 Using static XBPS

The tools in the static set are identical to the normal ones found on most systems. The only distinction is that these tools are statically linked to the musl C library, and should work on systems where nothing else does. In systems where the platform can no longer boot, it is recommended to chroot in with Void installation media and use the static tools from there, as it is unlikely that even a shell will work correctly on the target system. When using static XBPS with glibc installation, environmental variable `XBPS_ARCH` need to be set.

70 Contributing

There's more to running a distribution than just writing code. This section explains how to be an active part of Void.

Please also visit the Void Web site for further information about how to participate, including our communication channels and how to contribute to the Void package repository.

70.1 Section Contents

- Usage Statistics
- Contributing To The void-docs Project

71 Usage Statistics

If you would like to contribute usage reports, the PopCorn program reports installation statistics back to the Void project. These statistics are purely opt-in, the reporting programs are *not* installed by default on any void systems.

PopCorn only reports which packages are installed, their version, and the host CPU architecture (the output of `xuname`.) This does not report which services are enabled, or any other personal information. Individual systems are tracked persistently by a random (client generated) UUID, to ensure that each system is only counted once in each 24-hour sampling period.

The data collected by *PopCorn* is available to view at http://popcorn.voidlinux.org

71.1 Setting up PopCorn

First, install the `PopCorn` package. Then, enable the `popcorn` service, which will attempt to report statistics once per day.

72 Contributing To The void-docs Project

The sources for this handbook are hosted in the void-docs repository on GitHub. If you would like to make a contribution, please read about the purpose of the Handbook, follow our style guide and submit a pull request.

73 Style Guide

This style guide outlines the standards for contributing to the void-docs project. The manual on https://docs.voidlinux.org is generated from an mdBook stored in the void-docs repository.

73.1 General

Although there will always be cases where command listings are appropriate, the contents of the Handbook should be written in American English (or the relevant language in the case of translations of the Handbook).

Outside of the 'installation' sections, step-by-step instructions containing 'magic' commands for copying-and-pasting are strongly discouraged. Users are expected to read the canonical documentation (e.g. man pages) for individual programs to understand how to use them, rather than relying on copying-and-pasting.

Command code-blocks should not be used to describe routine tasks documented elsewhere in this Handbook. For example, when writing documentation for the `foo` package, do not provide a command code-block stating that one should install it via `xbps-install foo`. Similarly, do not provide code blocks describing how to enable the `foo` service.

73.2 Formatting

For markdown formatting, the void-docs project uses the Versioned Markdown format, and enforces use of the auto-formatter `vmdfmt`, which works very similarly to `gofmt`. Most valid markdown is accepted by the formatter. The output format is described in the project's README.

After installing the `vmdfmt` package, you can format a file by running:

```
vmdfmt -w <filepath>
```

To format the entire *mdbook* from the repository root, outputting a list of files modified, run:

```
vmdfmt -w -l <filepath>
```

`vmdfmt` is used by the void-docs repository's `check.sh` script, which must be run locally before submitting a pull request.

73.3 Commands

Command code-blocks should start with a `#` or `$` character, indicating whether the command should be run as `root` or a regular user, respectively.

For example:

```
# vi /etc/fstab
```

and not:

```
$ sudo vi /etc/fstab
```

and also not:

```
vi /etc/fstab
```

Command code-blocks should be introduced with a colon (':'), i.e.:

For example:

```
$ ls -l
```

73.3.1 Placeholders

Placeholders indicate where the user should substitute the appropriate information. They should use angle brackets ('<' and >) and contain only lower-case text, with words separated by underscores. For example:

```
# ln -s /etc/sv/<service_name> /var/service/
```

and not:

```
# ln -s /etc/sv/[SERVICENAME] /var/service/
```

73.4 Links

Link text should not include sentence-level punctuation. For example:

```
[Visit this site](https://example.org).
```

and not:

```
[Visit this site.](https://example.org)
```

73.4.1 Internal links

Links to other sections of the Handbook must be relative. For example:

```
[example](./example.md#heading-text)
```

and not:

```
[example](example.md#heading-text)
```

When referring literally to a Handbook section, the section title should be placed in double-quotes. Otherwise, double-quotes are not required. For example:

```
For more information, please read the "[Power Management](./
    power-management.md)" section.
```

and

```
Void provides facilities to assist with [power management](./
    power-management.md).
```

73.4.2 Man Page Links

The first reference to a command or man page must be a link to the relevant man page on `https://man.voidlinux.org/`.

The link text must contain the title section number in parenthesis, and contain no formatting. For example: man(1), not ‘man(1)‘.

73.4.3 Auto Links

Auto links (links with the same title as URL) should use the following notation:

```
<https://www.example.com/>
```

They should not be formatted like this:

```
[https://www.example.com/](https://www.example.com/)
```

73.4.4 Checking links

If you’re including new links (either internal or external) in the docs or changing the current file structure, you should make use of the `mdbook-linkcheck` package:

```
$ mdbook-linkcheck -s
```

This will verify all the references, and warn you if there are any issues. If any link you’re using is correct but generating errors for some reason, you can add its domain to the exclude list in `book.toml`, under the `[mdbook.linkcheck]` key. `mdbook-linkcheck` is used by the void-docs repository’s `check.sh` script, which must be run locally before submitting a pull request.

73.5 Case

Filenames and directories should use kebab case when splitting words. For example the filename should be `post-install.md` not `postinstall.md`.

Words that are part of trademarks or well known package names are exempt from this rule. Examples include `PulseAudio` and `NetworkManager` which are well known by their squashed names.

73.6 Voice

Prefer the active imperative voice when writing documentation. Consider the following examples:

> Now we need to install the CUPS drivers and configure them.

This version is conversational and friendlier, but contains unnecessary language that may not be as clear to an ESL reader.

> Install and configure the CUPS drivers, then configure them as shown.

This version contains a clear command to act, and a follow up that shows what will be done next. It is clear both to native English speakers, ESL readers, and to translators.

73.7 Notes

Notes should only be used sparingly, and for non-critical information. They should begin with "Note: ", and not be block-quoted with >. For example, the Markdown should look like:

```
Note: You can also use program X for this purpose.
```

and not:

```
> You can also use program X for this purpose.
```

73.8 Block quotes

Block quotes (i.e. >) should only be used to quote text from an external source.

74 Submitting Changes

Proposed changes should be submitted as pull requests to the void-docs repository on GitHub. Please note that, unlike a wiki, submissions will be reviewed before they are merged. If any changes are required they will need to be made before the pull request is accepted. This process is in place to ensure the quality and standards of the Handbook are sustained.

74.1 Requirements

To clone the repository and push changes, git(1) is required. It can be installed via the `git` package.

Building the Handbook locally requires mdBook, which can be installed via the `mdBook` package.

74.2 Forking

To fork the repository a GitHub account is needed. Once you have an account, follow GitHub's guide on setting up a fork.

Clone the repository onto your computer, enter it, and create a new branch:

```
$ git clone https://github.com/<your_username>/void-docs.git
$ cd void-docs
$ git checkout -b <branch_name>
```

You can then edit the repository files as appropriate.

74.3 Making changes

To serve the docs locally and view your changes, run `mdbook serve` from the root of the repository.

Once you are satisfied with your changes, run the `check.sh` script provided in the repository root. This will run the `vmdfmt` command, which will wrap the text appropriately, and the `mdbook-linkcheck` command, which will check for broken links. Address any issues raised by `check.sh`.

Once `check.sh` runs without errors, push your changes to your forked repository:

```
$ git add <edited_file(s)>
$ git commit -m "<commit_message>"
$ git push --set-upstream origin <branch_name>
```

The commit message should be in the form `<filename>: <description_of_changes>`.

Pull requests should only contain a single commit. If a change is made after the initial commit, `git add` the changed files and then run `git commit -amend`. The updated commit will need to be force-pushed: `git push -force`.

If multiple commits are made they will need to be squashed into a single commit with `git rebase -i HEAD~X`, where `X` is the number of commits that need to be squashed. An

editor will appear to choose which commits to squash. A second editor will appear to choose the commit message. See git-rebase(1) for more information. The updated commit will need to be force-pushed: `git push -force`.