

Computing On Linux

Adam Harmanský

November 21, 2021



Contents

Preface	V
What Linux Is And What It Is Not	V
What Is An Operating System Anyway?	V
What is Linux?	VI
Installation	VII
Downloading the system and creating a bootable USB drive	VII
Microsoft Windows	VII
Linux	VII
Booting The Live USB Drive	VII
Running the installer	VIII
Basic settings	VIII
Partitions and Filesystems	VIII
Installation	VIII
Updating the system	VIII
Troubleshooting	IX
WiFi doesn't work	IX
Permission Denied On SSL	X
Storing Data	1
Computers And Numbers	1
Binary	1
Hexadecimal	2
Little Endian Vs. Big Endian	2
Storing information	2
Files	3
Representing Different Types Of Information	4
ASCII	4
Files On Linux	5
Basic Commands	6
ls And cd	6
Basic Text Editing With Vi	6
Displaying Files	7
Removing Files	7
Creating Directories	8
Command Arguments And Options	8
File Paths	9
Selecting Multiple Files	9
Home Directory	9
Moving, Copying And Renaming files	9
More On Files	9
File Attributes	9
Modifying File Attributes	11
Symbolic And Hard Links	11
Getting Help And Information	13
The Manual	13
ArchWiki	13
Online Forums	13
Void Linux Handbook	13

Programs	14
What Are Programs?	14
Shell Scripts	14
Installing Programs - The Package Manager	14
Becoming The Superuser	15
Installing Packages	15
Updating packages	16
Removing Packages	16
Searching For Packages	16
Processes And Services	18
Killing Processes	19
Shell Jobs	19
Stopping Programs	19
Running A Background Program Outside The Shell	19
Services	20
Enabling And Disabling Services	20
Virtual Terminals	20
Pipes	21
Files, Streams And File Descriptors	21
File Redirection	21
Redirecting The Output Of A Command Into A File	21
Redirecting A File Into The Input Of a Command	21
Redirecting Other Streams	22
Special Files In The /dev Directory	22
Pipes	22
Pipe-oriented Data Processing	23
Finding Regular Expressions With Grep	23
Processing Text With Sed	25
Processing Tables With Awk	26
Named Pipes	29
Managing Drives On Linux	31
Mounting Drives	31
Partitioning Drives	31
Creating Filesystems	31
Resizing The Filesystem	32
Copying Disk Images With dd	32
Shell Scripting	32
Shell And Environment Variables	32
Environment Variables	32
Shell Variables	33
Shell Script Arguments	33
Comments	34
Exit Values And Conditions	34
The Test Command	34
If And Else	35
The For Loop	35
Command Substitution	35
The Case Statement	36

Bashrc And Profile	36
Exec	37
The X Graphical Environment	38
Starting X	39
The X Selection And Clipboard	40
Keyboard Shortcuts	40
Simple Application Launcher With Dmenu	41
Starting X On Login	41
Advanced Window Management With DWM	42
Setting Up DWM	42
Using DWM	42
Workspaces	43
The ST Terminal Emulator	43
Fonts	43
Audio	44
X Tweaks	44
Time In The DWM Status Bar	44
Touchpad Tapping	44
Enable Touchpad While Typing	45
Faster Keyboard Repeat	45
Setting The Wallpaper	45
More Customization	45
GTK Themes	45
Icon Themes	46
Cursor Themes	46
Suckless Patches	46
Going Further	47
General User Programs	47
Edit Text More Efficiently With Vim	47
Checking The Battery Status With ACPI	47
Displaying The Calendar	47
The GNU Calculator	47
Calculate The Prime Factors Of A Number	47
Archiving Files	47
Finding Files	48
Displaying Images	48
Interacting With Websites And Downloading Files With Curl	48
Getting The Current Time	48
Setting The Brightness The Easy Way	49
Playing Video	49
Removing A File Completely	49
Editing Images	49
Browse The Web In The Terminal	49
Counting Words In A Document	49
Reading PDF Files	49
Funny Eyes That Follow The Cursor	49
Scripting Programs	49
Displaying A Message Window	49
Getting The Contents Of The X Selection	49
Wait For A Given Amount Of Time	50

Pipe-Oriented Programs	50
Encoding And Decoding Data In Base64 And Base32	50
Getting N Lines From The Beginning Or The End Of A File	50
Getting The Hexadecimal Contents Of A File	50
Shuffling The Lines In A File	50
Sort Lines Of Text	50
System Administration	50
Stopping And Rebooting The Computer	50
Changing The Default Shell For A User	50
Checking The Disk Usage	50
Reading The Kernel Logs	51
Filesystem Check	51
Getting And Setting The System Hostname	51
Getting The IP Address	51
CPU Info	51
Memory Usage	51
Remotely Access Another Computer With SSH	51
Redirect The Intermediate Output Of A Pipeline	51
Remove Repeating Lines In A File	51
See How Long The Machine Has Been Running	51
Create A User Or Group	51
Add A User To A Group	52
Edit Superuser Access	52
Remove Old Versions Of The Kernel	52
Open A Root Shell	52
Connecting To WiFi Networks	52
Monitor Configuration	52

Preface

This book is not only about Linux, but also about computing in general. It introduces the reader to the world of Linux and free software. The book doesn't rely on past experience with computers - it will be just as useful (and just as confusing) for people that come from another operating system, like Microsoft Windows or MacOS. What this book *does* rely on are external sources of information. This book will not tell you everything. It will only guide you on your journey of exploring Linux. It will tell you where to look for solutions to problems and what programs to use. It is like the trunk of a pine tree - every chapter branches off into multiple topics, which branch off into information. However, every piece of information requires the base knowledge, provided by this book.

What Linux Is And What It Is Not

What Is An Operating System Anyway?

You may be familiar with the terms *hardware* and *software*. Hardware consists of the physical components of the computer - the *central processing unit* - CPU, the *memory* - RAM ¹, hard drives etc. Software consists of the programs on the computer - its thoughts. Computers, though, don't think like we, humans, do. An operating system is the layer between hardware and software - it allows programs to run without needing to communicate directly with hardware and the user to type in commands instead of modifying the hardware of the computer.

The core part of the operating system is the **kernel**. When the computer is switched on, a special piece of software called the BIOS (basic input-output system) loads the kernel into memory ². The kernel then loads the device *drivers* and finally the *init* program, which is the first *user-space* program run by the computer. Drivers are simple programs that allow the user-space programs to use the hardware without knowing what data to send to the specific chips on the computer. For example, to create a file, the program doesn't need to know how the hard drive works or how the filesystem is organized - it makes a *system call* and the system figures out what to do.

Another important thing the kernel does is *multitasking*. A real CPU usually only has a couple *cores* and single-core CPUs are still common. Yet the computer can run a web browser, a text editor, and an e-mail application at once! This is because the kernel can stop a task mid-execution and allow other programs to run for a split second. In a *preemptive multitasking* system, the programs don't even notice that they need to be interrupted - they are frozen in time until it is their turn again. Of course, when a program is waiting for another device, like the keyboard, the operating system can let other programs run in the meantime. If all the programs are just waiting for input and not computing anything, the computer should, in theory, use 0% of the CPU time.

The kernel also manages memory. When a program needs memory, it tells the system to allocate a chunk of memory. The operating system needs to be very careful not to let multiple programs use the same place in the memory for different things. *Shared memory* is also managed by the operating system - some programs actually rely on using the same piece of memory to communicate. This can be also used to reduce the amount of memory used by so called *shared libraries*.

An operating system also consists of a user interface and some basic programs required for file manipulation and installing other programs.

The **shell** is a program that allows the user to run other programs. It can either interpret commands, or provide a GUI (graphical user interface) to run other graphical programs. In Linux, most programs

¹RAM stands for random access memory

²Actually, the kernel is loaded by a special program called the bootloader, which is not a part of the kernel. The BIOS finds the bootloader and the bootloader then finds the kernel.

are text-based. They are run in a *terminal*³ that displays the output and reads the input for a program. The shell we will be using is the *GNU Bourne-again shell* - BASH. It is very common and mostly complies with the POSIX⁴ standards.

An important part of the operating system is the C standard library. Programs made in the C programming language use special *functions* instead of system calls. This helps reinforce the procedural paradigm of the language. Creating an extra layer of abstraction is useful for more efficient memory management and simpler IO (input-output) procedures. The C library (`libc`) we will be using is the *GNU libc*, known as `glibc`.

What is Linux?

Many people think that Linux is a whole operating system, but it's actually just the kernel. Many parts of the operating system we'll be using, like the shell or the file management programs come from the GNU project. GNU and Linux are both free software. This means, that they're open source - their source code is accessible for free. Free software is more secure and available for everyone, simple to extend upon and fun to use.

What is unique about free software is the variety. It is much more modular - you can change many parts of your OS, or even make your own operating system based on Linux. Because of this, there isn't one "Linux", or better said "GNU/Linux" system. Instead, there are different *distributions* - different operating systems based on the Linux kernel (most often containing GNU programs) that use different combinations of software. Some of the more popular distributions are Ubuntu, Fedora, Arch Linux or Debian. We will be using the **Void Linux** distribution - a small independent Linux operating system that is fast, small and lightweight. It is by far not the most popular or common Linux distribution, but that is not a problem, since many programs and commands don't differ across Unix-like operating systems - most of these commands should even work on MacOS!

Glossary

Term	Explanation
Unix	An operating system developed by Bell Labs in the 1970's
GNU	An effort to make an entirely free alternative to Unix
Linux	A free kernel, adapted by the GNU project.

³The terminal is actually a physical device that displays text and reads keyboard input. Most modern computers will likely run a **terminal emulator** - a program that draws the text on the screen

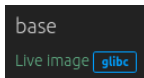
⁴POSIX is a set of standards specified by IEEE for maintaining compatibility between operating systems

Installation

Before using an operating system, we have to install it first. This will be tricky, since we don't know anything about computers yet. So if you don't understand something, don't bother! We need to get things set up first and explain everything later.

Downloading the system and creating a bootable USB drive

For this step, you will need a working computer (with a web browser and Microsoft Windows or Linux) and a USB drive (2GB should be more than enough). Visit the website <https://voidlinux.org/> and click the Download button. You will be taken to the download page. Here, make sure *x86_64* is selected (use *i686* for 32-bit machines) and download the base glibc live image:



You will download an ISO image of the USB drive. Now you need to copy it to the drive itself.

Microsoft Windows

A simple way to do this on Microsoft Windows is to use a program called Rufus at <https://rufus.ie/>. Start the program, select the device, press the **SELECT** button and select the ISO file, press **START** and in case of any question follow the recommended options. When the installation is done, you can remove the USB drive.

Linux

For Linux, you don't need to install an extra program. First type `lsblk` to see which drive is your USB stick. If you know the size of the USB drive, you should find it easily. This is the output I get:

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINTS
sda	8:0	0	238.5G	0	disk	
'-sda1	8:1	0	238.5G	0	part	/
sdb	8:16	1	14.4G	0	disk	
+-sdb1	8:17	1	468M	0	part	
'-sdb2	8:18	1	32M	0	part	

You can see that `sdb` is the USB drive. `umount` the drive if the `MOUNTPOINTS` field isn't empty. Now `cd` into the directory containing the downloaded file and type:

```
dd if=void-live-* of=/dev/sdb
```

Wait until the prompt appears again. Remove the drive and move onto the next step.

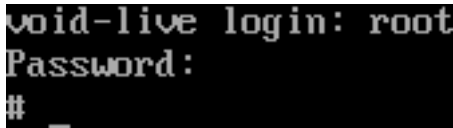
Bootting The Live USB Drive

Plug the USB stick into the computer you want to install Linux on. You will now need to get into the BIOS boot menu. The process of entering the boot menu may be different for every PC. On most PCs, press the power button and immediately start rapidly pressing the **F12** key. If a menu listing your drives appears, you have succeeded at opening the boot menu. If it does not, try other keys, like **F11**, **Backspace**, etc. If nothing works, try entering the BIOS settings using the **F2** or **DEL** key. There, rearrange the boot order so the USB drive has the highest priority. After installation, you will need to change the boot order back to how it was before.

In the boot menu, select the USB drive. You should see another menu with the Void Linux logo in the background. Here, just press **Enter** to select the default option. A live Linux system will now boot up.

Running the installer

You will be asked for a username and a password for the live system. The username is `root` and the password is `voidlinux`. A prompt will now show up:



```
void-live login: root
Password:
#
```

To run the installer, type `void-installer` and press `Enter`.

Basic settings

You will be shown a dialog saying something along the lines of “Welcome to void Linux blah blah blah ...”. Press `Enter`. Now, you will be shown an installation menu. Select **Keyboard** and select `us`.

We will now connect to internet. For **network**, select the default adapter and if you are asked whether to use `dhcp`, select `yes`. If something doesn’t work, refer to the *Troubleshooting* section or ask someone who knows what to do.

Set source to `local` and the hostname to the name of your computer. Locale defines the language and number formatting. Select `en_US`. Set the timezone.

The root password is the password for the root user. The `root` user is the administrator of the computer. The root user can install programs, edit system files, ... Then create a normal user - enter your name, full name and password. In the bootloader option, select your primary hard drive - the boot loader is what loads your operating system on startup. Select the graphical terminal option if you want to.

Partitions and Filesystems

In the Partition option, select your primary drive and use `cfdisk`. If you are asked to select a label type, select `gpt`. Remove all existing partitions. Press `new` and set the size to `1M`. Then press `type` and set the type to `BIOS boot`. Press the down arrow to select the free space. Press `new` and now leave the partition size at the default value. The type of this partition should be `Linux filesystem`. Press `write` and type `yes`. Then press `quit`.

Now select **Filesystems**. Choose the larger one of the partitions that you have just created. Set the filesystem type to `ext4` and the mountpoint to `/`. When you are asked whether to create a new filesystem, press `Yes`. Press `Done`.

Installation

Press `Install`. Wait for that system to install. After the installation is finished, unplug the USB drive and restart the computer. **DON NOT HOLD THE ENTER KEY UNTIL IT REPEATS** (I know that from my own experience). If you changed the boot order at the beginning, change it back.

You should now see a login prompt with the name of your machine:



```
lmao login:
```

Updating the system

It is important to update your freshly installed system after installing. Log in as `root` (use the root password that you selected upon installation). When you see the root prompt (the `#` character), type

in the following command:

```
xbps-install -S
```

If it fails, your internet connection is not configured correctly. Then, type:

```
xbps-install -u xbps
```

It may happen that the `xbps` package is already up to date. Then, use:

```
xbps-install -Su
```

And press `y` (and then `Enter`) to update the system.

Type `exit`.

Troubleshooting

These are some common issues you might encounter during installation.

WiFi doesn't work

I actually think that the current version of the Void Linux installer is broken, because it has never happened to me that WiFi had worked the first time I set my laptop up. The installer has probably broken your `wpa_supplicant` configuration. The solution shouldn't be hard, though.

After installing the system, log in as root and type `ip a` to see a list of your internet adapters:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: wlp2s0: <BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 7c:2a:31:b8:db:e8 brd ff:ff:ff:ff:ff:ff
```

The `lo` adapter is just a local loopback adapter. Then, you should see a second adapter name starting with `wl`. Remember the name of the adapter - mine is `wlp2s0`.

Then, run `vi /etc/sv/wpa_supplicant/run`. This will open a special text editor called `vi`. Now press `dd`, until the entire file is deleted. Then, press `i` to enter *insert mode*. Type in the following (replace `wlp2s0` with the name of your WiFi adapter):

```
#!/bin/sh
exec 2>&1
exec 1>&-
exec wpa_supplicant -iwlp2s0 -c/etc/wpa_supplicant/wpa_supplicant.conf
```

Press the `ESC` key to return to normal mode. Type `:wq` to save and exit.

Now edit the network configuration file with `vi /etc/wpa_supplicant/wpa_supplicant.conf`. You should see something like this:

```
# Default configuration file for wpa_supplicant.conf(5).

ctrl_interface=/run/wpa_supplicant
ctrl_interface_group=wheel
eapol_version=1
ap_scan=1
```

```
fast_reauth=1
update_config=1
```

Add here your networks.

- If you don't, press i, add the text above, press ESC to return to normal mode and :wq to exit.
- If you do see the text above, just type :q to exit vi.

To add your WiFi network use the following command:

```
wpa_passphrase network_name network_password >> /etc/wpa_supplicant/wpa_supplicant.conf
```

The >> operator appends the output of the command to the end of the file.

Now your WiFi should work! Type in `sv restart wpa_supplicant dhcpcd` to restart the network services and wait a second for them to start up. Then you can test your connection. For example, use the command `ping -c 4 www.google.com`. This will *ping* the Google servers four times - it sends a short message to the server and the server responds. If the server responds in time, you should see the following:

```
PING www.google.com (172.217.23.228) 56(84) bytes of data.
64 bytes from prg03s06-in-f228.1e100.net (172.217.23.228): icmp_seq=1 ttl=60 time=13.6 ms
64 bytes from prg03s06-in-f4.1e100.net (172.217.23.228): icmp_seq=2 ttl=60 time=13.5 ms
64 bytes from prg03s06-in-f4.1e100.net (172.217.23.228): icmp_seq=3 ttl=60 time=13.5 ms
64 bytes from prg03s06-in-f4.1e100.net (172.217.23.228): icmp_seq=4 ttl=60 time=13.6 ms
```

Congratulations! You have just set up WiFi!

Permission Denied On SSL

The Void Linux servers use an encrypted protocol to transfer packages. This protocol requires the system time to be set correctly. On older hardware, however, the battery of the hardware clock may be damaged or just discharged, and the system clock is reset. There are two ways to solve the problem:

1. You can just set the system time using the `date -s` command, as shown below. The time requires second-level precision, which you might not be able to achieve without another computer:

```
date -s "Nov 29 18:54:25 2077"
```

After running this command, you may attempt to synchronize the hardware clock with the system clock:

```
hwclock -w
```

Because your hardware clock is probably broken, it will be useful to install the program `ntp`, which updates the system clock from the internet. Install the program with:

```
SSL_NO_VERIFY_PEER=1 xbps-install ntp
```

Then, you can run the following command to update the time from the internet:

```
ntpdate -u 0.pool.ntp.org
```

We will enable `ntp` as a service so that it runs every time the computer starts up:

```
ln -s /etc/sv/ntpd /etc/runit/runsvdir/current/
```

2. A temporary solution would be to bypass the encryption mechanism using the following command:

```
export SSL_NO_VERIFY_PEER=1
```

Storing Data

Computers And Numbers

You may already know that all that computers really do is that they just manipulate ones and zeros. But why ones and zeros? And how can these ones and zeroes be useful? It all has to do with how the computer is built.

First of all, why do computers even use electric signals? The answer is rather obvious - electricity can move very quickly and efficiently. There is one problem, though. Humans use the decimal system to represent numbers - we use the figures 1, 2, 3, 4, 5, 6, 7, 8, 9, and 0. This is too many for one wire. By using different voltages, we could achieve multiple signal levels in theory, but in the real world, wires can have different resistance, capacity, or length and it would be hard for an electric circuit to tell which one of the signals it is really receiving. That's why we only use two numbers - 1 and 0. Usually, 1 is represented with a **HIGH** signal - a positive voltage and a 0 is a **LOW** signal - a negative voltage. This makes circuits simpler and more reliable. The number system that uses only ones and zeros is called the *binary* system.

Binary

A single binary digit - from now on referred to as a *bit*, is not very useful - the same way you can't do much only with numbers from 0 to 9. That is why we group the bits together, just as we do in the decimal system when we create numbers like 12 or 845. Except that the last digit in the binary system is one, not nine. This is a small conversion table from binary to decimal.

decimal	binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000

You can notice that just as we have the tens and hundreds place in the decimal system, we have the ones, twos, fours and eights place in binary. If you have the number 8569 in decimal, you could write it in the following notation:

$$8569 = 9 \cdot 10^0 + 6 \cdot 10^1 + 5 \cdot 10^2 + 8 \cdot 10^3$$

The same number in binary would be:

$$8569 = 10000101111001 = 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + 1 \cdot 2^6 + 0 \cdot 2^7 + 1 \cdot 2^8 + 0 \cdot 2^9 + 0 \cdot 2^{10} + 0 \cdot 2^{11} + 0 \cdot 2^{12} + 1 \cdot 2^{13}$$

Addition, subtraction, multiplication and division work the same in binary as they do in decimal, but that is far out of the scope of this chapter. If you are interested in how that works, you can find a lot of great explanations on the internet.

Hexadecimal

You can see that where we had to use four digits in decimal, we have to use 13 bits in binary. This is not a problem for computers, but it's rather hard for us to make sense of. That's why you will often see the *hexadecimal* notation. The hexadecimal number system used 16 digits - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. It's a compromise - the numbers are shorter, but can be decoded quickly into binary. That's because there are 16 possible combinations of four bits. So to convert between binary and hexadecimal, we can just convert each group of four bits into the corresponding hexadecimal digit:

decimal	binary	hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

To distinguish the number system, programmers often use the `0x` prefix for hexadecimal numbers. The suffix `h` is also common for hexadecimal and `b` for binary.

$$0x2F3E = 2F3Eh = 10111100111110b = 12094$$

Little Endian Vs. Big Endian

Digits in a number are arranged by their significance - the most significant digit is on the left and the least significant digit is on the right - so in the number 123, the digit 1 is the most significant digit - it has the greatest effect on the size of the number and the digit 3 is the least significant. Humans always use the *big endian* notation - you start from the “big end” and end at the “little end”. It's readable for humans, but not very useful for computers. For a number of reasons, many computers use the *little endian* notation - you start with the least significant bit and end at the most significant bit. You need to be careful with the notation - if you read the number the other way around, it can cause a big problem.

Storing information

To store information for use later, we need a special circuit that can hold an electric signal after being activated by a short pulse. These circuits exist and are called “flip-flops”. They are great to store small amounts of information inside the CPU itself. Storing and manipulating large amounts of data requires a large continuous field of these circuits. We can create an array of flip-flop-like devices called *memory*. Every item in memory has an address - a number representing where the information is located. To see what number is at a given place in memory, you need to know its address. The type of memory that can give you the number at an arbitrary address is called *random access memory* or RAM. However, it

would be quite impractical to address numbers bit by bit. Therefore, memory is organized into *bytes* - groups of 8 bits. One byte can be represented by two hexadecimal digits.

The computer interprets programs step by step. Therefore, it needs to store the results of previous steps for later use. The more RAM your computer has, the more information it can process at once. The amount of information in the memory is measured in bytes - *B*, kilobytes - *KB*, megabytes - *MB*, gigabytes - *GB* and even terabytes - *TB*. Special units have been developed to further simplify measuring the amount of memory. The binary number 1000000000 (2^{10}) represents the decimal number 1024. This number is close to the decimal 1000 only by 2.4. The unit *KiB* - kibibyte represents 1024 bytes. Other units include the MiB - mebibyte (1024^2B), or GiB - gibibyte.

Uppercase B stands for byte and lowercase b stands for bits.

REMEMBER: $1B = 1 \text{ byte} = 8b = 8\text{bits}$

The RAM loses its contents upon shutdown. Permanent information storage is achieved by the use of *hard drives*, also called *hard disks*. The hard drive contains a physical ferromagnetic disc that is magnetized in different polarities to represent bits. After the computer is turned off, the contents of the drive remain unchanged.

Files

To store different types of data and to logically separate information, hard drives are divided into files. A file is a piece of data stored on the hard drive that has a name and other *attributes*. To separate files even more and to make searching for files faster, directories (also referred to as folders) are used. A directory may contain multiple files or other directories. This is an example of a file structure:

```
.
|-- bimbows
|   |-- Logs
|   |   |-- VBox.log
|   |   '-- VBox.log.1
|   |-- bimbows.vbox
|   |-- bimbows.vbox-prev
|   '-- bimbows.vdi
'-- lmao
    |-- Logs
    |   '-- VBox.log
    |-- lmao.vbox
    |-- lmao.vbox-prev
    '-- lmao.vdi
```

The set of directories we need to follow to reach a given file, separated by the slash character (/), is called a *path*. The path starts in the *root directory* of the filesystem. Linux uses only one root directory named /.

NOTE: Many operating systems use multiple root directories. These are then named **A:/**, **B:/**, **C:/** and so on.

NOTE: Microsoft Windows uses the backslash (\) character instead of /. The backslash has a completely different meaning on Unix.

The hard drive, however, is just a continuous block of memory. The effect of files and directories is achieved by a special data structure called a *filesystem*. Many filesystem types exist. Some of the more common are FAT, VFAT (or FAT32), NTFS and the Linux ext2, ext3 and ext4. The filesystem we will use is ext4.

To make the matter even more complicated, hard drives are divided into *partitions*. A partition is a logical subdivision of the drive. The benefit of partitioning a hard drive is that different filesystems can reside on different partitions. Different partitioning schemes exist. The process of partitioning hard drives is further described in the chapter *Managing Drives*.

Representing Different Types Of Information

We now know how to represent numbers with ones and zeroes. But if you have used a computer before, you know that numbers are not the only thing we can store in the memory. Computers can process text, pictures, special databases, etc. Unix-like systems like Linux prefer encoding most data as text. Text files can be edited by the user without the need of special programs and can be manipulated with pipe-oriented programs.

ASCII

ASCII⁵ is a text encoding standard. Because the standard was developed before the *byte* was standardized, symbols in ASCII are represented by 7 bits. The most significant bit is left unset (set to zero).

ASCII (1977/1986)																
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_0	NUL 0000	SOH 0001	STX 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
1_16	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
2_32	SP 0020	! 0021	" 0022	# 0023	\$ 0024	% 0025	& 0026	' 0027	(0028) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3_48	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4_64	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5_80	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C] 005D	^ 005E	_ 005F
6_96	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7_112	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	 007C	} 007D	~ 007E	DEL 007F

Figure 1: ASCII

Let's write "Hello, world!" in hexadecimal ASCII:

```
48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21
H e l l o ,   w o r l d !
```

To insert line breaks, the line feed (newline) character is used. Its code is 0x0A. On some systems, two characters are used to represent a new line - the carriage return (0x0D) and the line feed. The ASCII standard was originally developed for teletypewriters, which worked very much like standard typewriters - the machine first needs to move the carriage back to the start of the line and then shift

⁵ASCII stands for the American Standard Code for Information Interchange

the paper downwards. Unix-like systems (like Linux) don't use the extra carriage return character inside of text files.

Unicode is an extension to ASCII. It uses the unused most significant bit to represent multi-byte characters. Having a larger amount of options, unicode contains Chinese characters, emoji, etc. The website <https://www.ssec.wisc.edu/~tomw/java/unicode.html> contains all the unicode characters.

Files on the computer can also store pictures and video. These are more difficult to encode efficiently. Therefore, multiple *file formats* exist. To quickly tell the type of a file, an *extension* is used at the end of its name. For example, *file.png* is probably a PNG image.

Files On Linux

Linux takes the concept of a file further than just storing information. Everything in Linux is a file. Microsoft Windows (or other DOS-like systems) have multiple root directories - the C drive, D drive, etc. These directories contain the files on different drives. Linux and other Unix-like systems take a different approach.

In Linux, there is only one root directory named `/`. Drives are *mounted* in different locations. When a drive is mounted into a directory, its contents appear in the given location. For example, when the drive `sda` is mounted in the directory `/home/adam/documents/`, the contents of `/home/adam/documents/` become the contents of the drive `sda`.

In Linux, everything is a file. Devices are also files - you can find them in the `/dev/` directory. Writing to a device file writes bytes to the device. Reading from a device file reads bytes from the device. In the `/proc/` filesystem, every running process on the computer has a directory and the `/tmp/` filesystem can be ⁶ located in the RAM of the computer.

Some of the important directories in Linux are:

Directory	Function
<code>/bin</code>	Programs
<code>/dev</code>	Device files
<code>/etc</code>	Configuration files (settings)
<code>/home</code>	Home directories of non-root users
<code>/opt</code>	Program files
<code>/proc</code>	Processes
<code>/root</code>	Home directory of the root user
<code>/run</code>	More temporary files
<code>/sbin</code>	Programs vital for system operation
<code>/sys</code>	Like <code>/dev</code> , but better
<code>/tmp</code>	Temporary files
<code>/usr/bin</code>	More Programs
<code>/var</code>	Size of variable length, logs, etc.

⁶This depends on your installation. In void Linux, it is.

Basic Commands

In this chapter, you will learn how to manipulate files on Linux. Log in as your normal user (not root) to open a shell.

ls And cd

Use the command `ls` to list the files in the current working directory. After logging in, the working directory is the home directory of your user. The home directory for the user `adam` would be `/home/adam/`. You can change your working directory with the `cd` command.

```
[adam@lmao ~]$ ls
test_file  data  directory
[adam@lmao ~]$ cd directory
[adam@lmao directory]$ ls
another_file  more  data
```

Basic Text Editing With Vi

After a fresh installation, your home directory will be empty. Let's create a file with some text. Open the *Vi* text editor using the command `vi`. You should see a graphical user interface:

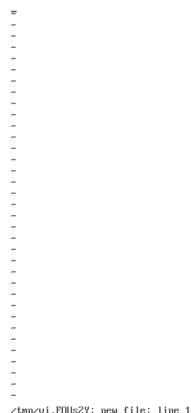


Figure 2: The Vi text editor (colors inverted to save on ink)

The vi text editor has multiple “modes”. The editor starts in the **NORMAL** mode. Here, you can use the **h**, **j**, **k** and **l** keys to move the cursor left, down, up and right respectively. By pressing **i**, you can enter the **INSERT** mode. In insert mode, pressing keys inserts characters. To exit insert mode, press the **ESC** key. If your keyboard doesn’t have the **ESC** key, hold down **CTRL** and press **[**. Pressing **:** takes you to the **COMMAND** mode. Here, you can use the command **w** to write (save) the file on the disk. The command-mode command **q** quits vi. Here is a quick summary of some of the vi commands:

Command	Action
h	move the cursor to the left
j	move the cursor down
k	move the cursor up
l	move the cursor to the right
w	move one word forward
e	move to the end of the word

Command	Action
b	move one word back
i	insert before the cursor
a	insert after the cursor
I	insert at the start of the line
A	insert at the end of the line
x	delete one character
J	join lines
dd	delete the entire line
dw	delete to the start of the next word
de	delete to the end of the current word
u	undo the last change
ESC	exit insert mode
:w	write file to disk
:w <filename>	write file to the file named
:q	quit vi
:q!	quit even though the changes weren't saved
:wq	save and quit
:h	show help

Let's crate a file with some text. Press **a** or **i** to enter insert mode. Type some text and press **ESC**. Now type **:w my_file** to write the data into a file and press enter. Then type **:q** to exit vi.

If you now use the **ls** command, you should see your file.

HINT: To open a file with vi, use **vi filename**.

Displaying Files

There are many ways to display the content of a file. The simplest one is the command **cat**. It just outputs the file to the terminal:

```
cat my_file
```

If you want to be able to scroll through a file, you can either use the vi text editor or a tool called *less* ⁷.

```
less my_file
```

Use the **q** key to quit less.

Removing Files

Use the **rm** command to remove a file. For example, To remove the file **my_file**, the following command can be used:

```
rm my_file
```

You can also use the **rm** command to remove entire directories with the files inside them. For that, you need to activate the *recursive* flag of the command. Type **rm -r my_directory** to remove the directory **my_directory**.

⁷The name less comes from the command more, which didn't support scrolling and other functionality

Creating Directories

To create a directory, use the `mkdir` command. For example, I will use `mkdir my_directory`. Now when I use `ls`, I can see the directory in the file listing. Depending on how your shell is configured, the name of the directory should be of a different color. You can now use `cd` to get inside the directory. Try using `ls` now. The newly created directory will be empty.

To get out of the directory, use `cd ..`. The `..` directory refers to the parent directory of a child directory in the file structure. There is also a directory named `.` which refers to the directory itself. That will be useful later.

HINT - use `cd` without a directory name to quickly get back into your home directory.

Command Arguments And Options

Every space-separated item that comes after the name of the command is called an *argument*. An argument can describe a filename or other input for the command. Arguments beginning with the `-` character are usually interpreted as *options* for the command. These options can change what the command does. For example, use `ls -l` in a non-empty directory to list file attributes. We will talk more about file attributes later.

```
total 520
-rw-r--r-- 1 adam adam    329 Nov  5 20:50 41f8.png
-rw-r--r-- 1 adam adam    995 Nov  5 20:01 96df.png
-rw-r--r-- 1 adam adam   3350 Nov  5 19:03 base-glibc-live-image.png
-rw-r--r-- 1 adam adam  86311 Nov  6 12:14 c651.png
-rw-r--r-- 1 adam adam   3664 Nov  6 16:03 d787.png
-rw-r--r-- 1 adam adam  30604 Nov  6 16:02 linuks.md
-rw-r--r-- 1 adam adam 380215 Nov  6 15:54 linuks.md.pdf
-rw-r--r-- 1 adam adam   8911 Nov  6 15:15 vi_ui.png
```

HINT: To insert a literal space in the argument, you have two options: either use single or double quotation marks (`'` and `"`) - `"Hello, World!"`, or *escape* the space with the backslash (`\`) - `Hello\ World!`. The backslash character is often used to escape non-printable characters, like the space or the newline. To insert a newline character, use `\n`. To insert a literal backslash, use `\\`.

You can also use the `-a` option for `ls` to list *all* files. This also displays hidden files and directories. If a file or directory is hidden, its name starts with the `.` character. This is what I get when I type `ls -a`:

```
.      41f8.png                c651.png  linuks.md.pdf
..     96df.png                d787.png  vi_ui.png
.git   base-glibc-live-image.png linuks.md
```

This explains why we couldn't see the special `.` and `..` directories! Their name started with `.`, so they were hidden!

HINT: Multiple single-character command options can be specified with one `-` character.

For example, `ls -la` will list all files while displaying their attributes, just like `ls -l -a`.

Multi-character options start with the sequence `--`. For many single-character options, there is a verbose multi-character alternative. For example, `ls --all` is the same as `ls -a`. Some options are only multi-character.

File Paths

The location of a file on the hard drive is called a path. The path to a file can be specified in multiple ways. If you just use the name of the file, the file is opened from the current working directory. If you want to use a file from a directory that is inside the current working directory, just use `my_directory/my_file`. If you want to use a file from a completely different directory regardless of your current working directory, the file path needs to begin with the `/` character. For example, `/etc/rc.conf` will be the system configuration file regardless of the current working directory.

HINT: Use the `pwd` command to print the working directory.

HINT: Using `ls` with the name of the directory lists the specified directory. For example `ls my_directory`, or `ls /etc` list the specified directories.

Selecting Multiple Files

To select every file that starts with `fi`, we can use the `*` selector:

```
[laptu-toptu]~$ ls fi*
file1  file2
```

Home Directory

Every user should have a home directory. In this directory, they should store their personal files and settings. The `~` character represents the home directory of the user. For example, if you want to access a file named `my_file` in your home directory, but don't want to use the full path, you can just use `~/my_file`. The shell prompt often also abbreviates the home directory to `~`.

Home directories for human users should follow the naming convention `/home/user_name/`. One exception is the root user, with the home directory in `/root/`.

Moving, Copying And Renaming files

To move or rename a file, use the `mv` command. The following example renames the file `source` to `destination`:

```
mv source destination
```

If the destination is a directory, the name of the file will not change and it will be moved into the directory. The `mv` command can also move and rename directories.

To copy a file, use the `cp` command. The syntax is similar to that of `mv`:

```
cp source destination
```

Because copying files is very different from only changing the references in the filesystem with the `mv` command, copying entire directories requires the recursive flag to be activated. For many commands that require this flag, it's usually set with the `-r` option. Let's copy my friends memes to my home directory:

```
cp -r memes ~
```

More On Files

File Attributes

When we use `ls -l`, we can see the file attributes. This is an example output of the command:

```
drwxr-xr-x 2 adam adam 4096 Oct 7 18:00 anj
drwxr-xr-x 2 adam adam 4096 Nov 4 19:00 bio
drwxr-xr-x 2 adam adam 4096 Oct 22 09:42 dej
drwxr-xr-x 2 adam adam 4096 Oct 7 18:00 etv
drwxr-xr-x 2 adam adam 4096 Oct 25 13:23 fyz
drwxr-xr-x 2 adam adam 4096 Oct 22 09:43 geg
-rw-r--r-- 1 adam adam 2642 Oct 6 06:42 ll
-rw-r--r-- 1 adam adam 1684 Oct 14 19:20 ms.md
-rw-r--r-- 1 adam adam 95119 Oct 14 19:22 ms.md.pdf
drwxr-xr-x 3 adam adam 4096 Nov 4 18:41 sjl
```

The line starts with a string of characters ⁸ like this: `drwxr-xr-x`. The first character is the type of the file. It can be one of the following:

Character	File Type
-	Normal file
d	Directory
l	Symbolic link
b	Block device
c	Character device
p	FIFO file
s	Socket file

NOTE: We will discuss all of the special file types in later.

Then, three groups of three characters for the *access permissions* follow. The options are `r` for read access, `w` for write access and the `x` flag allows the file to be executed as a program. The three groups define access for the owner, group and all users, respectively. These three fields are often represented by a three digit octal number (that's right, octal numbers are common in computing too!). Let's see what permissions my files have:

```
drwxr-xr-x 2 adam adam 4096 Oct 7 18:00 anj
```

The `anj` directory gives everyone read access, but only I can write to the file. The `x` flag is irrelevant for directories. The octal file permission code for this directory is 755 - 111101101 in binary. Let's see another example with a normal file this time:

```
-rw-r--r-- 1 adam adam 2642 Oct 6 06:42 ll
```

The file `ll` also allows read access for everyone, but write access only for me. The octal file permission code for this file would be 644 - 110100100. The final example is my e-mail configuration file, which stores my e-mail password in plain text. It would be dangerous to allow everyone on the system to see the contents. That's why its permissions look like this:

```
-rwx----- 1 adam adam 767 Sep 26 19:31 /home/adam/.mailrc
```

Nobody except me can read or write to this file. The octal file permission code for this file is 700 - 111000000.

The second field in the output of `ls -l` tell us how many times the file is referenced in the filesystem. When the number reaches zero, the file can be removed from the disk. This is important when using *hard links*.

Then, the owner and the group for the file follow. A group in Linux is just a set of users.

⁸A field of text data is often called a "string" of text

The fifth field is the size of the file in bytes. Notice that the size of a directory is almost always 4096, because that is the minimum size for a directory on the current filesystem. This is because the disk is divided into chunks to lower the amount of bits required to address files in the filesystem.

Then, the date and time of the last modification follow.

Modifying File Attributes

chmod changes the access rights of a file - the 9 characters that we see on the output of `ls -l`. There are multiple ways to specify these values:

The `+` and `-` operators can be used to only change a single bit of these permissions. For example, the following command sets the executable flag for `my_file`.

```
chmod +x my_file
```

If you want to make the file not executable again, just type:

```
chmod -x my_file
```

These commands will set the executable flag for every user. If you want to set the flag only for your user, use prepend the letter `u` before the `+` or `-` character:

```
chmod u+x my_file
```

The **chmod** command can also set the permissions from an octal number. For example, to make a file only accessible by the owner, use:

```
chmod 700 my_file
```

chown changes the owner of the file. The syntax is simple:

```
chown adam my_file
```

The **chgrp** command changes the group of users that owns the file:

```
chgrp wheel my_file
```

touch changes the modification time of a file to the current date and time:

```
touch my_file
```

If the file doesn't exist, it will be created. Touch is also often used to create empty files.

Symbolic And Hard Links

Linking files can be useful to create copies of the original file that also change their content upon modification of the original file. This allows making references across the filesystem and saving disk space, since a link doesn't occupy the same amount of space as the source file. There are two types of links - *symbolic links* and *hard links*. Both are created with the **ln** command. Let's create a hard link to the file `source`, named `destination`. But first, let's see the attributes of the file `source`:

```
[adam@my_machine ~]$ ls -l source
-rw-r--r-- 1 adam adam 145 Nov  7 10:10 source
```

What we're interested in is the link count: it's currently one. Let's now link the file to the file `destination`:

```
ln source destination
```

Now, let's list the files in the directory:

```
[adam@my_machine ~]$ ls -l
total 8
-rw-r--r-- 2 adam adam 145 Nov  7 10:10 destination
-rw-r--r-- 2 adam adam 145 Nov  7 10:10 source
```

The link count is now two. This means that two files refer to the same data on the disk. The advantage of hard links is that we can now remove the original file and the copy will remain:

```
[adam@my_machine ~]$ rm source
[adam@my_machine ~]$ ls -l
total 4
-rw-r--r-- 1 adam adam 145 Nov  7 10:10 destination
```

One disadvantage of hard links is that they cannot be used on directories. This is what the manual for `ln` says:

```
-d, -F, --directory
    allow the superuser to attempt to hard link directories (note:
    will probably fail due to system restrictions, even for the
    superuser)
```

Another type of a link is the *symbolic link*. Symbolic links can refer to any type of file (even the device files!). They just contain the path of the real file or directory. To create a symbolic link, activate the `-s` option of `ln`. You should use an absolute path for the destination.

```
ln -s my_directory /home/adam/directory
```

Symbolic links are special files. You can distinguish them with `ls -l`:

```
[adam@my_machine example]$ ls -l
total 4
lrwxrwxrwx 1 adam adam 16 Nov  7 10:26 destination_directory -> source_directory
drwxr-xr-x 2 adam adam 4096 Nov  7 10:25 source_directory
```

The first character on the attributes line is `l`. If you remove the original directory the link will break. Broken links should be blinking red in the output of `ls -l` (if your shell, terminal and `ls` is configured correctly).

HINT: To create symbolic links relative to the directory the link is located in, use the `-r` option.

Getting Help And Information

There is a lot to know about Linux. The more you know, the more efficient you will get. Knowing a lot of information can help form a more correct perspective of what's going on. This short chapter lists a couple of sources of information.

The Manual

GNU/Linux has a complete manual that covers almost every command. The command `man` invokes the manual. The manual pages list all of the options for a given command, its syntax and other information. It may be useful to read the manual for the manual itself (command `man man`) to know about the different sections of the manual.

NOTE: From now on, it is recommended to read the manual page for every command. This book is not a complete guide to Linux, it only tells you where to look for solutions to real problems. The manual should tell you the details.

You will often see the name of a command followed by a number in parenthesis, like `man(1)`. This tells you that the command has a manual page at the section 1 of the manual.

The manual doesn't tell you how to use the program step-by step. It just lists the options alphabetically. Sometimes it's useful to first find a step-by-step guide on the internet and then learn more about the command in the manual.

ArchWiki

The website ArchWiki is a great source of information on specific topics for every Linux user, even for those that don't use the Arch Linux distribution. It describes setting up different programs and graphical applications.

Online Forums

Online forums can be a great source of information. Before asking a question on an online forum, make sure that:

- the question hasn't been answered already (even on another forum)
- you have read the manual and it doesn't say how to fix the problem, otherwise you will be told to *RTFM* ⁹
- you provide enough information - provide logs, the commands you have used, etc.
- you at least know what is broken

Void Linux Handbook

The Void Linux handbook can be found on <https://docs.voidlinux.org/>. It only covers setting up programs and other things that are unique to the Void Linux distribution.

⁹RTFM stands for "read the forking manual"

Programs

What Are Programs?

A computer program is a sequence of instructions performed by the computer. To run a program, it needs to be loaded from the hard drive into memory. Control is then transferred to the program by the operating system. Since programs are stored on the hard drive, a program is just a file. A file becomes a program, when the executable flag (the `x` attribute) is set. To tell the program loader how to load the program into memory, executable files use a special format called **ELF**.

Almost every command is actually just a program¹⁰. Programs can be stored in multiple directories. For the shell to let us execute a program by typing its name, its directory has to be in the `$PATH` shell variable. We will talk more about shell variables later. The common directories for programs are `/bin`, `/usr/bin`¹¹, or `/sbin`¹².

Sometimes you may want to run an executable file located in the current working directory. For this, you need to use `./program`. Remember, `.` refers to the same directory as the directory it's located in. This way, we explicitly tell the shell to use the current working directory.

Shell Scripts

The *shebang* is a special string of text that, when located at the start of an executable text file, executes the text file with the command interpreter specified. This allows us to write simple shell scripts - sequences of shell commands, that behave like a normal program. An example of a shell script would be the following file:

```
#!/bin/sh
```

```
echo "Hello, world!"
```

The `#!` sequence is the shebang. After the shebang, the full path of the command interpreter follows. `/bin/sh` is the systems default command interpreter. The `echo` command just prints the specified text. To make this script executable, you will need to use `chmod +x my_script`. Now, you can use the command `./my_script` to run your first shell script! Shell scripts will become more useful when we learn about pipes and shell programming.

HINT: `/bin/sh` in Void Linux is symlinked to the *dash* shell - a strictly POSIX-compliant shell. This means, that it doesn't have some specific features of the *bash* shell that we are using for our command interpreter. For example, the `$RANDOM` variable doesn't work in *dash*. The advantage of using a strictly POSIX-compliant shell is performance, which can be orders of magnitude higher. If you're sure that you need to use *bash* for you script, just replace `/bin/sh` with `/bin/bash`.

Installing Programs - The Package Manager

Whatever you want to do with your computer, you probably want to run some more specific programs that don't come with the operating system - especially for Void Linux, a minimalist distribution that doesn't come with many big programs pre-installed. For that, you need to *install* more programs. Most Linux distributions come with a *package manager* - a program that can install, update and manage programs (or other packages like fonts, themes, etc.). Programs are downloaded from online *repositories* and installed on the machine. The package manager knows which files on the machine to change when

¹⁰Only commands that modify the properties of the shell (like `cd`) are built into the shell.

¹¹in Void Linux, `/usr/bin/` is a symlink to `/bin`

¹²`sbin` is often used for programs the system wouldn't work without, like `init`, or the shell

installing updates and can also track *dependencies* - if one package requires another package, it installs both. Void Linux comes with its own package manager called *xbps*.

Becoming The Superuser

The superuser (the `root` user) is the only user that can modify system files and directories. Therefore, he is the only user that can install packages. One way to execute programs as the superuser is to log in as `root`. But that means that you have to log out of your current session (command `exit`), or at least switch to another terminal.

Instead, the program `sudo(1)` can be used. After `sudo`, the command to be executed as the superuser follows. As an example, let's try to open the system network configuration file:

```
[adam@my_machine ~]$ cat /etc/wpa_supplicant/wpa_supplicant.conf
cat: /etc/wpa_supplicant/wpa_supplicant.conf: Permission denied
```

We are denied permission. This is because the network configuration file has the attributes `-rw-r-----` root root. Let's try opening the file as root with `sudo`:

```
[adam@my_machine ~]$ sudo cat /etc/wpa_supplicant/wpa_supplicant.conf
Password:
# Default configuration file for wpa_supplicant.conf(5).
```

```
ctrl_interface=/run/wpa_supplicant
ctrl_interface_group=wheel
eapol_version=1
ap_scan=1
fast_reauth=1
update_config=1
```

```
# Add here your networks.
```

NOTE: When using `sudo`, the password you are typing is not visible.

Installing Packages

To install a program, use the `xbps-install` command as the superuser. It's recommended to use the `-S` flag to first synchronize with the repositories and get the latest version of the package. Let's install the program `cowsay`.

```
[adam@my_machine ~]$ sudo xbps-install -S cowsay
[*] Updating repository 'https://alpha.de.repo.voidlinux.org/current/x86_64-repodata' ...
x86_64-repodata: 1705KB [avg rate: 54GB/s]
```

Name	Action	Version	New version	Download size
cowsay	install	-	3.04_3	18KB

Size to download: 18KB

Size required on disk: 41KB

Space available on disk: 168GB

Do you want to continue? [Y/n]

Press `y` and then `enter`. The output should look something like this:

```
[*] Downloading packages
```

```
cowsay-3.04_3.x86_64.xbps.sig: 512B [avg rate: 19MB/s]
cowsay-3.04_3.x86_64.xbps: 18KB [avg rate: 303MB/s]
cowsay-3.04_3: verifying RSA signature...
```

...

```
cowsay-3.04_3: installed successfully.
```

```
1 downloaded, 1 installed, 0 updated, 1 configured, 0 removed.
```

Let's see if the program has installed correctly:

```
[adam@my_machine ~]$ cowsay test
```

```
-----
< test >
-----
      \   ^__^
       \  (oo)\_______
          (__)\       )\/\
              ||----w |
               ||     ||
```

You have just installed your first program.

Updating packages

To update every package on the system, use the following command:

```
sudo xbps-install -Su
```

That's all there is to it. No rebooting, no extra configuration.

Removing Packages

To remove a package, use the command `xbps-remove`. To remove the package `cowsay`, for example (Why would you ever do that!?), use the following command:

```
sudo xbps-remove cowsay
```

If you have just uninstalled a large package, you may want to remove all of the unneeded dependencies. For that, use the `-o` flag. This scans for the orphaned packages and removes them automatically:

```
sudo xbps-remove -o
```

Searching For Packages

The `xbps-query` command can retrieve information about packages. To scan for a package by its name or description, use `xbps-query -Rs`. Let's search for `tetris`:

```
[adam@my_machine ~]$ xbps-query -Rs tetris
[-] crack-attack-1.1.14_10 Tetris Attack clone
[-] ltetris-1.2.3_1         Tetris clone using SDL
[-] quadrapassel-3.38.1_1  GNOME classic falling-block game (Tetris)
[*] vitetris-0.59.1_1      Terminal-based Tetris clone
```

The asterisk means that the package is already installed (great game, by the way).

HINT: Install the `bash-completion` package, log out and log back in. Now, when typing the name of a package or a file, you can press tab and the name will be completed for you. If nothing happens upon pressing tab, press it another time to get a list of the possible options. This is useful for typing commands quickly.

Processes And Services

A Process is a single program, loaded in memory, running on the computer. To be more precise, every process has its own *virtual memory space*¹³. Each process has a unique PID - process ID. The init process has a PID of 1. A process can have multiple threads (which share the virtual memory of their process). To achieve the effect of *multitasking*, the operating system switches between these processes.

The `ps` command lists the processes running in your terminal session. To display every process running on the system, use the command `ps -e`. This list is pretty long and not very human-readable. Let's install the program `htop` from the `htop` package. Upon running `htop`, you should see a graphical user interface:

The screenshot shows the htop interface with the following system statistics at the top:

- CPU: 3.4% (represented by 11 bars)
- Mem: 61.6M/978M (represented by 11 bars)
- Swap: 0K/0K (represented by 1 bar)
- Tasks: 26, 0 thr, 68 kthr; 1 running
- Load average: 0.00 0.00 0.00
- Uptime: 00:00:52

The main table lists running processes with the following columns: PID, USER, PRI, NI, VIRT, RES, SHR, S, CPU%, MEM%, TIME+, and Command. The processes listed include:

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
710	micu	20	0	8024	4176	3296	R	0.7	0.4	0:00.15	htop
1	root	20	0	996	4	0	S	0.0	0.0	0:00.20	runit
620	root	20	0	2352	1116	1028	S	0.0	0.1	0:00.00	runsudir -P /run/runit/runsudir/curre
626	root	20	0	2200	208	172	S	0.0	0.0	0:00.00	runsv agetty-tty6
627	root	20	0	2200	212	172	S	0.0	0.0	0:00.00	runsv sshd
628	root	20	0	2200	208	172	S	0.0	0.0	0:00.00	runsv agetty-tty4
629	root	20	0	2200	212	172	S	0.0	0.0	0:00.00	runsv acpid
630	root	20	0	2200	216	172	S	0.0	0.0	0:00.00	runsv uuidd
631	root	20	0	2200	208	172	S	0.0	0.0	0:00.00	runsv agetty-tty1
632	root	20	0	2200	208	172	S	0.0	0.0	0:00.00	runsv agetty-tty5
633	root	20	0	2200	212	172	S	0.0	0.0	0:00.00	runsv dhcpcd
634	root	20	0	2200	212	172	S	0.0	0.0	0:00.00	runsv agetty-tty2
635	root	20	0	2200	208	172	S	0.0	0.0	0:00.00	runsv udevd
636	root	20	0	2200	208	172	S	0.0	0.0	0:00.00	runsv agetty-tty3
637	root	20	0	3080	2660	2224	S	0.0	0.3	0:00.00	dhcpcd: [master] [ip4] [ip6]
638	root	20	0	5424	1600	1492	S	0.0	0.2	0:00.00	agetty tty5 38400 linux
639	root	20	0	5424	1712	1604	S	0.0	0.2	0:00.00	agetty tty3 38400 linux
640	root	20	0	5424	1644	1540	S	0.0	0.2	0:00.00	agetty tty2 38400 linux
641	root	20	0	15776	3152	2756	S	0.0	0.3	0:00.00	udev
643	root	20	0	3132	2296	1992	S	0.0	0.2	0:00.02	login -- micu
644	_uuidd	20	0	2552	412	340	S	0.0	0.0	0:00.00	uuidd -F -P
647	root	20	0	2360	220	180	S	0.0	0.0	0:00.00	acpid -f -l
648	root	20	0	5424	1620	1512	S	0.0	0.2	0:00.00	agetty tty4 38400 linux
649	root	20	0	6964	4956	4432	S	0.0	0.5	0:00.00	sshd: /usr/bin/sshd -D [listener] 0 o
651	root	20	0	5424	1660	1556	S	0.0	0.2	0:00.00	agetty tty6 38400 linux
673	micu	20	0	7116	3876	3140	S	0.0	0.4	0:00.01	-bash

At the bottom, there is a menu bar with the following options: F1 help, F2 Setup, F3 Search, F4 Filter, F5 Tree, F6 SortBy, F7 Nice, F8 Nice, F9 Kill, F10 Quit.

Figure 3: The htop user interface

NOTE: Use `q` to exit htop.

Processes in Linux have a hierarchical structure: each process (except init) has a parent process. To display the structure of processes, press the `t` key in htop. You can also use the command `ps tree` in the `psmisc` package - its output is even more readable.

¹³Even though the memory of a computer is just a linear address space, a special paging mechanism is implemented in the CPU memory management unit that remaps pages of physical memory to create a virtual memory space. This mechanism allows the program to always appear at the same position in virtual memory regardless of its physical address. The paging mechanism is also used by shared memory to allow multiple virtual memory environments to share pages of physical memory.

Killing Processes

To kill a process by its PID, use the `kill` command. The `kill` command sends the `SIGTERM` signal to a process. If the process is broken, the `TERM` signal might not always work. In that case, use `kill -9` to send the `SIGKILL` signal to a process.

To kill a process by its name, use the `pkill` command. The `pkill` command kills all the processes containing the specified sequence of characters. Alternatively, use the `k` key in `htop`.

Shell Jobs

You can run multiple processes in a single shell session. Let's say that I am using the `vi` text editor. I want to suspend `vi` and check the time using the command `date(1)`. In the `vi` text editor, I hold down `CTRL` and press `Z`. I am returned to the shell with the following message:

```
[1]+  Stopped                  vi
```

This tells us the job ID of the program. I now use the command `date` to check the time. I now know the time and want to return. I use the sequence `%1` and press enter to resume the job. Let's say that I want to calculate pi in the background while I edit my file in `vi`. I have the program `calculate-pi` in my current working directory. To run it in the background, I append `&` at the end of the command:

```
./calculate-pi &
```

I use the command `jobs` to list the running jobs:

```
[adam@micv ~]$ ./calculate-pi &
[2] 24379
[adam@micv ~]$ jobs
[1]+  Stopped                  vi
[2]-  Running                  ./calculate-pi &
```

If I now wanted, for some reason, to interact with the program `calculate-pi`, I would use `%2` to bring it to the foreground. Then, I'd press `ctrl-z` again to stop it. To get it running in the background again, I would use the command `bg %2`. To return to `vi` again, I would use `%1`.

Stopping Programs

To stop the program running in the foreground, press `ctrl-c`. For example, let's `ping(1)` the google servers:

```
ping www.google.com
```

The `ping` command will now repeatedly output the following message:

```
64 bytes from prg03s06-in-f228.1e100.net (172.217.23.228): icmp_seq=1 ttl=60 time=13.8 ms
64 bytes from prg03s06-in-f4.1e100.net (172.217.23.228): icmp_seq=2 ttl=60 time=13.7 ms
64 bytes from prg03s06-in-f4.1e100.net (172.217.23.228): icmp_seq=3 ttl=60 time=13.5 ms
64 bytes from prg03s06-in-f4.1e100.net (172.217.23.228): icmp_seq=4 ttl=60 time=13.6 ms
```

To stop pinging, I press `ctrl-c`. This sends the `SIGINT` signal to the process.

Running A Background Program Outside The Shell

If we have some jobs running in the background and want log out and leave the jobs running, we can use the `disown` command. Now, we can log out and leave the processes running. To start a process outside the shell, we can also use the following syntax:

```
(my_process &)
```

Services

Services are processes managed by `init` that run in the background. They often serve a vital system function or host some kind of a server. When the process for a service dies, a new process is started.

The `init` system for Void Linux is called `runit`. Services running in `runit` can be managed by the `sv(8)` command. `sv` has multiple sub-commands:

Command	Explanation
<code>up</code>	starts the service
<code>down</code>	stops the service
<code>once</code>	runs the service. When the process dies, a new one is not created.
<code>restart</code>	restarts a service
<code>status</code>	print the status of the service

The manual page offers more information.

Enabling And Disabling Services

Every `runit` service has its directory in `/etc/sv/`. To enable a service, use `ln -s` to create a symlink from `/etc/sv/service_name` to `/etc/runit/runsvdir/current/`. To disable the service, remove the symlink with `rm`.

Virtual Terminals

Linux hosts multiple virtual terminals. So far, we have only been using the first terminal (TTY1). You can switch to other terminals with `CTRL+ALT+F1`, `CTRL+ALT+F2`, `CTRL+ALT+F3`, and so on.

Pipes

Pipes, file redirection and other stream operations are some of the most beautiful thing to do on Unix/Linux.

Files, Streams And File Descriptors

Before we start using pipes, we have to understand how files work. When a program wants to open a file, it has to make the `open(2)` ¹⁴ system call. The system call returns a *file descriptor*. The file descriptor is a small integer representing the file. One can then use the `read(2)` and `write(2)` system calls to read and write to the file.

Three file descriptors are initialized upon program startup:

File Descriptor	Stream Name	Abbreviation	Description
0	Standard input	stdin	Data input for the program
1	Standard output	stdout	Data output of the program
2	Standard error	stderr	Warnings, user interaction, etc.

When you start a program in the shell, keyboard input is redirected to stdin and stdout and stderr are redirected to the terminal output.

By pressing `^D` (control-d), we can send the `SIGQUIT` signal to the process. The default action on `SIGQUIT` is to close the standard input. This means that no more bytes will be read and processed from stdin. In the shell, `^D` is an alternative to the built-in `exit` command.

NOTE: From now on, we will use the *caret notation* for CTRL. `ctrl + LETTER = ^LETTER`. Later, you will find out that `ctrl + character` results in a one-byte character.

File Redirection

Redirecting The Output Of A Command Into A File

We can use the `>` character to redirect the output of a command to a file:

```
ls -l > file_listing
```

The command above will list the files in the current working directory into the file `file_listing`. The contents of the file have now become the output of the command.

To append text to the end of a file, we can use the character sequence `>>`. To append some text to the end of `file_listing`, we can use the `echo(1)` command:

```
echo "Hello, World!" >> file_listing
```

Now the file contains the original text with the text “Hello, World!” at the end.

Redirecting A File Into The Input Of a Command

Let’s say that we want to number the lines in our file. The `nl` command does just that. It takes its input, adds line numbers and spits the numbered lines out on the output. Let’s first try running the command on our keyboard input:

¹⁴Manual section 2 documents system calls


```
this is a test
  1  this is a test
pretty cool!
  2  pretty cool!
```

Now, press `^D` to end the input. The program stops. Let's feed the file named `parrot_love` to its input:

```
[micv]~$ nl < parrot_love
  1  A parrot once had sex with me. I did not recognize
  2  the act as sex until it was explained to me afterward,
  3  but being stroked on the hand by his soft belly
  4  feathers was so pleasurable that I yearn for another
  5  chance. I have a photo of that act; should I go to
  6  prison for it?
  7          --Richard Stallman
```

The contents of the file have not been changed. Let's run the command again, but redirect the standard output to another file:

```
nl < parrot_love > parrot_love_numbered
```

The contents of the file `parrot_love_numbered` should now contain a numbered copy of the original file.

Redirecting Other Streams

If you know the file descriptor of the stream, you can redirect it to a file with `N>`, where `N` is the file descriptor. For example, to create a file listing of the compiler errors, which are sent to `stderr`, we can use the following command:

```
gcc bad_code.c -o bad_code 2>long_list_of_errors
```

We can also redirect a file descriptor into another file descriptor. For that, we use `A>&B`, where `A` is the source file descriptor and `B` is the destination file descriptor. This command redirects the `stderr` of the C compiler into its `stdout`:

```
gcc bad_code.c -o bad_code 2>&1
```

We can also use the sequence `&-` to close the file descriptor. Let's say that we want to hide the list of errors:

```
gcc bad_code.c -o bad_code 2>&-
```

Special Files In The `/dev` Directory

The `/dev` directory contains files `stdin`, `stdout`, and `stderr` that open as the three file descriptors described above. The `/dev` directory also contains other files with special behavior. For example, the file `/dev/null` is the same as `&-` and `/dev/zero` reads all zeroes.

Pipes

Pipes allow us to redirect the output of one command to the input of another command. There is plenty of Linux commands that allow us to process information in different ways. By joining them together, we can easily manipulate large amounts of data.

Let's number the lines on the output of `ls`. We will use the `|` (pipe) character to connect the output of `ls` into the input of `nl`:

The output will look like this:

Have you ever wanted a cow to list your files? Pipes allow you to do that easily:

```

/ AH VirtualBox VMs boat.mp4 book \
| calculate-pi dead.letter documents
| downloads example fun lmao.c
| minecraft_server minecraft_server.zip |
| notes parrot_love pictures source
\ splatit.sc timetable videos /

-----

\      ^__^
 \    (oo)\_______
      (__)\       )\/\
           ||----w |
           ||     ||

```

```
FULL_TEMP=$( curl wttr.in/$1 2>&- | # get the weather report
sed 7q | # first seven lines
grep °C | # find the temperature
sed s/\\x1b\\\\[\\[0-9\\\\;\\]*m//g | # remove escape sequences
sed s/\\s*°C.*$// | # remove the °C mark
sed s/[+\\-\\]\\[^0-9\\]/g | # remove each + or - that isn't followed by a number
sed s/^\\[0123456789+\\-\\]*// | # extract the number
sed s/^\\\\(\\[^+\\-\\]\\\\)/+\\1/ # add a + at the beginning
```

Finding Regular Expressions With Grep

Control character	Meaning
^	Start of line
\$	End of line
.	Any character
*	0 or more times
a*	the letter a zero or more times

Control character	Meaning
<code>.*</code>	anything
<code>\+</code>	1 or more times
<code>.\+</code>	at least one character
<code>[abc]</code>	a, b, or c
<code>[^abc]</code>	Anything but a, b and c

NOTE: There is a lot more to regular expressions than this. A quick cheatsheet can be found at http://stanford.edu/~wpmarble/webscraping_tutorial/regex_cheatsheet.pdf. You don't need to know everything - usually, you will learn much more by trying to achieve something real. Later on, we will discover more tools that use regular expressions, some of which are really useful for everyday computing. For example, both the programs `less` and `vi` have the `/` command, which can search for a regular expression.

Let's use `grep` to only list directories:

```
ls -l | grep ^d
```

This command will search for lines beginning with `d`, which, in the output of `ls -l`, are directories. Let's check if it works:

```
drwxr-xr-x  4 adam adam      4096 Nov  5 18:41 VirtualBox VMs
drwxr-xr-x  3 adam adam      4096 Nov  7 17:05 book
drwxr-xr-x  4 adam adam      4096 Nov  5 15:41 documents
drwxr-xr-x 11 adam adam     12288 Nov  6 14:25 downloads
```

Now, let's list files that have the executable flag set:

```
ls -l | grep ^...x..x..x
```

One slight problem is that this will also list directories. Let's create a regular expression that only matches lines that don't begin with the letter `d`:

```
ls -l | grep ^[^d]..x..x..x
```

Now, the output is only a single file:

```
-rwxr-xr-x  1 adam adam      23 Nov  7 14:04 calculate-pi
```

NOTE: Be careful about using the backslash character in command arguments. You either have to escape the backslash with another backslash (`\\+`), or put it in single-quotes (`'\+'`). You also shouldn't be sure about characters like `*`, `[`, `$`, or `^`. It's better to escape every special character than to smash your head against the keyboard for 5 hours trying to figure out which character you didn't escape in your 20KiB regular expression.

For the next example we will install the package `words-en`. This downloads the file `/usr/share/dict/words`, which contains almost every word in the English language. Each word is one line. Let's find every word that begins with the letter `b` and ends with the letter `a`:

```
cat /usr/share/dict/words | grep '^b.*a$'
```

Let's break this regular expression down: The `^` character means that we start on the beginning of a line. Then, the letter `b` follows. `.*` means any amount of any character. The regular expression ends with an `a` at the end of the line. This is the output:

```
baa
babushka
bacchanalia
```

```
bacteria
....
```

The list is pretty long. Let's join the lines together. For that, we can use the program `tr`. `tr` translates a single character from the argument on the left into the corresponding character from the argument on the right. We want to translate the newline character `\n` into a space. The pipeline would look something like this:

```
cat /usr/share/dict/words | grep '^b.*a$' | tr '\n' ' '
```

HINT: press the up arrow in the shell to go back in command history

NOTE: I once again remind you to read the manual pages for the new commands that we have discussed here.

The list is now more compact, but it's pretty long horizontally. Let's use the command `fold` to wrap the lines:

```
[micv]~$ cat /usr/share/dict/words | grep '^b.*a$' | tr '\n' ' ' | fold
baa babushka bacchanalia bacteria baklava balaclava balalaika balboa ballerina b
alsa banana bandanna barbacoa barista barracuda basilica bazooka begonia bellado
nna beluga beta betcha biretta boa bodega boga bola bologna bonanza bougainville
a bra bradycardia bravura brouhaha bulimarexia bulimia burka bursa [micv]~$
```

There is only one problem now. The newline character following the last line has also been transformed into a space. The shell prompt now doesn't begin at the start of the line. To fix this, we can simply print a newline at the end. To execute multiple commands in one line, use a semicolon (;):

```
[micv]~$ cat /usr/share/dict/words | grep '^b.*a$' | tr '\n' ' ' | fold; printf \n
baa babushka bacchanalia bacteria baklava balaclava balalaika balboa ballerina b
alsa banana bandanna barbacoa barista barracuda basilica bazooka begonia bellado
nna beluga beta betcha biretta boa bodega boga bola bologna bonanza bougainville
a bra bradycardia bravura brouhaha bulimarexia bulimia burka bursa
[micv]~$
```

NOTE: The `printf` command works almost the same way as `echo`. The main difference is that it can interpret the standard backslash escape sequences by default.

Processing Text With Sed

Sed performs different operations on lines of input. It uses single-character commands, like `s` or `d`. The single most important command in sed is the `s` command, which stands for *substitute*. The syntax is the following:

```
s/regexp/text/
```

For example, to replace every occurrence of `ae` to `æ` in the input, use the following command:

```
sed s/ae/æ/
```

Try typing random words containing `ae`. You will notice that something is wrong here:

```
[micv]~$ sed s/ae/æ/
aeiou
æiou
saelae logic
sælae logic
```

Press `^D` to quit sed. The problem here is, that by default, sed only substitutes the first match of the regular expression. To fix this, we need to set the *global* flag of the `s` command. This is achieved by appending `g` after the last slash:

```
sed s/ae/æ/g
```

Now, `saelae` logic translates to `sælæ` logic.

Let's use sed to translate the first character of every word in the input to `h`. To do this, we can use the special sequence `\<`, which represents the beginning of a word:

```
sed s/\\<./h/g
```

Notice how I used a triple slash - one double slash represents a single slash and the third slash escapes the `<` character, so it isn't interpreted as a redirection operator. The above command can be therefore also written as `'s/\<./h/g'`. Let's give it a try:

```
this is a test
hhis hs h hest
```

Let's use sed to transform a short sequence of text into second person. We can use multiple sed commands by separating them with semicolons:

```
[micv]~$ cat parrot_love | sed 's/me/you/g;s/\. I/\. You/g;s/I/you/g'
A parrot once had sex with you. You did not recognize
the act as sex until it was explained to you afterward,
but being stroked on the hand by his soft belly
feathers was so pleasurable that you yearn for another
chance. You have a photo of that act; should you go to
prison for it?
```

Commands in sed can also be prepended by a line number. In that case, they are only executed on the given line. An example could be the `d` command that deletes the line:

```
[micv]~$ cat test_file
this is a test
of the sed editor
and it works!
[micv]~$ sed 2d
this is a test
and it works!
```

Instead of a line number, a regular expression can be used. If the line matches the regular expression, the command is executed. Let's delete every line containing the word `sheeesh`:

```
sed '/she\+sh/d'
```

Processing Tables With Awk

When writing this chapter, I wanted it to have a descriptive title, just like "Sed - The Stream Editor". However, awk has so much functionality and so much use that it is hard to describe it in three to four words. Awk is another text processing language, but unlike sed, awk can deal with numbers just as well as it can deal with text.

Awk is really a full programming language, but for our needs, not every feature of awk is important. The basic syntax of an awk command is:

```
WHEN{WHAT}
```

When **WHEN** is not specified, the action is executed on every line. When **WHAT** is not specified, the default action is to print the line.

Each line that awk reads is called a *record*. A record is split into *fields*. By default, fields are separated by tabs and spaces. Let's print the second field of every line in the output of `ls -l`:

```
total 44
-rwxr-xr-x  1 adam adam  161 Sep 28 10:42 bazonga
drwxr-xr-x  3 adam adam 4096 Nov  8 06:28 book
drwxr-xr-x  3 adam adam 4096 Nov  5 15:40 documents
drwxr-xr-x  2 adam adam 4096 Nov  5 08:14 downloads
...
[laptu-toptu]~$ ls -l | awk '{print $2}'
44
1
3
3
2
...
```

There is one problem. The first line is **Total** field of `ls -l`. To fix this, we can add a condition that says that the line should only be printed once its number is greater than one. To do this, we use the built-in variable **NR** (number of records):

```
ls -l | awk '(NR>1){print $2}'
```

There is also a variable called **NF** (number of fields). To print the last field of every line, use **\$NF**:

```
ls -l | awk '(NR>1){print $NF}'
```

NOTE: Prepending the name of an awk variable with the **\$** symbol represents the n-th field. To print the second-to-last field of the line, we can use **\$(NF-1)**.

The command above may not be very exciting. Let's print the first and last field of `ls -l` to only display file access rights and the name of the file:

```
ls -l | awk '(NR>1){print $1, $NF}'
```

The best thing about awk is that if a field contains a number, we can perform mathematical operations on it. The following example demonstrates that pretty well:

```
[laptu-toptu]~$ awk '{print $1" plus "$2" equals "($1+$2)}'
12 45
12 plus 45 equals 57
800 300
800 plus 300 equals 1100
```

Because awk is a full programming language, we can make our own variables. Let's create a variable called **x** in the **BEGIN** field. The **BEGIN** field executes when awk starts. Then, we can compute the sum of the first fields of a file. At the **END**, we print the sum:

```
[laptu-toptu]~$ awk 'BEGIN{x=0} {x += $1} END{print "-----"; print x}'
12
13
56
2
-----
83
```

NOTE: The += character is often used in programming languages as a shorter version of `x = x + n`.

Press `^D` to end input and see the sum.

NOTE: Here you can notice the difference between `^C` and `^D`. `^C` will stop the program, while `^D` just tells the program that its input has ended. This is the same as coming to the end of a file.

We can change the field separator with the variable `FS`. Let's take a look at the file `/etc/passwd`. This file lists the users on the system:

```
[laptu-toptu]~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/sh
nobody:x:99:99:Unprivileged User:/dev/null:/bin/false
_dhcpd:x:999:999:_dhcpd unprivileged user:/var/db/dhcpd:/sbin/nologin
_uidd:x:998:998:_uidd unprivileged user:/var/empty:/sbin/nologin
adam:x:1000:1000:Adam Harmansky:/home/adam:/bin/bash
dbus:x:22:22:dbus unprivileged user:/var/empty:/sbin/nologin
...
```

To get a list of users and their default shells, we can use the following command:

```
[laptu-toptu]~$ cat /etc/passwd | awk 'BEGIN{FS=":"} {print $1, $NF}'
root /bin/sh
nobody /bin/false
_dhcpd /sbin/nologin
_uidd /sbin/nologin
adam /bin/bash
dbus /sbin/nologin
rtkit /sbin/nologin
...
```

This list might not be very readable. Let's use the tab character to separate the fields instead:

```
$ cat /etc/passwd | awk 'BEGIN{FS=":"} {print $1"\t"$NF}'
root    /bin/sh
nobody  /bin/false
_dhcpd  /sbin/nologin
_uidd   /sbin/nologin
adam    /bin/bash
...
```

It's still not great, but not bad either. One more thing that we could do is not to print the users that have `/sbin/nologin` as their shell. These users are probably special users created for processes. To do this, we can specify a condition:

```
$ cat /etc/passwd | awk 'BEGIN{FS=":"} ($NF!="sbin/nologin"){print $1"\t"$NF}'
root    /bin/sh
nobody  /bin/false
adam    /bin/bash
polkitd /bin/false
```

To only print the users that have `/bin/nologin` as their shell, we can use the `==` symbol:

```
$ cat /etc/passwd | awk 'BEGIN{FS=":"} ($NF=="sbin/nologin"){print $1"\t"$NF}'
_dhcpd  /sbin/nologin
_uidd   /sbin/nologin
```

```
dbus    /sbin/nologin
rtkit   /sbin/nologin
pulse   /sbin/nologin
nethack /sbin/nologin
_mlocate      /sbin/nologin
rsvlog  /sbin/nologin
```

NOTE: The == symbol tests for equivalence. The = symbol assigns a value to a variable.

A regular expression can also be used as a condition. To print the names and access rights for directories, we can use the following command:

```
[laptu-toptu]~$ ls -l | awk '/^d/{print $NF}'
book
documents
downloads
fun
notes
pictures
source
videos
```

Named Pipes

FIFO buffers, also called named pipes, are files that allow us to create multiple streams of pipes. FIFO stands for *first in, first out*. That just means that when you write data into a FIFO buffer, it comes out in the same order, just like in a pipe.

When you `write(2)` a byte into a named pipe, the program waits for another process to `read(2)` the byte. The same way, if a process wants to read a byte, it waits for another process to write the byte. Because two processes that access the file need to run at the same time, we need to use the `&` character:

```
[micv]~$ mkfifo fifo
[micv]~$ ls -l > fifo & cat fifo
[1] 4479
total 56
drwxr-xr-x  4 adam adam  4096 Nov  5 18:41 VirtualBox VMs
...
drwxr-xr-x 33 adam adam  4096 Nov  3 17:40 source
-rw-r--r--  1 adam adam  1510 Nov  7 09:18 timetable
drwxr-xr-x  2 adam adam  4096 Nov  5 15:20 videos
[1]+  Done                  ls --color=auto -l > fifo
[micv]~$ rm fifo
[micv]~$
```

To alternate the lines of output of two commands, we can use the command `paste`:

```
[micv]~$ ls -l > fifo & paste -d '\n' parrot_love fifo
[1] 5803
A parrot once had sex with me. I did not recognize
total 56
the act as sex until it was explained to me afterward,
drwxr-xr-x  4 adam adam  4096 Nov  5 18:41 VirtualBox VMs
but being stroked on the hand by his soft belly
drwxr-xr-x  3 adam adam  4096 Nov  8 15:55 book
```



```
feathers was so pleasurable that I yearn for another
drwxr-xr-x  4 adam adam  4096 Nov  5 15:41 documents
chance. I have a photo of that act; should I go to
drwxr-xr-x 11 adam adam 12288 Nov  6 14:25 downloads
prison for it?
prw-r--r--  1 adam adam    0 Nov  8 16:28 fifo
...
[1]+  Done                  ls --color=auto -l > fifo
```

NOTE: Remember to remove the FIFO file if you don't want to use it anymore.

Managing Drives On Linux

Drives on Linux are represented by *block devices* in the `/dev` directory. They are named `/dev/sda`, `/dev/sdb`, `/dev/sdc` and so on. If one drive has multiple *partitions*, just like the root drive that we have set up during installation, the different partitions are named `/dev/sda1`, `/dev/sda2`... To list the drives on the system, we can use the command `lsblk`, the output of which should look something like this:

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINTS
sda	8:0	0	8G	0	disk	
sda1	8:1	0	1M	0	part	
'sda2	8:2	0	8M	0	part	/
sr0	11:0	1	1024M	0	rom	

Mounting Drives

Let's say that I want to use the device `/dev/sr0` (the CD-ROM). First, I need to *mount* it. The best place to mount drives is the `/media` directory. To mount `/dev/sr0` to a new directory called `/media/cdrom`, I can use the following command:

```
sudo mkdir /media/cdrom
sudo mount /dev/sr0 /media/cdrom
```

NOTE: Only the root user can mount and unmount drives.

Now, I should see the contents of the drive in the directory `/media/cdrom`.

To unmount the drive, I can simply use the command `umount`:

```
sudo umount /dev/sr0
```

Partitioning Drives

Drives can (but do not have to) be divided into partitions. Just like storing files requires a filesystem, dividing a drive into partitions requires a special data structure called the *partitioning table*. Just like with filesystems, different partitioning schemes exist. A very old and standardized way to partition drives is the *Master Boot Record* - MBR, which can hold at most four partitions. We are using a more modern and advanced partitioning scheme called the *GUID Partition Table* - GPT. GPT allows for an unlimited amount of partitions.

To edit the partitions of a drive, the simple `cfdisk` utility should suffice. It is a graphical alternative to `fdisk`. Invoking `cfdisk` is easy:

```
cfdisk /dev/your_drive
```

The graphical user interface is quite self-explanatory. It is controlled with the arrow keys. Remember to type out the full word **yes** when you want to write the changes to the disk.

Creating Filesystems

To create a filesystem on a new USB drive or another blank block device, use the `mkfs` series of commands. To create a `vfat` filesystem, suitable for USB drives, use the `mkfs.vfat` command:

```
mkfs.vfat /dev/sdc3
```

There is a lot of different filesystem formats. The two most common types that you should use are `ext4` and `vfat`. `ext4` is suitable for permanent hard drives. One problem with `ext4` is that it is quite prone to breaking on hard shutdown. It also cannot be opened by default in Microsoft Windows. `VFAT`

is an extension of the **fat** filesystem. I recommend using it for smaller drives and USB sticks. **VFAT** cannot be used for the Linux root directory.

Resizing The Filesystem

To enlarge an ext4 filesystem, unmount the drive, resize the partition and use the command **resize2fs PARTITION [SIZE]**. If **resize2fs** is not provided with a size, the maximum size for the filesystem is used. This is recommended. The full set of commands would be:

Copying Disk Images With dd

The command **dd** can be used to copy the contents of files into existing files. It can write data in blocks and show statistics of the operation. It is useful for creating and burning disk images. It has a rather strange syntax. Instead of standard command parameters, it uses the equals sign to set values. This is an example of copying a file with **dd**:

```
dd if=input of=output
```

dd can be used to create a file of given size, filled with zeros:

```
dd if=/dev/zero of=file bs=512 count=2
```

This copies two records of 512 bytes, totaling 1KiB, from the device **/dev/zero**, which reads all zeros. **dd** can also apply different simple filters to files, like making the letters uppercase or convert text into different formats. Refer to the manual page for more detail.

One great use of **dd** is to create disk images. A disk image is a raw copy of the contents of a disk put into a file. Disk images are common when installing an operating system or copying entire CD drives. In fact, the command **dd** is used in the *Installation* chapter. It can be also used to copy partitions or entire disks.

NOTE: The name *disk destroyer* comes from the experience of many Linux users that have messed up their whole Linux installation. Please be careful while manipulating disks.

Shell Scripting

With the knowledge of pipes and file redirection, we can finally start creating useful scripts.

Remember that we can use the *shebang* operator at the start of an executable file (**#!/bin/sh**) to open the file in a given command interpreter:

```
#!/bin/sh
```

```
echo "Hello, world!"
```

Shell And Environment Variables

Environment Variables

Environment variables define options for programs. When a program is loaded, its *environment variables* are inherited from the parent process. The program can then read these variables during run-time. To override a variable of a program, use **VARIABLE=value command ...**:

```
MANPAGER=more man ls
```

To set the environment variables for the entire shell session, you can use the **export** command:

```
export MANPAGER=more
```

When you now use `man ls`, you will see that the pager is set to `more`, even though we didn't tell it explicitly to do anything.

Some of the more popular environment variables are `PAGER`, or `EDITOR`.

Shell Variables

Shell variables are a generalization of environment variables. They allow the shell to be a Turing-complete¹⁵ programming language. Setting a shell variable is easy:

`VARIABLE=VALUE`

To use the value of a shell variable in a command, you need to use the `$` symbol:

```
[micv]~$ options=-l
[micv]~$ ls $options
total 2668
-rw-r--r--  1 adam adam 2357569 Nov 10 17:40 2021-11-10_17-38-06.mkv
-rw-r--r--  1 adam adam  249069 Nov 10 17:40 a.mp3
drwxr-xr-x  4 adam adam   4096 Nov  5 15:41 documents
...
```

One of the most important shell variables is the `PATH` variable. It tells the shell where executable files are located. Try using the following command:

```
[micv]~$ echo $PATH
/home/adam/opt/cross/bin:/home/adam/.local/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin
```

Let's try resetting the `PATH` variable:

```
[micv]~$ export PATH=
[micv]~$ ls
bash: ls: No such file or directory
```

(Log out and back in to reset the `PATH` to the default value.)

Shell Script Arguments

In a shell script, the variables `$1`, `$2`, `$3`, and so on, represent the arguments given to the script. The following script should explain argument variables pretty well:

```
#!/bin/sh

echo "The first argument is $1"
echo "The second argument is $2"
echo "You invoked the command as $0"
echo "All of the arguments are $@"
```

Let's try running the script:

```
[micv]~$ chmod +x script
[micv]~$ ./script apple banana orange
The first argument is apple
The second argument is banana
You invoked the command as ./script
All of the arguments are apple banana orange
```

¹⁵Turing-complete means, that any program could be theoretically programmed with the language.

Comments

Every program should use comments to make it more readable for humans. To insert a comment in the shell, use the `#` character:

```
[micv]~$ # This is a comment
[micv]~$ ls # Lists files
2021-11-10_17-38-06.mkv  documents  message.html  pictures  videos
'VirtualBox VMs'        downloads  minecraft_server  script
```

Exit Values And Conditions

On the `exit(2)` system call, the program is required to specify a *return value*. The return value tells us whether the program is exiting because it has done its job successfully, or because an error has occurred. The return value of zero indicates success. A non-zero return value tells us that there was an error. A well-made program should also return the error code of the error. To check the error code of the program, use the `$?` variable:

```
[micv]~$ cat not_here
cat: not_here: No such file or directory
[micv]~$ echo $?
1
[micv]~$ cat timetable
....
[micv]~$ echo $?
0
```

To execute a command after another command has succeeded, use the `&&` sequence:

```
[micv]~$ cat file && echo "success!"
haha
success!
[micv]~$ cat error && echo "success!"
cat: error: No such file or directory
```

The `&&` sequences can be chained together. The opposite of `&&` is `||`. The `||` operator executes another command only if the first one failed. It can be translated as “do A or B, whichever works”:

```
[micv]~$ cat error || cat file
cat: error: No such file or directory
haha
[micv]~$ cat file || cat error
haha
```

The command `true` always succeeds and the command `false` always fails.

The Test Command

The command `test` fails or succeeds based on whether a specified condition is met:

```
[micv]~$ test "test" = "testt" && echo ":" || echo ":(
:(
[micv]~$ test "test" = "test" && echo ":" || echo ":(
:)
```

The command `test` can be also substituted for square brackets:

```
[micv]~$ [ "test" = "test" ] && echo ":)" || echo ":(  
:)"
```

If And Else

Just like any programming language, the shell provides basic flow-control statements. The if and else statements execute code only if a condition is met:

```
[laptu-toptu]~$ cat program  
#!/bin/sh  
  
if [ "$1" = "--help" ] || [ "$1" = "-h" ]; then  
    echo "SYNTAX: $0 [OPTIONS]"  
    echo "  --help      show this text"  
else  
    echo "Hello, world!"  
fi  
[laptu-toptu]~$ ./program  
Hello, world!  
[laptu-toptu]~$ ./program test  
Hello, world!  
[laptu-toptu]~$ ./program --help  
SYNTAX: ./program [OPTIONS]  
    --help      show this text  
[laptu-toptu]~$ ./program -h  
SYNTAX: ./program [OPTIONS]  
    --help      show this text
```

The For Loop

The shell for loop works like the for loop in python. It *iterates* through a list of items and sets a variable that represents the current item:

```
[laptu-toptu]~$ for i in banana pineapple grapefruit; do echo "I'm eating a $i"; done  
I'm eating a banana  
I'm eating a pineapple  
I'm eating a grapefruit
```

Command Substitution

The for loop is useful in combination with *command substitution* - substituting any text with the output of a command. There are two ways to perform command substitution:

1. `$(command)` puts the entire output of the command into a single string, which is then interpreted as a single argument.
2. To split the output of a command into multiple arguments, the following sequence is used¹⁶:

```
'command'
```

Let's see what types of files I have using the command `file(1)`:

```
[~]$ for i in $(ls -l | grep '^[\^d]' | awk '{print $NF}'); do file $i; done  
64: cannot open '64' (No such file or directory)
```

¹⁶I had to put the example on a new line, since the backtick character represents a code block in the language that this book is written in

bazonga: Bourne-Again shell script, ASCII text executable
lmao: fifo (named pipe)
manual: Unicode text, UTF-8 text, with overstriking
program: POSIX shell script, ASCII text executable
test.test: ASCII text
timetable: Unicode text, UTF-8 text

One very common use of the backtick sequence is the `pkg-config` command, used to automatically determine flags for the C compiler:

```
[laptu-toptu]~$ pkg-config --cflags cairo
-I/usr/include/cairo -I/usr/include/glib-2.0 -I/usr/lib64/glib-2.0/include -I/usr/include/pixman-1 -I
-I/usr/include/freetype2 -I/usr/include/libpng16
[laptu-toptu]~$ cc -c file 'pkg-config --cflags cairo' ....
... Compiling ...
```

The Case Statement

The `case` statement is used to select from multiple options. The syntax of the `case` statement in the shell is rather strange:

```
#!/bin/sh

case $1 in
    '--help')
        echo "$1: does something"
        echo "SYNTAX: $1 [OPTION]"
        echo "  --help    Display this dialog"
        echo "  --troll  We do a little trolling"
        echo "  --list-files  List Files"
        ;;
    '--troll')
        cat /home/adam/.troll
        ;;
    '--list-files')
        ls
        ;;
esac
```

Bashrc And Profile

Your home directory should contain a file named `.profile`. This file is used to export environment variables and initialize user-specific settings like the timezone. The bash shell moves most of the configuration for interactive programs into the file `~/.bashrc`. The `profile` file is mostly used only to configure the `PATH` variable. A useful thing to do in the `profile` is to add a local directory for programs, so you can call your shell scripts without linking them to `/bin/`:

```
export PATH="/home/adam/.local/bin:$PATH"
```

NOTE: it might be required to remove the file `.bash_profile` for `.profile` to take effect.

You can now try adding executable files in `~/.local/bin/`. You will not need to specify the location of the commands. This is useful to create short utility scripts to tell you weather information, bootstrap programs, etc. You can also use the local bin directory to install programs for the local user.

The `bashrc` file can be used to configure the shell prompt, editor, pager, etc:

```
# If not running interactively, don't do anything
[[ $- != *i* ]] && return

PS1='[\[\e[92m\]\h\[\e[0m\]]\[\e[93m\]\w\[\e[96m\]\$\[\e[0m\] '

export PAGER=less
export EDITOR=vi

alias ls='ls --color=auto'
alias l='ls -lhrt'
alias la='ls -a'
alias x=startx
alias info='info --vi-keys'
```

The `PS1` variable is used to configure the shell prompt. It uses ANSI escape sequences to set the color and appearance. Refer to the Wikipedia page on ANSI escape sequences for more information. The `\w` sequence represents the current working directory `\h` is the hostname, `\u` is the username and `\e` represents the escape character.

The command `alias` can be used to set settings for commands or create new “commands” that just translate to the string on the right.

Exec

The `exec` command calls the `exec(2)` system call to change the running shell to another program. When the program exits, no more programs are executed. This is useful to reduce the number of processes in the process hierarchy. This is the file `/etc/sv/wpa_supplicant/run`, which also uses `exec` to permanently redirect file descriptors:

```
#!/bin/sh
exec 2>&1
exec 1>&-
exec wpa_supplicant -iwlp2s0 -c/etc/wpa_supplicant/wpa_supplicant.conf
```


The X Graphical Environment

The X graphical environment is used to run graphical programs in *windows*. A window is a rectangle on the screen that receives input events and can be drawn into. The X graphical environment consists of a server and clients. An X client can create windows, receive input events, set the shape of the mouse cursor, change the window properties, etc. A special client called the *window manager* allows the user to control the layout of the windows on the screen.

NOTE: You may often hear the name *Xorg* or *X11*. These refer to the same thing. The name Xorg comes from the X.org foundation, which maintains the protocol¹⁷. The other name, X11 is used because version 11 has been the latest version of the X protocol for the last few decades.

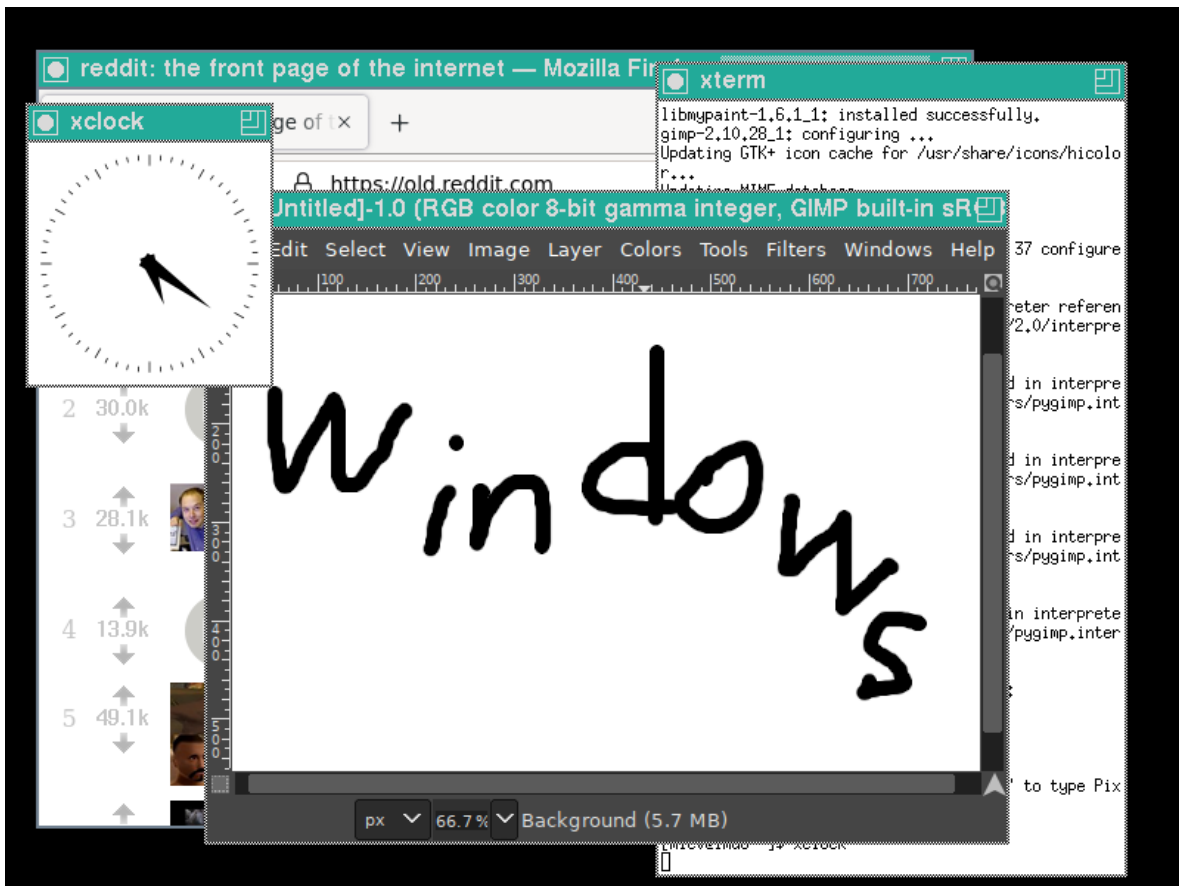


Figure 4: The X Graphical environment

The **xorg** package contains the X server with the default video and input drivers. Even though everything should run at decent performance with the default drivers, it is strongly recommended to install device-specific video drivers, which may rapidly increase performance, especially for 3D graphics and video playback. To find the right driver, you need to know the name of your graphics adapter. If you don't know it, you may try using the command `lspci`. When you know the name of your graphics adapter, consult the Void Linux Handbook on how to install the right drivers for your machine.

You will now need to install and set up a *window manager* and a *terminal emulator*. For now, we will be

¹⁷The actual software that implements the protocol is mostly provided by the XFree86 project

using the `twm` window manager, because of its simplicity. Note that `twm` is pretty outdated and should be replaced with something better later. We will discuss the options after we get X working. The terminal emulator is a graphical program that emulates the behavior of a terminal, just like the Linux virtual terminal that we have been using so far. Our terminal emulator will be the `xterm` terminal¹⁸ from the package `xterm`.

Graphical dialogs are navigated with the computer *mouse*. Moving the mouse on a physical surface moves the mouse *cursor* on the screen. The mouse has three buttons - the left button, which is the “primary” button, the middle button which has a lot of different uses, and the right button, which often reveals options in graphical programs. The middle button can also be used as a *scroll wheel*. The scroll wheel is used to navigate documents up and down. An alternative to the computer mouse is a *touchpad*. Not all touchpads have all three of the buttons. We will configure a better way to press buttons on the touchpad later.

Starting X

Before we start, make sure the following packages are installed:

1. `xorg`
2. the specific drivers for your graphics adapter
3. `xterm`
4. `twm`

When X is started with the command `startx`, it calls the script `.xinitrc` located in your home directory. This file is used to start the window manager and to specify some options. Edit the file `~/.xinitrc` with Vi and add the following line:

```
twm
```

Save and exit with the command `:wq`. Now call the command `startx`. You should see a blank screen. Press and hold the left mouse button. A menu should appear. Release the mouse over the `xterm` option. An outline of the window will appear. Move the cursor to the desired location and press the left mouse button. An xterm window should appear:

To move the window, drag it by the *title bar*. The title bar also contains two buttons: *iconify* (left) and *resize* (right). The *iconify* button shrinks the window into an *icon* window. When the icon window is pressed, the window is restored. To resize the window, press and hold the *resize* button. Now, move your cursor to the edge or corner that you want to extend/shrink. Now, moving the cursor resizes the window accordingly. Release the cursor to apply the changes.

To *close* a window, press and hold the left mouse button over a blank portion of the screen. Select the *delete* option and click on the window you want to close. There is also an option to *kill* a window. This completely kills the X client that owns the window. It may be useful when a program is not responding to the delete event.

There is plenty of graphical applications. One application that you might want to install is the Mozilla Firefox web browser from the `firefox` package. To launch a graphical application, you can just use the normal command in the terminal emulator, as with any normal program. There is only one problem with this approach - the terminal window will now be occupied by the program. It is rather impractical to have just as many terminal windows running as there are graphical applications.

NOTE: The term *application* is often used to describe graphical programs.

HINT: Exit X by exiting Twm in the main Twm menu.

¹⁸Again, many alternatives exist. You will later find a terminal emulator that suits your workflow.

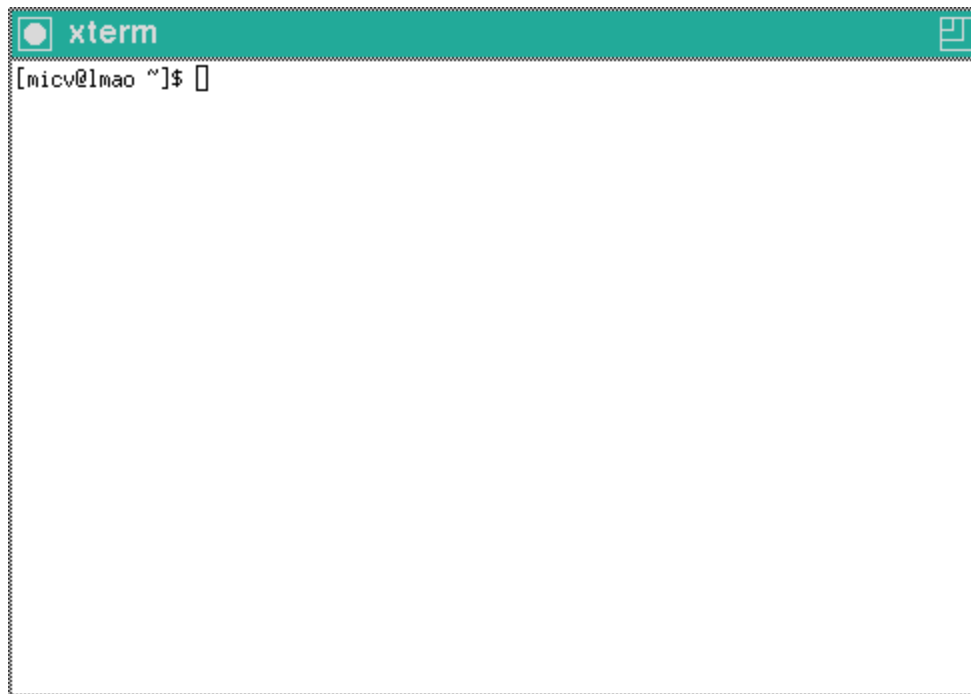


Figure 5: The XTerm terminal emulator

The X Selection And Clipboard

One thing we shouldn't continue without is the X *selection*. The selection is used to copy text between programs. To copy into the selection, press and hold the left mouse button and move the cursor. The selected text should be highlighted. To *paste* the text from the selection, use the middle mouse button. This acts just as if the text was typed by the user.

An alternate buffer called the *clipboard* exists. The clipboard can be also used to copy images and hypertext. Copying text into the clipboard works by pressing the `ctrl+c` hotkey in graphical applications. To paste text, use `ctrl+v`. However, terminal emulators cannot use this sequence, since `^C` has a very special meaning and `^V` is also used by some programs. Some terminal emulators support using `ctrl+shift+c` and `ctrl+shift+v` to copy and paste text into the clipboard, but XTerm isn't one of them. For terminal emulators, it is recommended to use the simpler **primary** selection.

Keyboard Shortcuts

To make the workflow more efficient, keyboard shortcuts can be used to quickly launch applications. The Simple X Hotkey Daemon - `sxhkd` provides a lot of options to quickly set up keyboard shortcuts. Install it from the `sxhkd` package. Edit the file `~/.config/sxhkd/sxhkdrc` (create the directory if it doesn't exist). Let's set up a few basic keyboard shortcuts:

```
mod4 + shift + Return
    xterm
mod4 + f
    firefox
```

NOTE: Mod4 refers to the *super* key - the key that usually has the Microsoft WindowsTM logo on it. If your keyboard doesn't have the key, replace `mod4` with `alt`.

We have made a keyboard shortcut to launch XTerm with Super+Shift+Return (enter) and Firefox with Super+F. Now, we need to start `sxhkd` when we start X. Edit the file `~/.xinitrc` to the following:

```
sxhkd &
twm
```

Start X. Now, pressing the corresponding keys should open the given programs.

Simple Application Launcher With Dmenu

To start any program by typing its name, we can use the program `dmenu`. `Dmenu` is a program made by the *Suckless*¹⁹ community. Due to the way it is configured (by editing the `config.h` file in the source code), it is recommended to compile `dmenu` by yourself. Run the following command to install the required packages:

```
sudo xbps-install -S curl make gcc libX11-devel libXinerama-devel libXft-devel
```

Open Firefox and head to the website <https://tools.suckless.org/dmenu>. Click on the download link. An archive file will be saved in the directory `~/Downloads` (CAPITAL D). To extract the archive, visit the directory and use the following command:

```
tar xvzf dmenu-*.tar.gz
```

`cd` in to the directory. Now, it's useful to copy the file `config.def.h` into `config.h`:

```
cp config.def.h config.h
```

You can now edit `config.h` to set the font size and colors of the menu. You can make these changes at any time, but remember to rebuild `dmenu` to see the changes applied.

To build and install the program, use the following command:

```
sudo make clean install
```

That's it. `Dmenu` is a pipe-oriented user interface program. It takes every line on the input and puts it into a searchable menu. When the user selects an option, it outputs it to the standard output. You can try it out with commands like `ls | dmenu`. `Dmenu` is useful to create custom dialog windows, like control panels, music selection, etc.

The script `dmenu_run` provided with `dmenu` allows the user to choose any program to run. Let's add it to our `sxhkdrc`:

```
mod4 + shift + Return
    xterm
mod4 + f
    firefox
mod4 + r
    dmenu_run
```

Restart X. You will now be able to launch any program with Super+R. Some simple graphical application to try are `xclock`, `xlogo`, `xcalc` (which is completely broken, use the terminal-based `calc` command instead, it is much more usable), or `xclipboard`.

Starting X On Login

To start X When the user logs in to TTY1, add the following lines at the end of your `~/.profile`:

¹⁹We will be using more programs from Suckless later.

```

if [ -z "$DISPLAY" ] && [ $(tty) == /dev/tty1 ]; then
    startx
    exit
fi
if [ -n "$BASH_VERSION" ]; then
    . $HOME/.bashrc
fi

```

Advanced Window Management With DWM

The `twm` window manager is pretty old and doesn't support many modern standards, like full-screen windows. Managing windows on `twm` may also be impractical, since windows will overlap very often. Closing windows on `twm` is a nightmare.

A great window manager that solves most of these problems is the `dwm` window manager developed by the Suckless project. `Dwm` is a *tiling* window manager - it automatically organizes windows to occupy the largest possible space on the screen. It is driven by keyboard shortcuts, which will greatly improve the speed of your workflow. Visit <http://dwm.suckless.org> for more information on `dwm`.

Setting Up DWM

Download the latest `dwm` from the Suckless website. Extract it using `tar xvzf dwm*` and `cd` into the directory. Copy the file `config.def.h` into `config.h`. Open `config.h` in `vi`.

Locate the comment `/* key definitions */`. On the line `#define MODKEY Mod1Mask`, replace `Mod1Mask` with `Mod4Mask` for `dwm` to use the Super key instead of Alt for keyboard shortcuts. If your keyboard doesn't have the Super key, you can skip this step, but remember that from now on, we will refer to the `MODKEY` as Super.

Move down until you see the beginning of the `keys` array:

```
static Key keys[] = {
```

Now, remove the lines containing the definitions for `dmenucmd` and `termcmd` (use the `vi` command `dd`), as we will be using `sxhkd` instead of the `dwm` keyboard shortcut tool to launch programs.

Locate the key definition for `MODKEY+XK_f`. This normally puts the window manager into the *floating* layout. However, we are already using Super+f to launch the web browser. Replace `MODKEY` with `MODKEY|ShiftMask` to set the keyboard shortcut to Super+Shift+f.

You are now ready to build and install `dwm`. Use the following command:

```
sudo make clean install
```

Now, edit your `xinitrc`:

```

sxhkd &
dwm

```

`Dwm` will now start when you call `startx`.

Using DWM

Since we left `sxhkd` enabled, starting programs is just as simple as before. Press Super+Shift+Enter to open a terminal window. The window will fill the screen. When another window is opened, they will split. Opening yet another window moves two of the windows to the right side of the screen. Use Super+J and Super+k to cycle window focus. To close a window, press Super+Shift+c.

The left side of the screen is the *master* area, which should contain one primary window. The right side of the screen is the *stack* area - it should contain multiple, less important windows. To change the ratio of the master and stack area, use **Super+h** and **Super+l** (it's like the cursor keys in vi). To increase the number of windows in the master area, use **Super+i** (increase) and **Super+d** (decrease).

Some Windows cannot be resized to fit the tiled mode. These windows are kept *floating* - they aren't managed by the tiling mechanism. To make a normal window floating, it can be moved or resized. To resize a floating window, hold down the super key and drag with the right mouse button. To move a floating window, do the same with the left mouse button. To put the window into tiled mode again, hold down the super key and press the middle mouse button.

Workspaces

Since tiled windows don't overlap, they occupy more screen space. Because of this, multiple *workspaces* are used. Workspaces work almost like multiple "displays" that can be switched between. To switch between workspaces, hold down super and press the key corresponding to the number of the workspace. To move a window into a different workspace, hold down Super+Shift and press the number of the workspace.

To display windows from multiple workspaces at once, hold down **Super+Ctrl** and press the number of the workspace you want to add to the visible workspaces. To show the windows from all workspaces, press **Super+0**.

To display a single window on multiple workspaces, hold down **Super+Ctrl+Shift** and press the number of the workspace that you want to add to the list of workspaces the window appears on. Press **Super+Shift+0** to display the window on all workspaces.

The ST Terminal Emulator

The Suckless terminal emulator **st** (Simple Terminal) supports modern terminal features and TrueType fonts. Install it from the Suckless website. It is configured the same way as **dwm** or **dmenu**.

Fonts

Most modern fonts use TrueType vector format. Vector fonts can be scaled to any size and use *antialiasing*.

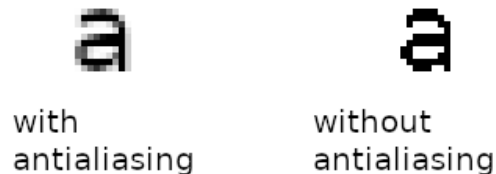


Figure 6: The effect of antialiasing

NOTE: A great place to download fonts for free is the website <https://fonts.google.com>.

To instal a TrueType font, copy the `.ttf` files in the directory `/usr/share/fonts/X11/TTF/`. Then, the name of the font can be used in the configuration files for Suckless programs or in the Firefox settings.

Audio

The ALSA sound interface can be installed from the `alsa-oss` package. To manage volume of audio devices using a terminal-based GUI, use the command `alsamixer` from the `alsa-utils` package. The command `amixer` can be used to control audio volume in scripts.

Some graphical applications require the PulseAudio interface from the `pulseaudio` package. The graphical program `pavucontrol` can be used to control the volume of PulseAudio devices. To start the PulseAudio sever with the X server, add `pulseaudio &` to your `xinitrc`:

```
sxhkd &
pulseaudio &
dwm
```

X Tweaks

Time In The DWM Status Bar

To display the time in the dwm status bar, you can edit you `xinitrc`:

```
sxhkd &
pulseaudio &

while true; do
    xsetroot -name "$(date +%H:%M)"
    sleep 10
done &

dwm
```

Touchpad Tapping

To enable touchpad tapping, locate your touchpad in the output of `xinput list`:

```
[laptu-toptu]~$ xinput list
+-Virtual core pointer                    id=2    [master pointer  (3)]
|   - Virtual core XTEST pointer          id=4    [slave  pointer  (2)]
|   - SYNA2B38:00 06CB:7FB5 Mouse         id=10   [slave  pointer  (2)]
|   - SYNA2B38:00 06CB:7FB5 Touchpad      id=11   [slave  pointer  (2)]
+-Virtual core keyboard                  id=3    [master keyboard (2)]
|   - Virtual core XTEST keyboard         id=5    [slave  keyboard (3)]
|   - Power Button                       id=6    [slave  keyboard (3)]
...
```

My touchpad is device number 11. To enable tapping for it, I can add the following line at the beginning of my `xinitrc`:

```
xinput set-prop 11 'libinput Tapping Enabled' 1
```

To emulate the left mouse button, tap on the touchpad with a single finger. To emulate the middle mouse button, tap the touchpad with two fingers. To emulate the right mouse button, tap the touchpad with three fingers. To emulate moving the mouse with a button pressed, tap the touchpad with one,

two, or three fingers and immediately put a single finger back on the touchpad and drag. To emulate the scroll wheel, put two fingers on the touchpad and move up and down.

Enable Touchpad While Typing

To enable the touchpad while keys are being pressed (which may be useful for videogames), you can add the following line at the beginning of your `xinitrc`:

```
xinput set-prop <device number> 'libinput Disable While Typing Enabled' 0
```

Faster Keyboard Repeat

To navigate terminal-based programs faster, the following line can be added at the start of your `xinitrc`:

```
xset r rate 300 50
```

Setting The Wallpaper

The wallpaper is an image at the background of the X Display. The effect of a wallpaper is achieved by drawing on the `root` window. One program capable of such think is the `feh` image viewer (package `feh`). To set the wallpaper, add the following command at the start of your `xinitrc`, or just run it in the terminal to set the wallpaper for the current X session only:

```
feh --bg-fill image.png
```

The image can be in almost any format. To set how the image is scaled into the root window, the option `--bg-fill` can be replaced with `--bg-scale`, `--bg-max`, `--bg-center`, or `--bg-tile`.

Another way to set the wallpaper is the graphical application called `nitrogen`. To use `nitrogen`, add the following line at the beginning of your `xinitrc`:

```
nitrogen --restore 2>&- &
```

Then, use the program `nitrogen` to select a wallpaper.

More Customization

GTK Themes

Some applications, like Firefox, Nitrogen, or Gimp, use the GTK UI toolkit. The appearance of GTK can be customized by installing *themes*. Many GTK themes exist. You can download them from the GTK section of www.pling.com.

To install a theme, extract the archive file that you have downloaded from the internet and copy the entire directory into the directory `~/.themes/` (create the directory if it doesn't exist). The theme will not be enabled yet.

To enable the theme for GTK3 applications, edit (or create) the file `~/.config/gtk-3.0/settings.ini`:

```
[Settings]
gtk-theme-name=vimix-dark-beryl
```

To enable the theme for GTK2 applications, add the following line to your `.xinitrc`:

```
export GTK2_RC_FILES=/home/adam/.themes/vimix-dark-beryl/gtk-2.0/gtkrc
```

To set the theme for GIMP, you need to select `system theme` in the appearance settings.

Icon Themes

To install an icon theme, extract the downloaded archive and copy, link, or move it into the directory `~/.icons/`. To enable the icon theme for GTK applications, edit the file `~/.config/gtk-3.0/settings.ini`:

```
[Settings]
gtk-theme-name=vimix-dark-beryl
gtk-icon-theme-name=Flat-Remix-Yellow-Dark
```

Cursor Themes

Cursor themes are really just modified icon themes. One simple way to apply a cursor theme system-wide is to install it like an icon theme and then rename the directory for the theme in `~/.icons/` to `default`.

Suckless Patches

Suckless programs can be modified by applying *patches* (from the website), or modifying the code. At some point, I will attach my version of `dwm` and `st` with the book. Some nice patches to try for `dwm` are `systray`, or `fullgaps`. Useful patches for `st` include `boxdraw` and `scrollback`. For `dmenu`, the `center` and `border` patches might be useful.

Going Further

This chapter will include short tutorials on different topics. The tutorials are only supposed to tell you what program to use and some basic options. It is strongly recommended to read the manual page for all of these commands and ideally find more sources online. These topics are not really sorted in any way.

General User Programs

Edit Text More Efficiently With Vim

The Vim text editor is a much better version of the `vi` editor that we have been using so far. To learn more about using vim, use the interactive tutorial that comes with the program - command `vimtutor`.

Checking The Battery Status With ACPI

To check the battery percentage on a battery-powered device, use the command `acpi`.

Displaying The Calendar

Use the command `cal` to show a simple text calendar. `cal -3` shows a three-month calendar.

The GNU Calculator

The command `calc` invokes the GNU calculator. The calculator supports variables, functions, C-like flow control, basic file I/O, and much more functionality.

Calculate The Prime Factors Of A Number

To get a list of prime factors in a number, the `factor` command can be used. To invoke it from GNU `calc`, use the following sequence:

```
!factor 123
```

Archiving Files

To store multiple files in a single file, an *archive* can be used. Multiple archive formats exist. Some can compress files, and some cannot.

ZIP The compressed ZIP archive format is very common on the internet. To create a zip archive, use the following command (install `zip` from the `zip` package):

```
zip -r archive.zip FILE [FILE2] [FILE3] ....
```

To decompress a ZIP archive, use the `unzip` command.

TAR To create a TAR archive (often called a tarball) of a directory, use the following command:

```
tar cfv tarball.tar DIRECTORY
```

To unzip a tar file, use the following options for the command `tar`:

```
tar xvf tarball.tar
```

A tarball can be compressed with `gz`:

```
gzip tarball.tar
```

This will create the file `tarball.tar.gz`. To decompress a gzipped tarball, use the `z` option for `tar`:

```
tar xvzf tarball.tar.gz
```

To Extract Any Archive , use the program `dtrx`, which will automatically figure out how to extract the archive.

Finding Files

There are two ways to find files:

1. The default `find` command with a rather baroque syntax:

```
find . -name 'file*'
```

2. The command `locate` from the package `mlocate`, which stores the information about files in a database. Make sure to run the command `updatedb` as root before using the `locate` command. The syntax is quite simple:

```
locate file
```

Displaying Images

For Xorg, many options exist:

- `feh` - simple, almost no keyboard shortcuts
- `sxiv`, `pqiv` - options to list images, many keyboard shortcuts, etc.

NOTE: To display an image outside the X graphical environment, the `fbi` command can be used.

Interacting With Websites And Downloading Files With Curl

The command `curl` can interact with URLs and websites. To get the content of my webpage, I can use the following command:

```
curl https://adamharmansky.github.io
```

To output the data into a file, the `-o` option is followed by a file name:

```
curl https://adamharmansky.github.io -o adam.html
```

To automatically determine the file name, the `-O` option can be used:

```
curl -O https://adamharmansky.github.io
```

This can be used to download any file if you know the URL.

HINT: Right-click on a URL in Firefox and press “Copy Link” to copy the URL. Middle-click to paste it into the terminal.

Measuring Download Speed The `curl` command can be used to measure the internet download speed. Large files filled with zeroes or random data can be downloaded from the internet:

```
curl https://speed.hetzner.de/1GB.bin >/dev/null
```

Getting The Current Time

To check the current system time, use the command `date`. An output format can be specified. Refer to the manual page for more information.

Setting The Brightness The Easy Way

To set the brightness as a normal user, install the program `light`. The `-S` option sets the brightness (in percent).

Playing Video

The `mpv` video player should do everything you need.

Removing A File Completely

To completely remove a file from the filesystem and destroy its contents, the `shred` command can be used. It works just like `rm`.

Editing Images

The GNU Image Manipulation Program (GIMP for short) can do almost everything you will need to manipulate images - drawing, filters, resizing, color management and much more.

To manipulate images with commands, use the command `convert` from the package `ImageMagick`. Refer to an online manual.

Browse The Web In The Terminal

If you don't have the luxury of the X windowing system, you can use the program `w3m` to browse the web from the terminal. Even though it is good for what it is, `w3m` doesn't support JavaScript and CSS, which are important components of modern websites. `W3m` can also display images using the `w3m-img` package.

Counting Words In A Document

The `wc` command shows three columns displaying the line count, word count and byte count of a file, respectively. It can also make a sum of all the specified files.

Reading PDF Files

The PDF document format is used a lot on the internet. One of the best document readers which support PDF is `zathura`. Download the package `zathura` and `zathura-pdf-mupdf`. Now `Zathura` should be able to open PDF files.

To open pdf files outside of X11, use the program `fbpdf`.

Funny Eyes That Follow The Cursor

Try the command `xeyes`.

Scripting Programs

Displaying A Message Window

To display a simple message on X11, the command `xmessage` can be used.

Getting The Contents Of The X Selection

The program `xclip` can do almost anything with the X clipboard and selection.

Wait For A Given Amount Of Time

To wait for a given amount of time in a shell script, use the `sleep` command which takes the number of seconds sleep. The number can be decimal, like 0.5.

Pipe-Oriented Programs

Encoding And Decoding Data In Base64 And Base32

To encode data into base 64, use the command `base64`. `base64 -d` decode data.

The command `base32` works the same, except that it uses base 32.

Getting N Lines From The Beginning Or The End Of A File

Use the command `head` to get 10 lines from the beginning of a file. The command `tail` does the opposite. The `-n` option specifies the number of lines. More information can be found in the manual.

Getting The Hexadecimal Contents Of A File

The command `xxd` makes a *hexdump* of a file of the standard input. It can also convert an edited hexdump back into a file.

Shuffling The Lines In A File

To shuffle the lines in a file or a pipeline, use the `shuf` command.

Sort Lines Of Text

To sort the lines of text, use the `sort` command. More information in the manual.

System Administration

Stopping And Rebooting The Computer

To stop the machine, the command `halt` can be issued by the superuser. To restart (reboot) the machine, the command `reboot` is used.

Putting The Computer To Sleep To put the computer into the sleep state, use the command `zzz`. The command `ZZZ hibernates` the computer - it saves the memory into the hard drive and shuts down. Upon starting, the system should resume operation.

Changing The Default Shell For A User

You can change your default with `chsh`. Some nice shells to try are the `fish` shell from the package `fish`, which has really good tab-completion, or `zsh`, which is really extensible.

Checking The Disk Usage

To check for the available space on all of the mounted hard drives, the command `df` can be used.

To see the disk usage of a single file, the `du` command can be used. It outputs the size in blocks, which may not be very human-readable. To make the output human-readable, use the `-h` option.

Reading The Kernel Logs

To see what is going on in the system, and to troubleshoot problems with devices, the command `dmesg` can be used as the superuser. To keep waiting for more events instead of closing `dmesg` after printing out the logs, the `-w` option can be used.

Filesystem Check

To locate and fix problems in the filesystem, the `fsck` command can be used.

Getting And Setting The System Hostname

Use the command `hostname`.

Getting The IP Address

To see the IP address of your computer for different adapters, use the `ip` command with the `addr` subcommand:

```
ip a
```

CPU Info

To get information about the CPU, use the command `lscpu`.

Memory Usage

To see the memory usage, use the `free` command, ideally with the `-h` option, which gives you the output in human-readable units.

Remotely Access Another Computer With SSH

To access a machine running an SSH server, use the command SSH:

```
ssh username@MACHINE_IP_ADDRESS_OR_URL
```

To host an ssh server, the `sshd` service needs to be enabled.

Redirect The Intermediate Output Of A Pipeline

The command `tee` copies its input both into `stdout` and the files specified. You can use it in a pipeline like this:

```
... | tee output | ...
```

Remove Repeating Lines In A File

Use the command `uniq` to remove repeating lines in a file or a pipeline.

See How Long The Machine Has Been Running

The command `uptime` will tell you statistics of how many people have been using the machine, what the average load was and how long the machine has been running.

Create A User Or Group

To add a new user, use the command `useradd`. To delete a user, use the command `userdel`. To create and remove a group, use `groupadd` and `groupdel`.

Add A User To A Group

To add a user to a group, use the following command:

```
usermod adam -a -G my_group
```

Edit Superuser Access

To edit the access rights for users with the `sudo` command, use the command `sudo visudo`. The `sudoers` file allows you to, for example, allow users to run certain commands like `reboot`, `halt`, or `zzz` without superuser access. The manual page for `sudoers(5)` should contain more than enough information on how to set up this file.

Remove Old Versions Of The Kernel

When updating the system, older versions of the kernel are kept. To remove them, issue the following command:

```
sudo vkgpurge rm all
```

Open A Root Shell

To open a root shell, use `sudo` with no arguments.

Connecting To WiFi Networks

To connect to a WiFi network, append the output of the program `wpa_passphrase` into `/etc/wpa_supplicant/wpa_supplicant.conf`:

```
wpa_passphrase network_name secret_password > /etc/wpa_supplicant/wpa_supplicant.conf
```

Monitor Configuration

The command `xrandr` can be used to configure displays - set the video modes, change the layout of the displays, etc.