

Getting around on the command line

Thom Wiggers

Digital Security, Radboud University

COVID-19 online edition

~ \$ whoami

- ▶ Thom Wiggers
- ▶ PhD candidate with Digital Security
 - ▶ Research into applying post-quantum cryptography
 - ▶ Supervisor: Peter Schwabe
- ▶ Teaching *Hacking in C* next quarter
- ▶ <https://thomwiggers.nl/>

Table of Contents

Linux

Getting around on the command line

Common commands showcase

Exercise

Special I/O on Linux

System administration

Exiting vim

Table of Contents

Linux

Getting around on the command line

Common commands showcase

Exercise

Special I/O on Linux

System administration

Exiting vim

Kernel vs Userspace

▶ Kernel

Kernel vs Userspace

- ▶ Kernel
 - ▶ Interacts with hardware

Kernel vs Userspace

- ▶ Kernel
 - ▶ Interacts with hardware
 - ▶ Implements access control

Kernel vs Userspace

- ▶ Kernel
 - ▶ Interacts with hardware
 - ▶ Implements access control
 - ▶ Doesn't do much on its own

Kernel vs Userspace

- ▶ Kernel
 - ▶ Interacts with hardware
 - ▶ Implements access control
 - ▶ Doesn't do much on its own
- ▶ Userspace

Kernel vs Userspace

- ▶ Kernel
 - ▶ Interacts with hardware
 - ▶ Implements access control
 - ▶ Doesn't do much on its own
- ▶ Userspace
 - ▶ Anything that's outside of the kernel

Kernel vs Userspace

- ▶ Kernel
 - ▶ Interacts with hardware
 - ▶ Implements access control
 - ▶ Doesn't do much on its own
- ▶ Userspace
 - ▶ Anything that's outside of the kernel
 - ▶ User utilities like `bash`, `ls`, `gedit`, `vim`, `factorio`

Kernel vs Userspace

- ▶ Kernel
 - ▶ Interacts with hardware
 - ▶ Implements access control
 - ▶ Doesn't do much on its own
- ▶ Userspace
 - ▶ Anything that's outside of the kernel
 - ▶ User utilities like `bash`, `ls`, `gedit`, `vim`, `factorio`
 - ▶ System administration and services like `systemd`, `sudo`, `passwd`

Kernel vs Userspace

- ▶ Kernel
 - ▶ Interacts with hardware
 - ▶ Implements access control
 - ▶ Doesn't do much on its own
- ▶ Userspace
 - ▶ Anything that's outside of the kernel
 - ▶ User utilities like `bash`, `ls`, `gedit`, `vim`, `factorio`
 - ▶ System administration and services like `systemd`, `sudo`, `passwd`
 - ▶ Graphics stuff like X11, Wayland, Gnome, GDM

Kernel vs Userspace (cont.)

Kernel: Linux

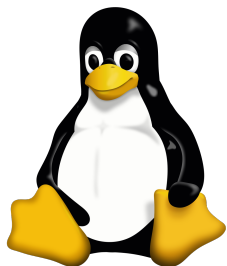


Figure: Tux

System management, access control, file systems, drivers, power management, ...

Kernel vs Userspace (cont.)

Kernel: Linux



Figure: Linus Torvalds

Kernel vs Userspace (cont.)

Kernel: Linux



Figure: Linus Torvalds being frustrated with NVIDIA

Userspace: GNU



Figure: GNU logo

GCC, Emacs, Coreutils, GPL license, GTK, Gnupg, Bash, ...

Kernel vs Userspace (cont.)

Kernel: Linux



Figure: Linus Torvalds being frustrated with NVIDIA

Userspace: GNU

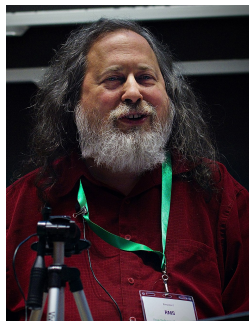


Figure: Richard Stallman

Kernel vs Userspace (cont.)

Kernel: Linux



Figure: Linus Torvalds being frustrated with NVIDIA

Userspace: GNU



Figure: Richard Stallman insists on calling Linux GNU/Linux

Kernel vs Userspace (cont.)

Kernel: Linux



Figure: Linus Torvalds being frustrated with NVIDIA

Userspace: GNU



Figure: Richard Stallman picks stuff from between his toes and smells it

Kernel vs Userspace (cont.)

Kernel: Linux



Figure: Linus Torvalds being frustrated with NVIDIA

Userspace: GNU



Figure: Richard Stallman picks stuff from between his toes and smells it, and eats it

Kernel vs Userspace (cont.)

Kernel: Linux



Figure: Linus Torvalds being frustrated with NVIDIA

Userspace: GNU



Figure: Richard Stallman quit after he defended someone related to Jeffrey Epstein stuff

Linux Distribution

- ▶ Usually you obtain Linux bundled with a userspace and bunch of programs as a *Linux distribution*



Figure: Ubuntu logo

Linux Distribution

- ▶ Usually you obtain Linux bundled with a userspace and bunch of programs as a *Linux distribution*
- ▶ Most popular one: Ubuntu



Figure: Ubuntu logo

Linux Distribution

- ▶ Usually you obtain Linux bundled with a userspace and bunch of programs as a *Linux distribution*
- ▶ Most popular one: Ubuntu
- ▶ Popular commercial one: Redhat Enterprise Linux



Figure: Ubuntu logo

Linux Distribution

- ▶ Usually you obtain Linux bundled with a userspace and bunch of programs as a *Linux distribution*
- ▶ Most popular one: Ubuntu
- ▶ Popular commercial one: Redhat Enterprise Linux
- ▶ Many distribution vendors ship a desktop and a server variants



Figure: Ubuntu logo

Linux Distribution

- ▶ Usually you obtain Linux bundled with a userspace and bunch of programs as a *Linux distribution*
- ▶ Most popular one: Ubuntu
- ▶ Popular commercial one: Redhat Enterprise Linux
- ▶ Many distribution vendors ship a desktop and a server variants
- ▶ Other (popular) distributions: [Debian](#), [Arch Linux](#), [Fedora](#), [Kali](#), [CentOS](#)



Figure: Ubuntu logo

Linux Distribution

- ▶ Usually you obtain Linux bundled with a userspace and bunch of programs as a *Linux distribution*
- ▶ Most popular one: Ubuntu
- ▶ Popular commercial one: Redhat Enterprise Linux
- ▶ Many distribution vendors ship a desktop and a server variants
- ▶ Other (popular) distributions: [Debian](#), [Arch Linux](#), [Fedora](#), [Kali](#), [CentOS](#)
- ▶ if you're a masochist try installing ["Linux from Scratch"](#)



Figure: Ubuntu logo

Table of Contents

Linux

Getting around on the command line

Common commands showcase

Exercise

Special I/O on Linux

System administration

Exiting vim

The shell

- ▶ `/bin/bash`: **B**ourne **a**gain **s**hell
 - ▶ Improved version of the classic Bourne shell `/bin/sh`
 - ▶ Other shells exist, but we'll use the default for now.
 - ▶ Thom uses something fancy himself but they all work more-or-less the same

What is a shell

- ▶ Program to interact with your computer and the software on



Figure: Different kind of shell

What is a shell

- ▶ Program to interact with your computer and the software on
- ▶ Graphical interface is also such a program!



Figure: Different kind of shell

What is a shell

- ▶ Program to interact with your computer and the software on
- ▶ Graphical interface is also such a program!
- ▶ Typically synonymous with command line though



Figure: Different kind of shell

The shell prompt

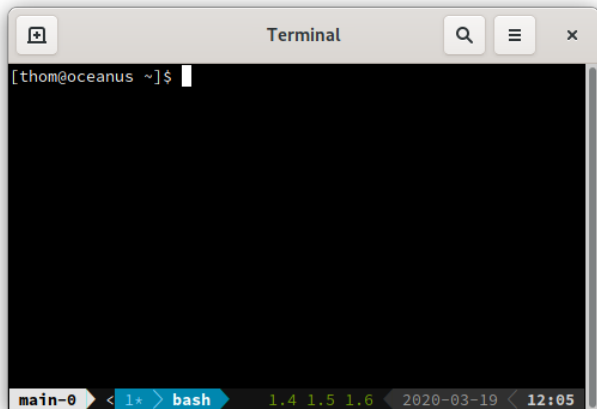
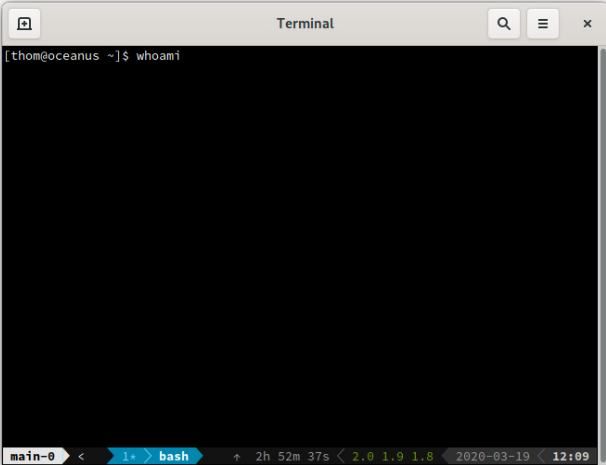


Figure: The shell prompt

Running programs

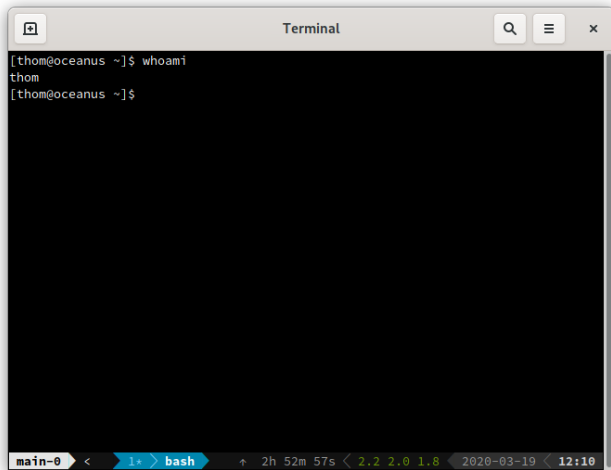


A screenshot of a terminal window titled "Terminal". The window has a title bar with a search icon, a menu icon, and a close icon. The terminal content shows the prompt "[thom@oceanus ~]" followed by the command "whoami". The output of the command is not visible. At the bottom of the terminal, there is a status bar with the following information: "main-0", a left arrow, "1", a right arrow, "bash", an up arrow, "2h 52m 37s", a left arrow, "2.0 1.9 1.8", a right arrow, "2020-03-19", and "12:09".

```
[thom@oceanus ~]$ whoami
```

Figure: Running whoami

Running programs

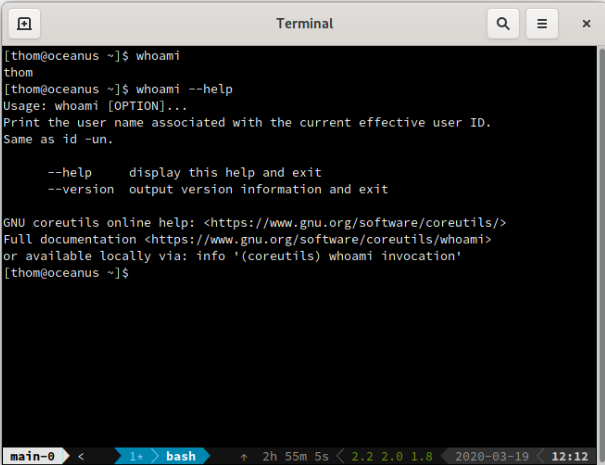


```
[thom@oceanus ~]$ whoami
thom
[thom@oceanus ~]$
```

The image shows a terminal window titled "Terminal" with a search icon, a menu icon, and a close icon in the title bar. The terminal content shows the command `whoami` being executed, resulting in the output `thom`. The prompt `[thom@oceanus ~]$` is visible before and after the command. At the bottom of the terminal, there is a status bar with the text `main-0`, a left arrow, `1*`, `bash`, a right arrow, `2h 52m 57s`, `2.2 2.0 1.8`, `2020-03-19`, and `12:10`.

Figure: Running whoami

Running programs



```
[thom@oceanus ~]$ whoami
thom
[thom@oceanus ~]$ whoami --help
Usage: whoami [OPTION]...
Print the user name associated with the current effective user ID.
Same as id -un.

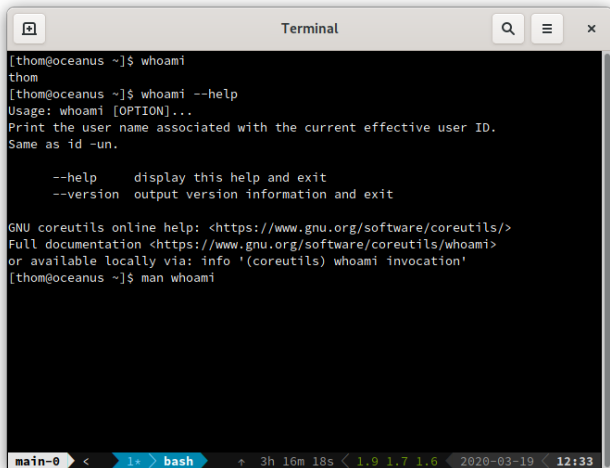
    --help      display this help and exit
    --version   output version information and exit

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/whoami>
or available locally via: info '(coreutils) whoami invocation'
[thom@oceanus ~]$
```

The image shows a terminal window titled "Terminal" with standard macOS window controls (search, menu, close). The terminal content shows the execution of the 'whoami' command, which returns the username 'thom'. A second execution of 'whoami --help' displays the usage information and options for the command. The terminal status bar at the bottom shows 'main-0', '1*' (current directory), 'bash' (shell), and system information including '2h 55m 5s', version '2.2 2.0 1.8', date '2020-03-19', and time '12:12'.

Figure: Running whoami

Running programs



```
[thom@oceanus ~]$ whoami
thom
[thom@oceanus ~]$ whoami --help
Usage: whoami [OPTION]...
Print the user name associated with the current effective user ID.
Same as id -un.

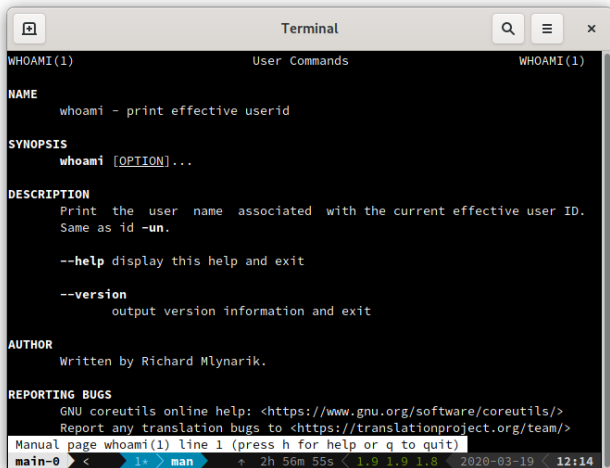
    --help      display this help and exit
    --version   output version information and exit

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/whoami>
or available locally via: info '(coreutils) whoami invocation'
[thom@oceanus ~]$ man whoami
```

The terminal window shows the execution of the `whoami` command, which outputs the user name `thom`. It then shows the help text for `whoami`, including usage instructions and options like `--help` and `--version`. The terminal also displays GNU coreutils online help and full documentation links. The prompt `[thom@oceanus ~]$ man whoami` is visible at the bottom.

Figure: Running whoami

Running programs



```
Terminal
WHOAMI(1)                                User Commands                                WHOAMI(1)

NAME
  whoami - print effective userid

SYNOPSIS
  whoami [OPTION]...

DESCRIPTION
  Print the user name associated with the current effective user ID.
  Same as id -un.

  --help display this help and exit

  --version
        output version information and exit

AUTHOR
  Written by Richard Mlynarik.

REPORTING BUGS
  GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
  Report any translation bugs to <https://translationproject.org/team/>
  Manual page whoami(1) line 1 (press h for help or q to quit)
main-0 < 1* > man + 2h 56m 55s < 1.9 1.9 1.8 < 2020-03-19 < 12:14
```

Figure: Running whoami

The manual

- ▶ There is a manual that should be available for most programs
- ▶ usage: `man <topic>`
- ▶ Manual for the manual: `man man`
- ▶ Googling for “man something” usually finds these man pages as well

Table of Contents

Linux

Getting around on the command line

Common commands showcase

Exercise

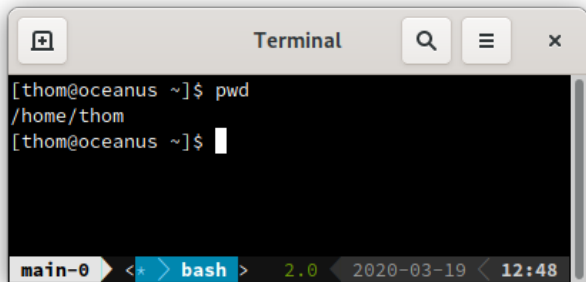
Special I/O on Linux

System administration

Exiting vim

Where am I: pwd

pwd: Print Working Directory



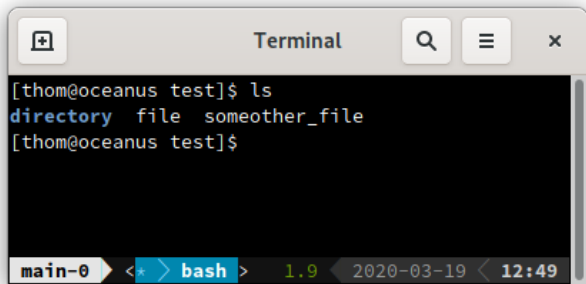
```
[thom@oceanus ~]$ pwd
/home/thom
[thom@oceanus ~]$
```

The image shows a terminal window titled "Terminal" with standard macOS window controls (close, search, menu). The terminal content shows a user named "thom" on a machine named "oceanus" in the home directory "~". The user enters the command "pwd", and the terminal outputs "/home/thom". The prompt returns to "[thom@oceanus ~]\$". At the bottom of the terminal, there is a status bar with "main-0", "bash", "2.0", "2020-03-19", and "12:48".

Figure: pwd

What is here: ls

ls: list files in folder



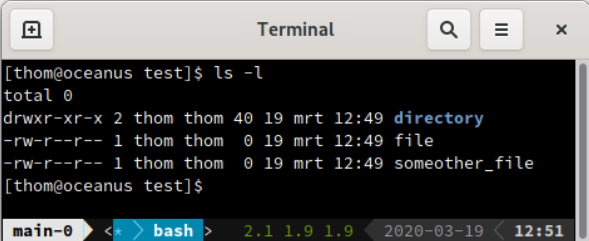
```
[thom@oceanus test]$ ls
directory file someother_file
[thom@oceanus test]$
```

The image shows a terminal window titled "Terminal" with standard macOS window controls (close, search, menu). The terminal content shows a user named "thom" at a host named "oceanus" in a directory named "test". The user has entered the command "ls", and the output is "directory file someother_file". The terminal status bar at the bottom shows "main-0", "bash", version "1.9", date "2020-03-19", and time "12:49".

Figure: ls

What is here with more detail: `ls -l`

`ls -l`: list files in folder with sizes, permissions, access times



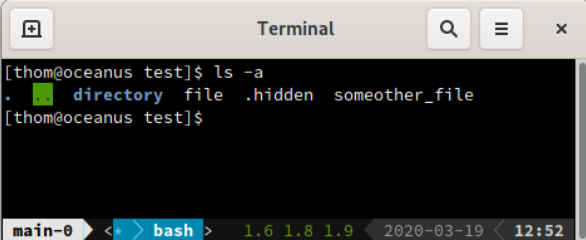
```
[thom@oceanus test]$ ls -l
total 0
drwxr-xr-x 2 thom thom 40 19 mrt 12:49 directory
-rw-r--r-- 1 thom thom  0 19 mrt 12:49 file
-rw-r--r-- 1 thom thom  0 19 mrt 12:49 someother_file
[thom@oceanus test]$
```

The image shows a terminal window titled "Terminal" with a search icon, a menu icon, and a close icon. The terminal content shows the command `ls -l` being executed in a directory named `test`. The output lists three items: a directory named `directory` with permissions `drwxr-xr-x`, size `40`, and access time `12:49`; a file named `file` with permissions `-rw-r--r--`, size `0`, and access time `12:49`; and a file named `someother_file` with permissions `-rw-r--r--`, size `0`, and access time `12:49`. The terminal prompt is `[thom@oceanus test]$`. At the bottom of the terminal window, there is a status bar showing `main-0`, `< * > bash >`, version numbers `2.1 1.9 1.9`, and a date/time stamp `2020-03-19 < 12:51`.

Figure: `ls -l`

What is here with hidden files: `ls -a`

`ls -a`: list files in folder including hidden files and folders

A screenshot of a macOS Terminal window titled "Terminal". The window shows a shell prompt "[thom@oceanus test]\$ ls -a" followed by the output: ". directory file .hidden someother_file". The prompt "[thom@oceanus test]\$" is repeated on the next line. At the bottom of the terminal, there is a status bar with "main-0", "bash", and version numbers "1.6 1.8 1.9", along with the date "2020-03-19" and time "12:52".

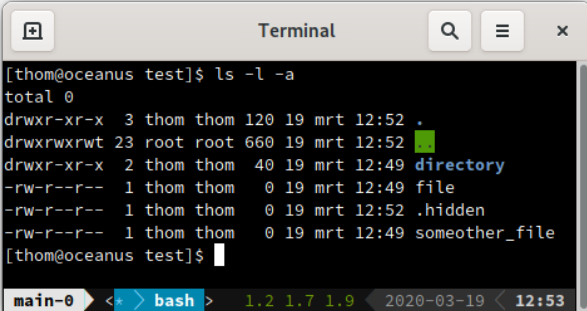
```
[thom@oceanus test]$ ls -a
. directory file .hidden someother_file
[thom@oceanus test]$
```

Figure: `ls -a`

Hidden files start with a dot. Two special filenames are `.` **current directory** and `..` **parent directory**.

What is here, in detail, with hidden files: `ls -l -a`

`ls -l -a`: list files in folder, in detail, including hidden files and folders



```
[thom@oceanus test]$ ls -l -a
total 0
drwxr-xr-x  3 thom thom 120 19 mrt 12:52 .
drwxrwxrwt 23 root root 660 19 mrt 12:52 [red square]
drwxr-xr-x  2 thom thom  40 19 mrt 12:49 directory
-rw-r--r--  1 thom thom   0 19 mrt 12:49 file
-rw-r--r--  1 thom thom   0 19 mrt 12:52 .hidden
-rw-r--r--  1 thom thom   0 19 mrt 12:49 someone_file
[thom@oceanus test]$
```

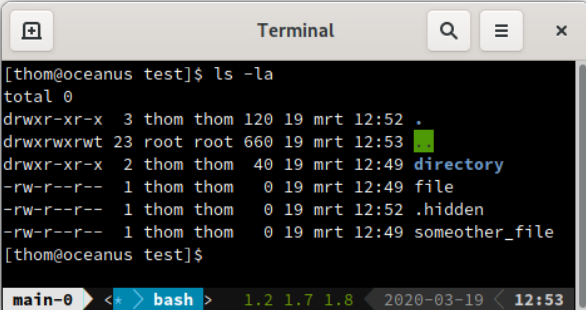
Figure: `ls -l -a`

Can also be written as `ls -la`

What is here, in detail, with hidden files: `ls -la`

`ls -la`: list files in folder, in detail, including hidden files and folders

Uses the abbreviated syntax for command line flags



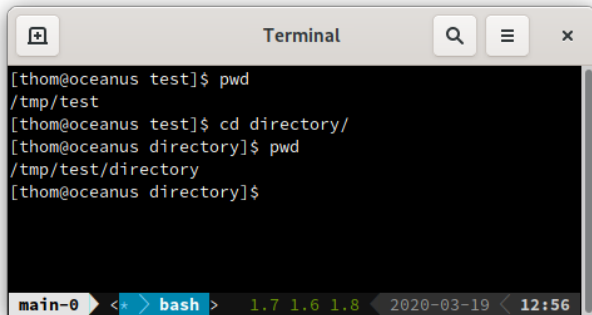
```
[thom@oceanus test]$ ls -la
total 0
drwxr-xr-x  3 thom thom 120 19 mrt 12:52 .
drwxrwxrwt 23 root  root 660 19 mrt 12:53 .
drwxr-xr-x  2 thom thom  40 19 mrt 12:49 directory
-rw-r--r--  1 thom thom   0 19 mrt 12:49 file
-rw-r--r--  1 thom thom   0 19 mrt 12:52 .hidden
-rw-r--r--  1 thom thom   0 19 mrt 12:49 someone_file
[thom@oceanus test]$
```

The terminal window shows the output of the `ls -la` command. The output lists the current directory and its contents in a long format. The first line is `total 0`. The following lines show the permissions, number of links, owner, group, size, month, day, and time for each file or directory. The files listed are `.`, `.`, `directory`, `file`, `.hidden`, and `someone_file`. The terminal window title is "Terminal" and the prompt is `[thom@oceanus test]$`. The terminal window also shows the shell name `main-0`, the shell `bash`, and the version numbers `1.2 1.7 1.8`. The date and time are `2020-03-19` and `12:53`.

Figure: `ls -la`

Moving around: cd

`cd dir`: change into directory `dir`

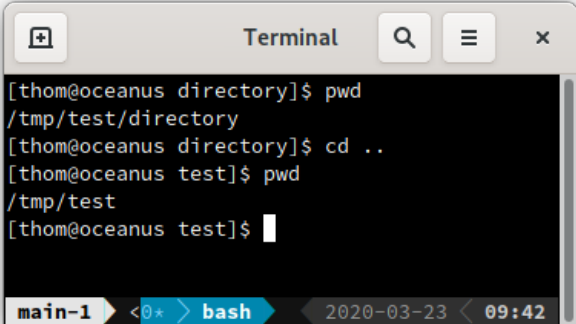
A screenshot of a macOS Terminal window titled "Terminal". The window has a title bar with a search icon, a menu icon, and a close icon. The terminal content shows a user named "thom" at a host named "oceanus" in a directory named "test". The user runs the command "pwd", which returns "/tmp/test". Then, the user runs "cd directory/", and the terminal returns the prompt "[thom@oceanus directory]". The user runs "pwd" again, and the terminal returns "/tmp/test/directory". Finally, the user runs "cd" without arguments, and the terminal returns the prompt "[thom@oceanus directory]". At the bottom of the terminal window, there is a status bar with "main-0" on the left, a blue arrow pointing right with "bash" in the middle, and "1.7 1.6 1.8" on the right. Below the status bar, the date "2020-03-19" and time "12:56" are displayed.

```
[thom@oceanus test]$ pwd
/tmp/test
[thom@oceanus test]$ cd directory/
[thom@oceanus directory]$ pwd
/tmp/test/directory
[thom@oceanus directory]$
```

Figure: Changing into directory

Moving to the parent directory: `cd ..`

`cd ..`: move to parent directory.

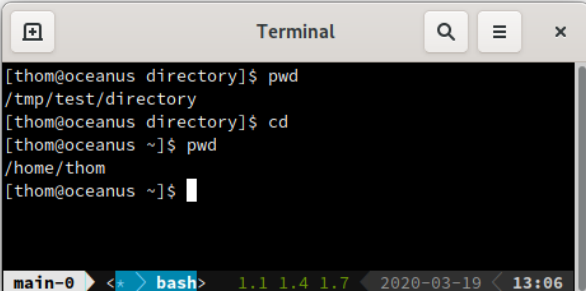
A terminal window titled "Terminal" with standard macOS window controls (close, zoom, search, menu). The terminal shows a sequence of commands and their outputs: 1. Command: `pwd`, Output: `/tmp/test/directory`. 2. Command: `cd ..`. 3. Command: `pwd`, Output: `/tmp/test`. 4. The prompt `[thom@oceanus test]$` is shown with a cursor. The terminal's status bar at the bottom displays `main-1`, `<0*`, `bash`, `2020-03-23`, and `09:42`.

```
[thom@oceanus directory]$ pwd
/tmp/test/directory
[thom@oceanus directory]$ cd ..
[thom@oceanus test]$ pwd
/tmp/test
[thom@oceanus test]$
```

Figure: Changing into parent directory

Moving to your home dir: `cd`

`cd`: change directory to home



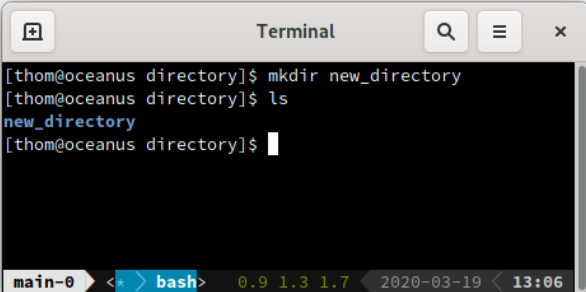
```
[thom@oceanus directory]$ pwd
/tmp/test/directory
[thom@oceanus directory]$ cd
[thom@oceanus ~]$ pwd
/home/thom
[thom@oceanus ~]$
```

The terminal window shows the user 'thom' at host 'oceanus' in the directory '/tmp/test/directory'. They execute 'pwd' which returns '/tmp/test/directory'. Then they execute 'cd' to move to their home directory. A second 'pwd' command returns '/home/thom'. The terminal status bar at the bottom shows 'main-0', 'bash', version '1.1 1.4 1.7', date '2020-03-19', and time '13:06'.

Figure: Changing into `/home/thom`

Creating a new folder: mkdir

`mkdir dir`: Create folder `dir`

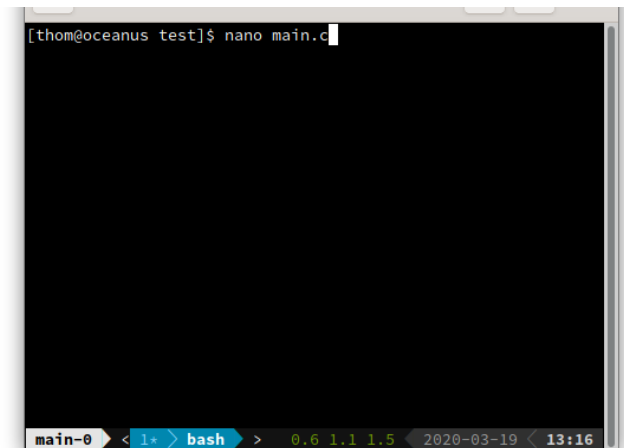
A terminal window titled "Terminal" with standard macOS window controls (close, zoom, search). The terminal shows a user named "thom" at a host named "oceanus" in a "directory". The user enters the command "mkdir new_directory", followed by "ls", which outputs "new_directory". The prompt returns to the shell. At the bottom, a status bar shows "main-0", "bash", version numbers "0.9 1.3 1.7", and a timestamp "2020-03-19 13:06".

```
[thom@oceanus directory]$ mkdir new_directory
[thom@oceanus directory]$ ls
new_directory
[thom@oceanus directory]$
```

Figure: Create new_directory

Editing a file

`nano file`: Edit file using nano

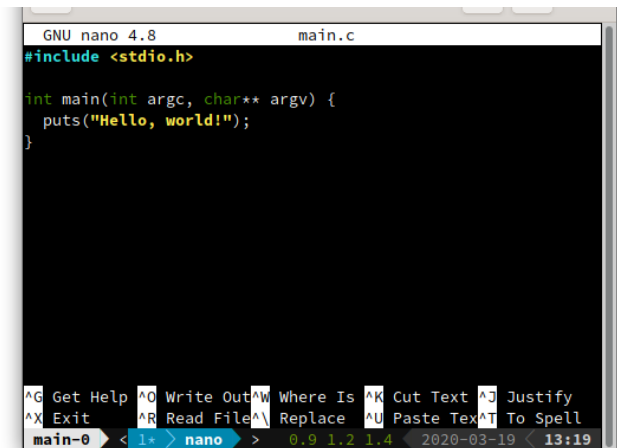


A terminal window showing the nano text editor editing a file named `main.c`. The prompt is `[thom@oceanus test]$ nano main.c`. The editor interface is dark with a light-colored status bar at the bottom. The status bar contains the following information from left to right: `main-0`, a left arrow, `1*`, a right arrow, `bash`, a right arrow, `0.6 1.1 1.5`, `2020-03-19`, a left arrow, and `13:16`.

Figure: Edit `main.c`

Editing a file

`nano file`: Edit file using nano



```
GNU nano 4.8          main.c
#include <stdio.h>

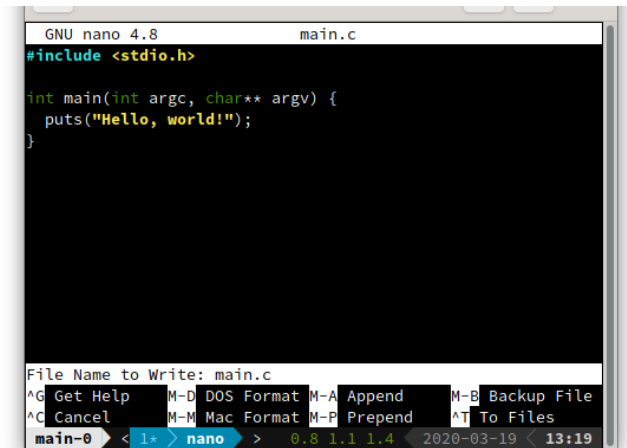
int main(int argc, char** argv) {
    puts("Hello, world!");
}
```

main-0 < 1* > nano > 0.9 1.2 1.4 < 2020-03-19 < 13:19

Figure: Edit main.c

Editing a file

`nano file`: Edit file using nano

A screenshot of the GNU nano 4.8 text editor. The window title is "main.c". The editor content shows a C program with the following code:

```
GNU nano 4.8          main.c
#include <stdio.h>

int main(int argc, char** argv) {
    puts("Hello, world!");
}
```

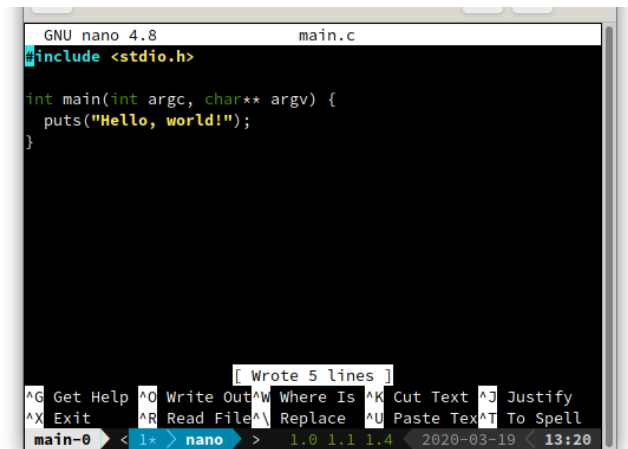
The bottom status bar displays "File Name to Write: main.c" and a menu of keyboard shortcuts: ^G Get Help, M-D DOS Format, M-A Append, M-B Backup File, ^C Cancel, M-M Mac Format, M-P Prepend, ^T To Files. The bottom-most bar shows "main-0", a cursor at the start, a blue bar with "1*" and "nano", and system information: "> 0.8 1.1 1.4 2020-03-19 < 13:19".

Figure: Edit main.c

I've pressed Ctrl+0 (^0)

Editing a file

`nano file`: Edit file using nano



```
GNU nano 4.8          main.c
#include <stdio.h>

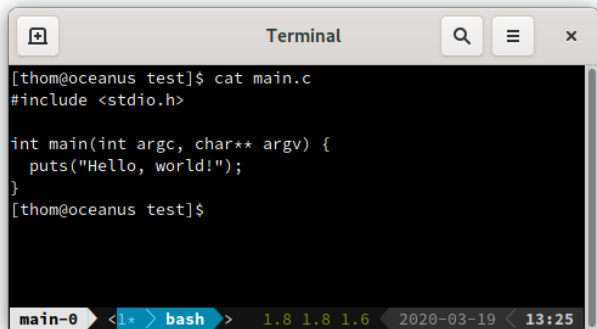
int main(int argc, char** argv) {
    puts("Hello, world!");
}

[ Wrote 5 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify
^X Exit      ^R Read File ^\ Replace  ^U Paste Tex ^T To Spell
main-0 > < 1* > nano > 1.0 1.1 1.4 < 2020-03-19 < 13:20
```

Figure: Edit main.c

Reading main.c: cat

`cat file`: Read file file

A terminal window titled "Terminal" with standard macOS window controls (close, zoom, search). The terminal shows the command `cat main.c` being executed, which outputs the contents of the file `main.c`. The output is a C program that includes `<stdio.h>` and has a `main` function that prints "Hello, world!". The terminal prompt `[thom@oceanus test]$` is visible before and after the command. At the bottom of the terminal, there is a status bar with "main-0", "bash", and system information including "1.8 1.8 1.6", "2020-03-19", and "13:25".

```
[thom@oceanus test]$ cat main.c
#include <stdio.h>

int main(int argc, char** argv) {
    puts("Hello, world!");
}
[thom@oceanus test]$
```

Figure: Read file

Searching in a file: `grep`

`grep pattern file`: Search for pattern in file
Supports regexes, case insensitive search, search in folders (recursively), see `man grep` and Google.



```
[thom@oceanus test]$ grep "Hello" main.c
puts("Hello, world!");
[thom@oceanus test]$
```

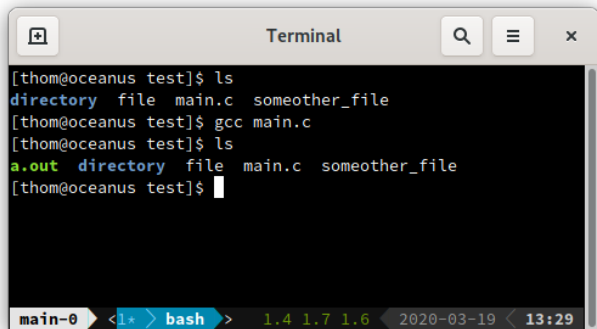
The image shows a terminal window titled "Terminal" with a search icon, a menu icon, and a close icon. The terminal content shows a user running the command `grep "Hello" main.c` in a directory named `test`. The output is `puts("Hello, world!");`. The terminal status bar at the bottom shows `main-0`, `<1*`, `bash`, `>`, `1.5 1.7 1.6`, `2020-03-19`, and `< 13:27`.

Figure: Search for Hello in `main.c`

Compiling a C program

`gcc file.c`: GNU C Compiler

Without options, compiles `file.c` to executable `a.out`



```
[thom@oceanus test]$ ls
directory file main.c someother_file
[thom@oceanus test]$ gcc main.c
[thom@oceanus test]$ ls
a.out directory file main.c someother_file
[thom@oceanus test]$
```

The image shows a terminal window titled "Terminal" with a search icon, a menu icon, and a close icon. The terminal content shows the execution of the following commands and their outputs:

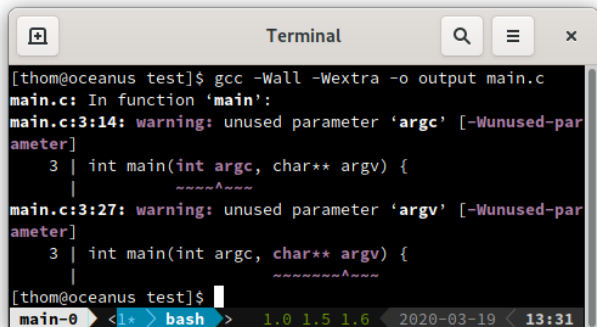
- `[thom@oceanus test]$ ls` outputs `directory file main.c someother_file`
- `[thom@oceanus test]$ gcc main.c` (no output)
- `[thom@oceanus test]$ ls` outputs `a.out directory file main.c someother_file`
- `[thom@oceanus test]$` (no output)

The terminal status bar at the bottom shows `main-0`, `<1*`, `bash`, `>`, `1.4 1.7 1.6`, `2020-03-19`, and `13:29`.

Figure: Compiling `main.c` to `a.out`

Compiling a C program, recommended settings

`gcc -Wall -Wextra -o output file.c`: Compile with warnings
Compiles `file.c` to executable `output`. Warnings are enabled



```
[thom@oceanus test]$ gcc -Wall -Wextra -o output main.c
main.c: In function 'main':
main.c:3:14: warning: unused parameter 'argc' [-Wunused-parameter]
     3 | int main(int argc, char** argv) {
       |             ~~~~~^~~~~
main.c:3:27: warning: unused parameter 'argv' [-Wunused-parameter]
     3 | int main(int argc, char** argv) {
       |                       ~~~~~^~~~~
[thom@oceanus test]$
```

Figure: Compiling `main.c` to `output`

Running a program in the current directory

`./file`: Run file

Necessary for programs that are not in a directory listed in the `$PATH` variable: for those you always need to specify *some* path.



```
Terminal
main.c:3:14: warning: unused parameter 'argc' [-Wunused-parameter]
   3 | int main(int argc, char** argv) {
     |             ^~~~~
main.c:3:27: warning: unused parameter 'argv' [-Wunused-parameter]
   3 | int main(int argc, char** argv) {
     |                       ^~~~~~
[thom@oceanus test]$ ./output
Hello, world!
[thom@oceanus test]$
```

The terminal window shows the compilation of a C program with two warnings about unused parameters. The program is then executed with `./output`, resulting in the output `Hello, world!`. The terminal prompt is `[thom@oceanus test]$`. The window title is `Terminal` and the status bar at the bottom shows `main-0`, `<1*`, `> bash`, `>`, `1.0 1.4 1.5`, `2020-03-19`, and `< 13:33`.

Figure: Run the output program

Table of Contents

Linux

Getting around on the command line

Common commands showcase

Exercise

Special I/O on Linux

System administration

Exiting vim

Exercise

1. Open a terminal
2. Create a folder `workshop` in your home directory
3. Create a file `hello.c` in this folder
4. Write your best `hello world` program
5. Compile it to a `hello` executable
6. Rename it to `helloworld`
7. Run it.
8. Make sure there's no other files in the folder
 - ▶ Delete them otherwise

I didn't show you all of the necessary commands to do all of this; use Google (“do something terminal linux” or “do something bash” usually works).

No cheating using the graphical interface!

Table of Contents

Linux

Getting around on the command line

Common commands showcase

Exercise

Special I/O on Linux

System administration

Exiting vim

Standard Output

When a program writes out (e.g. using `printf`, `puts`, `cout`, ...), it ends up in the terminal. This is called *standard output* or **stdout**.

Standard Output

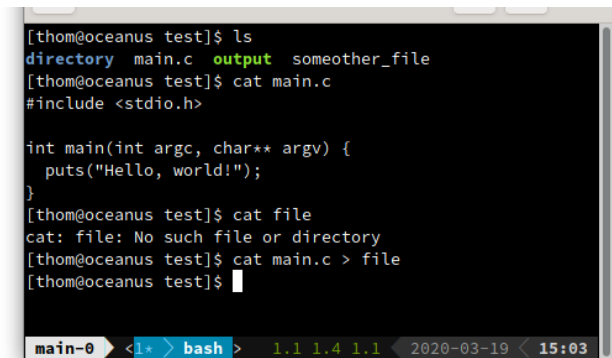
When a program reads in (e.g. using `gets`, `cin`, `readline`, ...), it reads from the terminal. This is called *standard input* or `stdin`.

Standard Error

There is another special form of output that a program can write, `stderr`. This also ends up in the terminal, but can be treated differently. Usually, this is used for errors or informational messages.

Redirecting output to a file

We can redirect standard output of a program to another file using `program > file`. This creates or, if it already existed, *truncates* file.

A terminal window with a black background and white text. The prompt is [thom@oceanus test]\$. The user enters 'ls', and the output is 'directory main.c output someother_file'. The user enters 'cat main.c', and the output is '#include <stdio.h>' followed by a blank line and 'int main(int argc, char** argv) {' followed by 'puts("Hello, world!");' followed by a closing brace. The user enters 'cat file', and the output is 'cat: file: No such file or directory'. The user enters 'cat main.c > file', and the output is empty. The user enters '\$', and the prompt is [thom@oceanus test]\$. The terminal has a status bar at the bottom with 'main-0', '< 1 * > bash >', '1.1 1.4 1.1', '2020-03-19', and '15:03'.

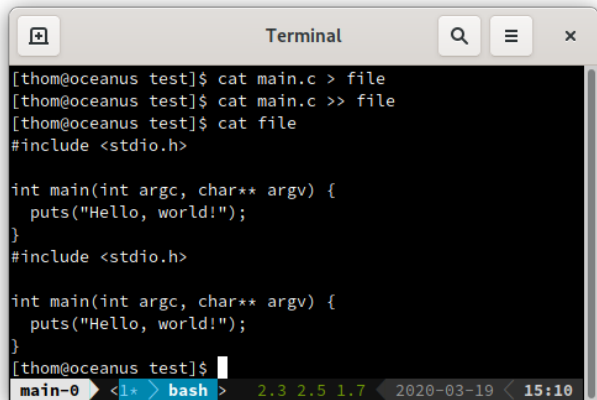
```
[thom@oceanus test]$ ls
directory main.c output someother_file
[thom@oceanus test]$ cat main.c
#include <stdio.h>

int main(int argc, char** argv) {
    puts("Hello, world!");
}
[thom@oceanus test]$ cat file
cat: file: No such file or directory
[thom@oceanus test]$ cat main.c > file
[thom@oceanus test]$
```

Figure: Redirecting the output of `cat main.c` to file

Appending output to a file

If we want to append the output to a file instead of truncating it, we can use `program >> file`.

A terminal window titled "Terminal" with standard macOS window controls (search, menu, close). The terminal shows a user named thom at a machine named oceanus in a directory named test. The user runs three commands: 1) `cat main.c > file`, which truncates the file 'file' to contain only the contents of 'main.c'. 2) `cat main.c >> file`, which appends the contents of 'main.c' to the end of 'file'. 3) `cat file`, which displays the contents of 'file'. The output shows that the file now contains two identical copies of the C code for a program that prints "Hello, world!". The terminal prompt is currently at the end of the third command.

```
[thom@oceanus test]$ cat main.c > file
[thom@oceanus test]$ cat main.c >> file
[thom@oceanus test]$ cat file
#include <stdio.h>

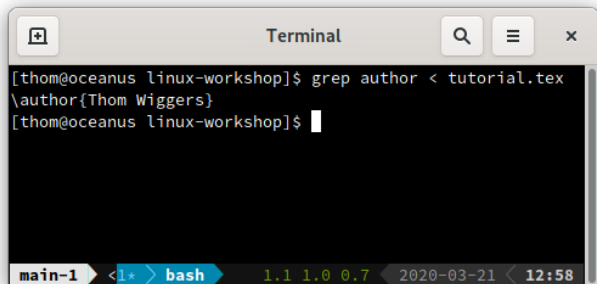
int main(int argc, char** argv) {
    puts("Hello, world!");
}
#include <stdio.h>

int main(int argc, char** argv) {
    puts("Hello, world!");
}
[thom@oceanus test]$
```

Figure: Appending the output of `cat main.c` to `file`

Providing a file as input to a program

If we want to take the contents of a file and provide it as input to a program, we can use `program < file`.



```
[thom@oceanus linux-workshop]$ grep author < tutorial.tex
\author{Thom Wiggers}
[thom@oceanus linux-workshop]$
```

The image shows a terminal window titled "Terminal" with a search icon, a menu icon, and a close icon. The terminal content shows a user at the prompt `[thom@oceanus linux-workshop]$` entering the command `grep author < tutorial.tex`. The output is `\author{Thom Wiggers}`. The prompt then returns to `[thom@oceanus linux-workshop]$`. At the bottom of the terminal, there is a status bar with "main-1", a navigation arrow, "bash", and system information: "1.1 1.0 0.7", "2020-03-21", and "12:58".

Figure: Providing `tutorial.tex` as input to `grep`

Redirecting output to a different program

It may also be useful to redirect output of one program to another program's `stdin`.

Imagine we want to check if thalia.nu mentions `borrel`. We use `curl` to get the webpage. Then we use `|` (pipe character) to redirect (pipe) `curl`'s `stdout` to `grep`'s `stdin`.

```
[thom@oceanus test]$ clear
[thom@oceanus test]$ curl https://thalia.nu | grep borrel
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload    Total   Spent    Left   Speed
100 23163  100 23163    0     0  91192    0  --:--:-- --:--:-- --:--:--  91553
[thom@oceanus test]$
```

Figure: Checking Thalia's front page for Borrels: none found

Redirecting output to a different program

It may also be useful to redirect output of one program to another program's `stdin`.

Imagine we want to check if thalia.nu mentions `borrel`. We use `curl` to get the webpage. Then we use `|` (pipe character) to redirect (**pipe**) `curl`'s **`stdout`** to `grep`'s **`stdin`**.

```
[thom@oceanus test]$ clear
[thom@oceanus test]$ curl https://thalia.nu | grep borrel
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 23163  100 23163    0     0  91192    0  --:--:-- --:--:-- --:--:--  91553
[thom@oceanus test]$
```

Figure: Checking Thalia's front page for Borrels: none found

The output about downloading the webpage is still printed, because it was printed to **`stderr`**!

Table of Contents

Linux

Getting around on the command line

Common commands showcase

Exercise

Special I/O on Linux

System administration

Exiting vim

Folder structure

- ▶ / root folder
 - ▶ /boot Boot loader stuff

Folder structure

- ▶ / root folder
 - ▶ /boot Boot loader stuff
 - ▶ /bin **Binaries**

Folder structure

- ▶ / root folder
 - ▶ /boot Boot loader stuff
 - ▶ /bin **Binaries**
 - ▶ /sbin Super user's /bin equivalent

Folder structure

- ▶ / root folder
 - ▶ /boot Boot loader stuff
 - ▶ /bin **Binaries**
 - ▶ /sbin Super user's /bin equivalent
 - ▶ /dev Special device files

Folder structure

- ▶ / root folder
 - ▶ /boot Boot loader stuff
 - ▶ /bin **Binaries**
 - ▶ /sbin Super user's /bin equivalent
 - ▶ /dev Special device files
 - ▶ /etc **System configuration**

Folder structure

- ▶ / root folder
 - ▶ /boot Boot loader stuff
 - ▶ /bin **Binaries**
 - ▶ /sbin Super user's /bin equivalent
 - ▶ /dev Special device files
 - ▶ /etc **System configuration**
 - ▶ /home **User's home folders**

Folder structure

- ▶ / root folder
 - ▶ /boot Boot loader stuff
 - ▶ /bin **Binaries**
 - ▶ /sbin Super user's /bin equivalent
 - ▶ /dev Special device files
 - ▶ /etc **System configuration**
 - ▶ /home **User's home folders**
 - ▶ /root **Root user's home folder**

Folder structure

- ▶ / root folder
 - ▶ /boot Boot loader stuff
 - ▶ /bin **Binaries**
 - ▶ /sbin Super user's /bin equivalent
 - ▶ /dev Special device files
 - ▶ /etc **System configuration**
 - ▶ /home **User's home folders**
 - ▶ /root **Root user's home folder**
 - ▶ /lib Shared libraries

Folder structure

- ▶ / root folder
 - ▶ /boot Boot loader stuff
 - ▶ /bin **Binaries**
 - ▶ /sbin Super user's /bin equivalent
 - ▶ /dev Special device files
 - ▶ /etc **System configuration**
 - ▶ /home **User's home folders**
 - ▶ /root **Root user's home folder**
 - ▶ /lib Shared libraries
 - ▶ /usr User-installed programs (/usr/bin), libraries (/usr/lib) and static data (/usr/share)

Folder structure

- ▶ / root folder
 - ▶ /boot Boot loader stuff
 - ▶ /bin **Binaries**
 - ▶ /sbin Super user's /bin equivalent
 - ▶ /dev Special device files
 - ▶ /etc **System configuration**
 - ▶ /home **User's home folders**
 - ▶ /root **Root user's home folder**
 - ▶ /lib Shared libraries
 - ▶ /usr User-installed programs (/usr/bin), libraries (/usr/lib) and static data (/usr/share)
 - ▶ /var Program-written data (logs, databases, caches)

Folder structure

- ▶ / root folder
 - ▶ /boot Boot loader stuff
 - ▶ /bin **Binaries**
 - ▶ /sbin Super user's /bin equivalent
 - ▶ /dev Special device files
 - ▶ /etc **System configuration**
 - ▶ /home **User's home folders**
 - ▶ /root **Root user's home folder**
 - ▶ /lib Shared libraries
 - ▶ /usr User-installed programs (/usr/bin), libraries (/usr/lib) and static data (/usr/share)
 - ▶ /var Program-written data (logs, databases, caches)
 - ▶ /tmp **Temporary files**

Owners and permissions

- ▶ Files and folders have an owner and a group

Owners and permissions

- ▶ Files and folders have an owner and a group
- ▶ Permissions are usually set on three levels

Owners and permissions

- ▶ Files and folders have an owner and a group
- ▶ Permissions are usually set on three levels
 - ▶ What can the owning **user** do?

Owners and permissions

- ▶ Files and folders have an owner and a group
- ▶ Permissions are usually set on three levels
 - ▶ What can the owning **u**ser do?
 - ▶ What can the **g**roup members do?

Owners and permissions

- ▶ Files and folders have an owner and a group
- ▶ Permissions are usually set on three levels
 - ▶ What can the owning **u**ser do?
 - ▶ What can the **g**roup members do?
 - ▶ What can the **o**thers do?

Owners and permissions

- ▶ Files and folders have an owner and a group
- ▶ Permissions are usually set on three levels
 - ▶ What can the owning **u**ser do?
 - ▶ What can the **g**roup members do?
 - ▶ What can the **o**thers do?
- ▶ The three permissions are:

Owners and permissions

- ▶ Files and folders have an owner and a group
- ▶ Permissions are usually set on three levels
 - ▶ What can the owning **u**ser do?
 - ▶ What can the **g**roup members do?
 - ▶ What can the **o**thers do?
- ▶ The three permissions are:
 - ▶ **r**ead

Owners and permissions

- ▶ Files and folders have an owner and a group
- ▶ Permissions are usually set on three levels
 - ▶ What can the owning **u**ser do?
 - ▶ What can the **g**roup members do?
 - ▶ What can the **o**thers do?
- ▶ The three permissions are:
 - ▶ **r**ead
 - ▶ **w**rite

Owners and permissions

- ▶ Files and folders have an owner and a group
- ▶ Permissions are usually set on three levels
 - ▶ What can the owning **u**ser do?
 - ▶ What can the **g**roup members do?
 - ▶ What can the **o**thers do?
- ▶ The three permissions are:
 - ▶ **r**ead
 - ▶ **w**rite
 - ▶ **e**xecute

Owners and permissions

- ▶ Files and folders have an owner and a group
- ▶ Permissions are usually set on three levels
 - ▶ What can the owning **u**ser do?
 - ▶ What can the **g**roup members do?
 - ▶ What can the **o**thers do?
- ▶ The three permissions are:
 - ▶ **r**ead
 - ▶ **w**rite
 - ▶ **x**ecute
- ▶ `ls -l` shows this information

Owners and permissions

- ▶ Files and folders have an owner and a group
- ▶ Permissions are usually set on three levels
 - ▶ What can the owning **u**ser do?
 - ▶ What can the **g**roup members do?
 - ▶ What can the **o**thers do?
- ▶ The three permissions are:
 - ▶ **r**ead
 - ▶ **w**rite
 - ▶ **x**ecute
- ▶ `ls -l` shows this information

Owners and permissions

- ▶ Files and folders have an owner and a group
- ▶ Permissions are usually set on three levels
 - ▶ What can the owning **u**ser do?
 - ▶ What can the **g**roup members do?
 - ▶ What can the **o**thers do?
- ▶ The three permissions are:
 - ▶ **r**ead
 - ▶ **w**rite
 - ▶ **x**ecute
- ▶ `ls -l` shows this information

```
[thom@oceanus test]$ ls -l
total 24
drwxr-xr-x 3 thom thom    60 19 mrt 13:06 directory
-rw-r--r-- 1 thom thom     0 19 mrt 12:49 file
-rw-r--r-- 1 thom thom    81 19 mrt 13:25 main.c
-rwxr-xr-x 1 thom thom 16592 19 mrt 13:31 output
-rw-r--r-- 1 thom thom     0 19 mrt 12:49 someother_file
```

Figure: `ls -l` shows ownership and permission information

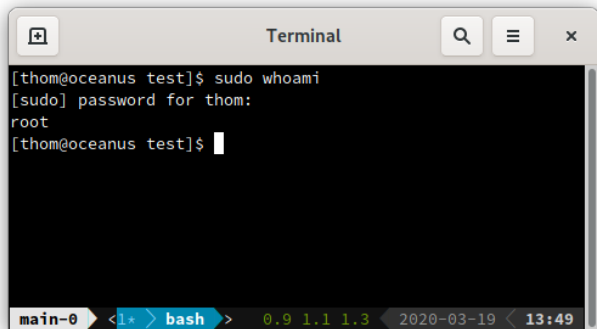
Changing owners and permissions

- ▶ Change owner of a file using `chown`
 - ▶ `chown otheruser file`
- ▶ Change group of a file using `chgrp`
 - ▶ `chgrp othergroup file`
- ▶ Change permissions (*mode*) of a file `chmod`
 - ▶ `chmod u+x file`
 - ▶ `chmod g+w file`
 - ▶ `chmod o-rwx file`

Putting on your robe and wizard hat: `sudo`

`sudo`: Super User Do

Run the specified command as `root`. Your user needs to be on a special list to do so (`/etc/sudoers`, edit using `visudo`). Get a root shell using `sudo -i`.



```
[thom@oceanus test]$ sudo whoami
[sudo] password for thom:
root
[thom@oceanus test]$
```

The image shows a terminal window titled "Terminal" with a search icon, a menu icon, and a close icon. The terminal content shows a user named "thom" at a host named "oceanus" in a directory "test" running the command "sudo whoami". The terminal prompts for the password "thom" and then outputs "root". The prompt then returns to "[thom@oceanus test]\$". At the bottom of the terminal, there is a status bar showing "main-0", a navigation bar with "1*" and "bash", and system information including "0.9 1.1 1.3", "2020-03-19", and "13:49".

Figure: Running a command as root

Installing applications: apt

- ▶ `apt update`: Update the cached index of packages
- ▶ `apt search`: Search for applications (searches in cached index)
- ▶ `apt install`: Install an application
- ▶ `apt upgrade`: Upgrade the installed applications

For non-Debian based distributions (so unlike Ubuntu), the package managers usually have equivalent commands.

Table of Contents

Linux

Getting around on the command line

Common commands showcase

Exercise

Special I/O on Linux

System administration

Exiting vim

Vim

Someone told you that you should really try Vim, first hit is free.

Vim

Someone told you that you should really try Vim, first hit is free.



Figure: Nancy Reagan

Seriously though, what's up with Vim

Vim is a super-powerful editor, but it has a very weird model behind it that makes it hard to use.

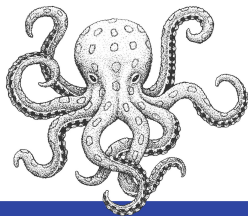
Stick to nano or gedit unless you want to invest a lot of time.

If you do want to learn it, consider one or more of the following:

- ▶ `vimtutor` (usually comes with your Vim installation)
- ▶ <https://vim-adventures.com/> fun game, not free after first few levels
- ▶ <https://openvim.com> Another online tutorial
- ▶ <https://github.com/jmoon018/PacVim> teaches you Vim's movements
- ▶ <https://missing.csail.mit.edu/2020/editors/>
- ▶ <https://vimeo.com/user1690209> or whatever video tutorial you can find

Exiting Vim

Just memorize these fourteen contextually dependant instructions



Exiting Vim

Eventually

ORLY?

@ThePracticalDev

Figure: Exiting Vim

<https://github.com/hakluke/how-to-exit-vim>

Finding this presentation

This presentation can be found at <https://thomwiggers.nl/teaching/hacking-in-c-2020/shell-tutorial/>. I will also link it on the Hacking in C Brightspace page.

An alternative tutorial that goes a bit further is [this one by MIT's "Missing Semester"](#).

See you at Hacking in C!