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

Table of Contents

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

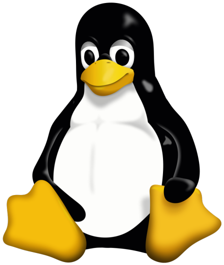


Figure: Tux

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 Jeffery 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

The shell

- `/bin/bash`: **B**ourne **a**gain **sh**ell
  - 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!

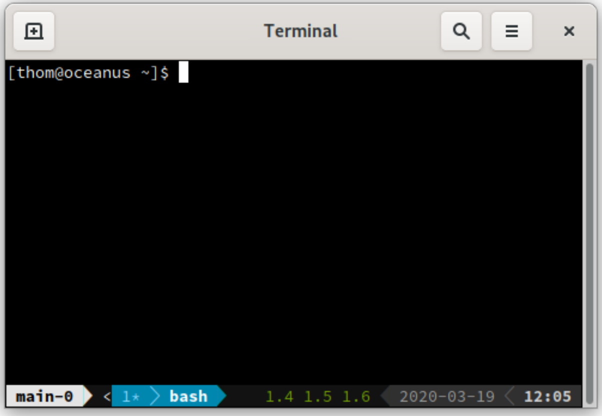
Figure: Different kind of shell

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

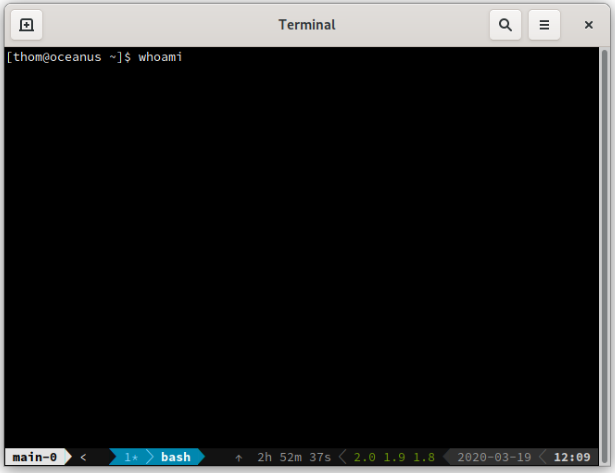


```
[thom@oceanus ~]$ 
```

```
main-0 > < 1* > bash        1.4 1.5 1.6 < 2020-03-19 < 12:05
```

Figure: The shell prompt

Running programs

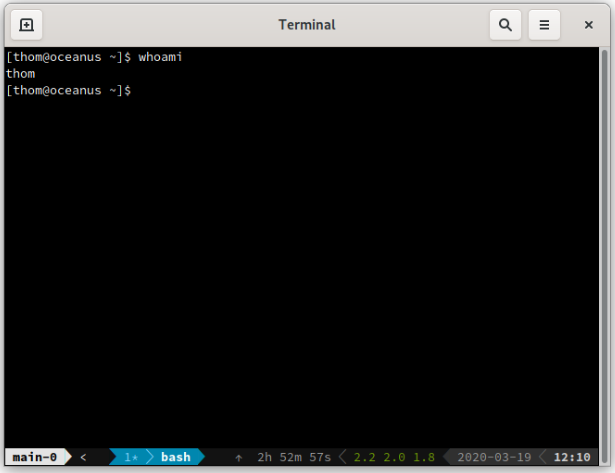


```
[thom@oceanus ~]$ whoami
```

```
main-0 < 1* bash ↑ 2h 52m 37s 2.0 1.9 1.8 2020-03-19 12:09
```

Figure: Running `whoami`

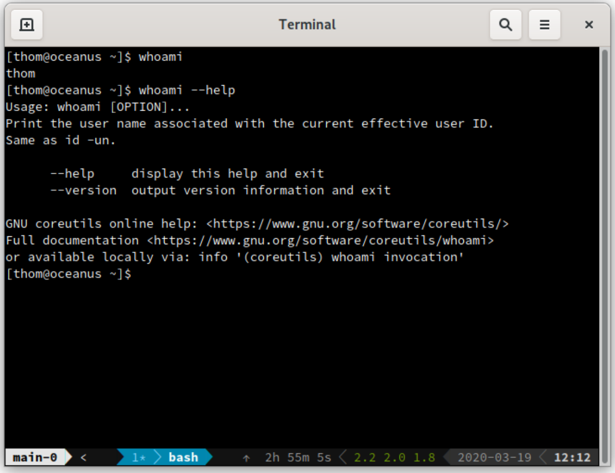
Running programs



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

Figure: Running whoami

Running programs

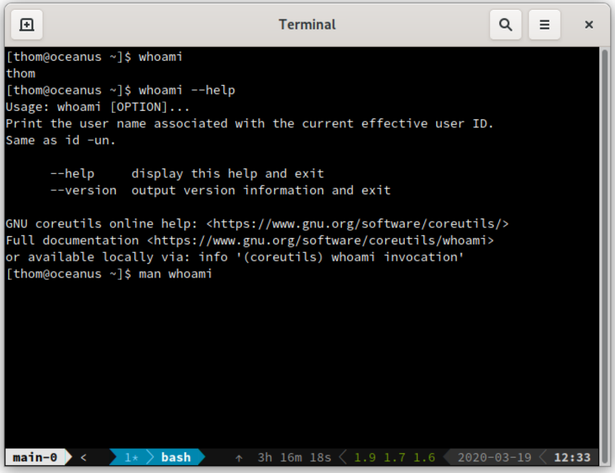


```
[thom@oceanus ~]$ whoami
thom
[thom@oceanus ~]$ whoami --help
Usage: whoami [OPTION]...
Print the user name associated with the current effective user ID.
Same as id -un.

      --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 ~]$
```

```
main-0  <   1*  bash    ↑  2h 55m 5s  2.2 2.0 1.8  2020-03-19  12:12
```

Figure: Running `whoami`

Running programs

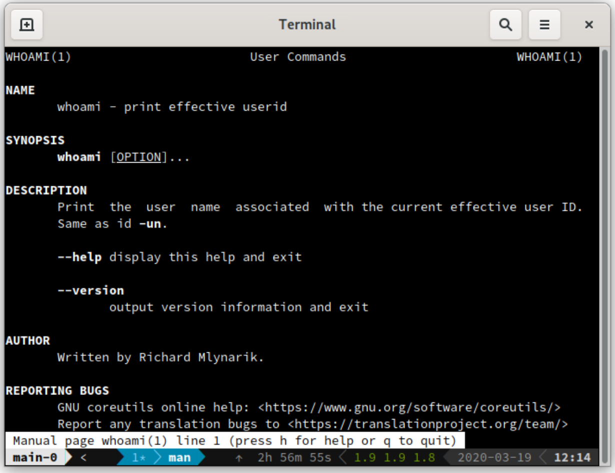


```
[thom@oceanus ~]$ whoami
thom
[thom@oceanus ~]$ whoami --help
Usage: whoami [OPTION]...
Print the user name associated with the current effective user ID.
Same as id -un.

      --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
```

```
main-0  <   1*  bash      ↑ 3h 16m 18s  1.9 1.7 1.6  2020-03-19  12:33
```

Figure: Running `whoami`

Running programs



```
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`

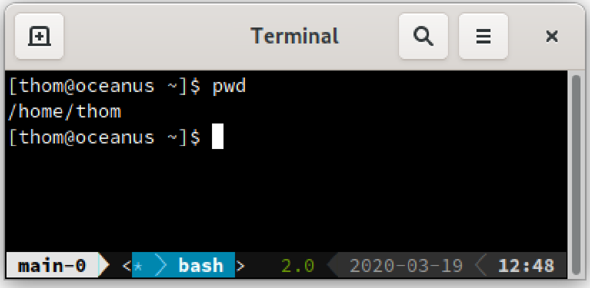
The manual

- 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

Where am I: pwd

pwd: Print Working Directory

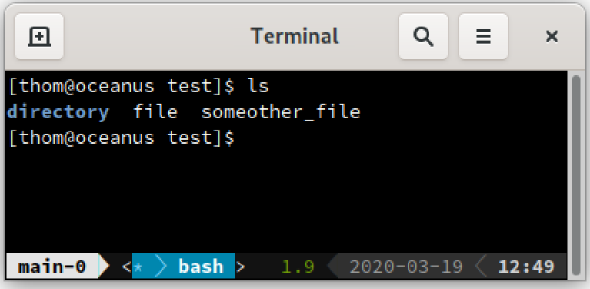


```
[thom@oceanus ~]$ pwd
/home/thom
[thom@oceanus ~]$ 
```

Figure: pwd

What is here: `ls`

`ls`: list files in folder

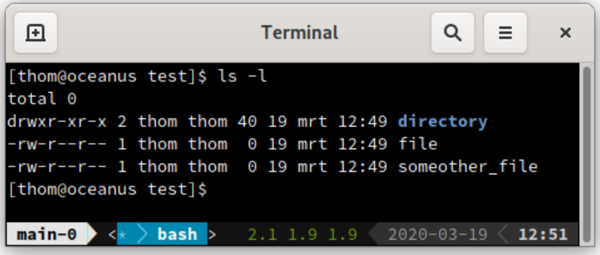


```
[thom@oceanus test]$ ls
directory  file  someother_file
[thom@oceanus test]$
```

Figure: `ls`

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

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

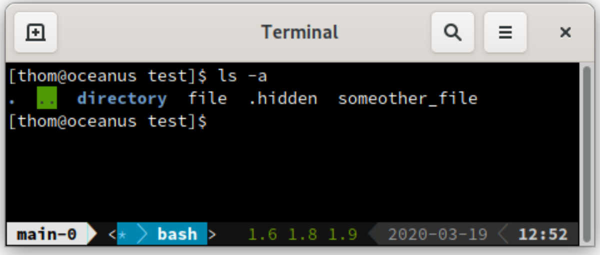


```
[thom@oceanus test]$ ls -l
total 0
drwxr-xr-x 2 thom thom 40 19 mrt 12:49 directory
-rw-r--r-- 1 thom thom  0 19 mrt 12:49 file
-rw-r--r-- 1 thom thom  0 19 mrt 12:49 someother_file
[thom@oceanus test]$
 main-0  <*  bash >   2.1 1.9 1.9  2020-03-19  12:51
```

Figure: `ls -l`

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

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



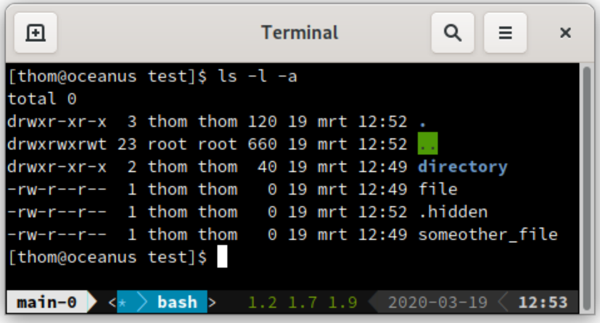
```
[thom@oceanus test]$ ls -a
.  ..  directory  file  .hidden  someother_file
[thom@oceanus test]$
```

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



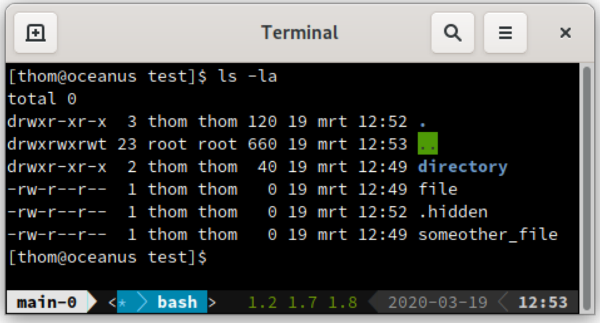
```
[thom@oceanus test]$ ls -l -a
total 0
drwxr-xr-x  3 thom thom 120 19 mrt 12:52 .
drwxrwxrwt 23 root root 660 19 mrt 12:52 ..
drwxr-xr-x  2 thom thom  40 19 mrt 12:49 directory
-rw-r--r--  1 thom thom   0 19 mrt 12:49 file
-rw-r--r--  1 thom thom   0 19 mrt 12:52 .hidden
-rw-r--r--  1 thom thom   0 19 mrt 12:49 someother_file
[thom@oceanus test]$ 
main-0 > <* > bash >  1.2 1.7 1.9 < 2020-03-19 < 12:53
```

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

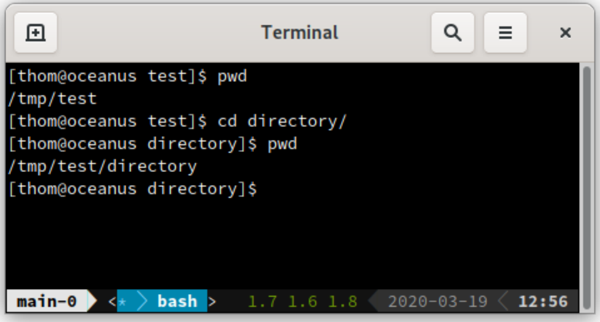


```
[thom@oceanus test]$ ls -la
total 0
drwxr-xr-x  3 thom thom 120 19 mrt 12:52 .
drwxrwxrwt 23 root root 660 19 mrt 12:53 ..
drwxr-xr-x  2 thom thom  40 19 mrt 12:49 directory
-rw-r--r--  1 thom thom   0 19 mrt 12:49 file
-rw-r--r--  1 thom thom   0 19 mrt 12:52 .hidden
-rw-r--r--  1 thom thom   0 19 mrt 12:49 someother_file
[thom@oceanus test]$
 main-0 > <* > bash >   1.2 1.7 1.8 < 2020-03-19 < 12:53
```

Figure: `ls -la`

Moving around: cd

cd dir: change into directory dir

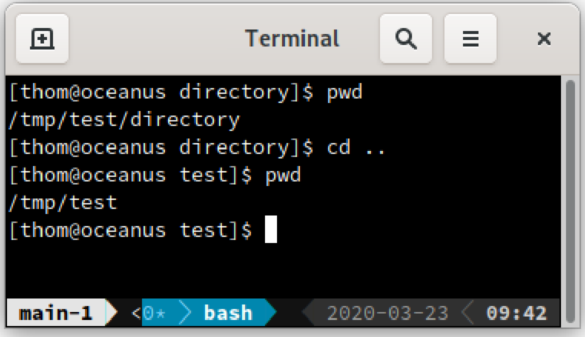


```
[thom@oceanus test]$ pwd
/tmp/test
[thom@oceanus test]$ cd directory/
[thom@oceanus directory]$ pwd
/tmp/test/directory
[thom@oceanus directory]$
```

Figure: Changing into directory

Moving to the parent directory: `cd ..`

`cd ..`: move to parent directory.

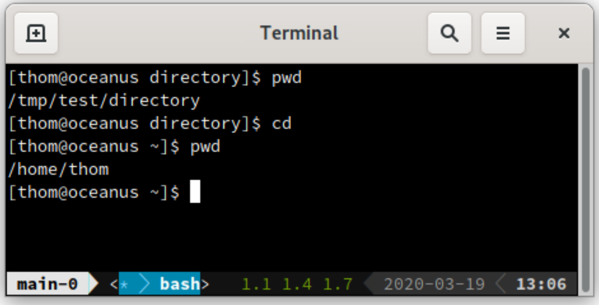


```
[thom@oceanus directory]$ pwd
/tmp/test/directory
[thom@oceanus directory]$ cd ..
[thom@oceanus test]$ pwd
/tmp/test
[thom@oceanus test]$
```

Figure: Changing into parent directory

Moving to your home dir: cd

cd: change directory to home

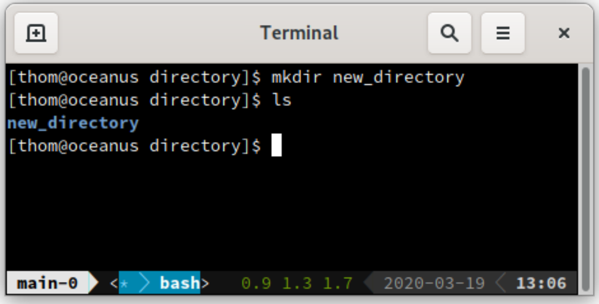


```
[thom@oceanus directory]$ pwd
/tmp/test/directory
[thom@oceanus directory]$ cd
[thom@oceanus ~]$ pwd
/home/thom
[thom@oceanus ~]$
main-0  <* > bash>   1.1 1.4 1.7  2020-03-19  13:06
```

Figure: Changing into /home/thom

Creating a new folder: mkdir

mkdir dir: Create folder dir

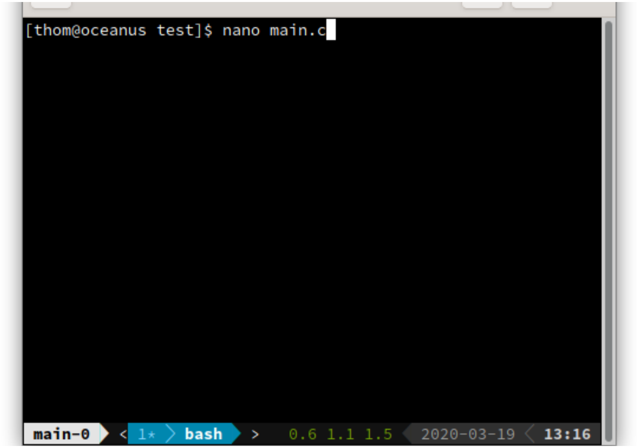


```
[thom@oceanus directory]$ mkdir new_directory
[thom@oceanus directory]$ ls
new_directory
[thom@oceanus directory]$ 
 main-0  <* bash>   0.9 1.3 1.7  2020-03-19  13:06
```

Figure: Create new_directory

Editing a file

`nano file`: Edit `file` using `nano`

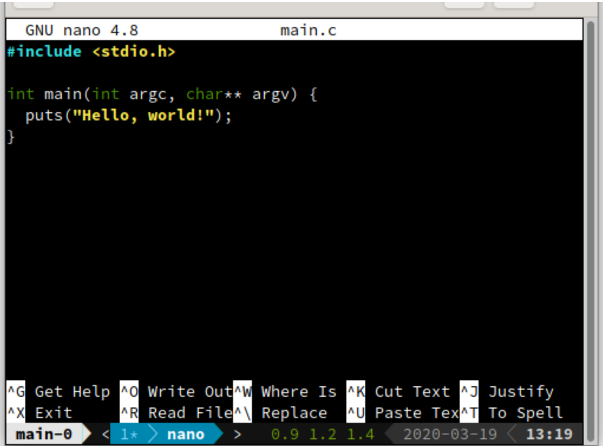


```
[thom@oceanus test]$ nano main.c
```

Figure: Edit `main.c`

Editing a file

`nano file`: Edit `file` using `nano`

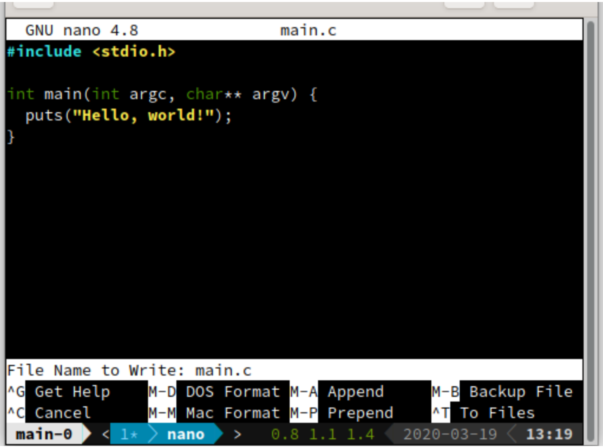


```
  GNU nano 4.8                        main.c
#include <stdio.h>

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













^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  >   0.9 1.2 1.4  2020-03-19  13:19
```

Figure: Edit `main.c`

Editing a file

`nano file`: Edit `file` using `nano`



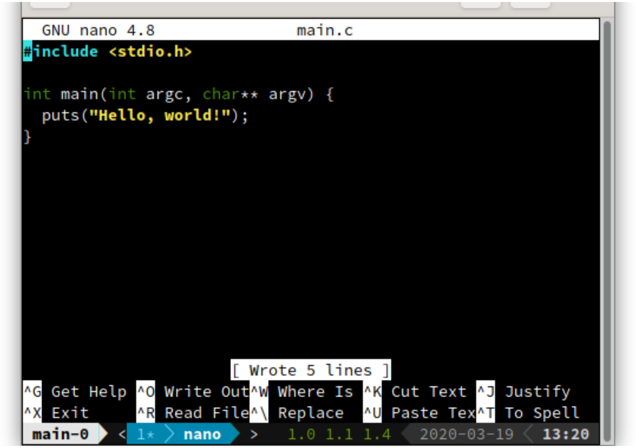
```
 GNU nano 4.8                    main.c
#include <stdio.h>

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











File Name to Write: main.c
^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
 main-0  < 1* nano >   0.8 1.1 1.4  2020-03-19  13:19
```

Figure: Edit `main.c`

I've pressed `Ctrl+O (^O)`

Editing a file

`nano file`: Edit `file` using `nano`

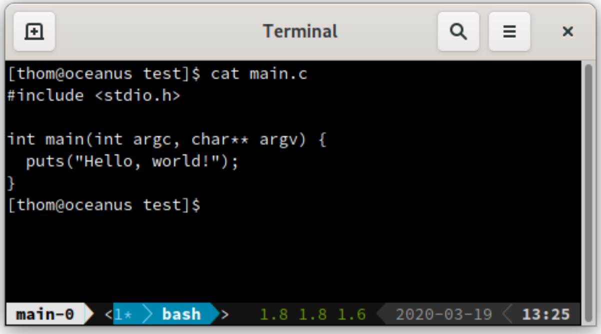


```
  GNU nano 4.8                     main.c
#include <stdio.h>

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`



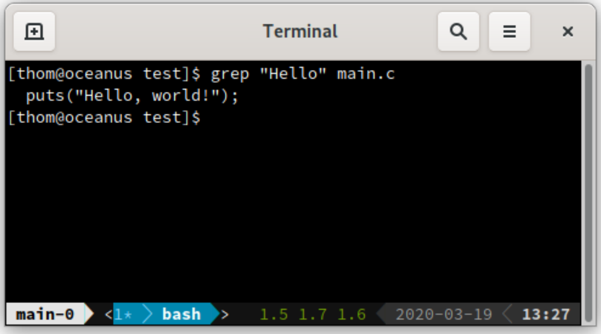
```
[thom@oceanus test]$ cat main.c
#include <stdio.h>

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

```
main-0  <1*  bash >    1.8 1.8 1.6  2020-03-19  13:25
```

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.



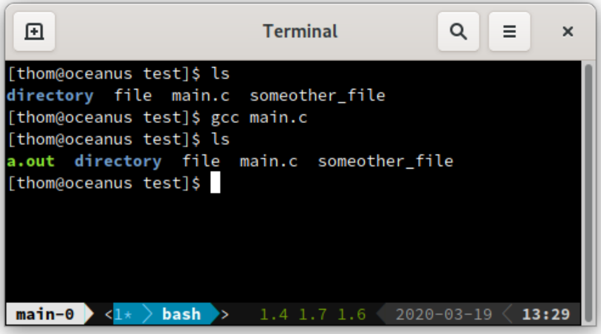
```
[thom@oceanus test]$ grep "Hello" main.c
  puts("Hello, world!");
[thom@oceanus test]$
```

```
main-0 <1* bash >  1.5 1.7 1.6  2020-03-19  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`

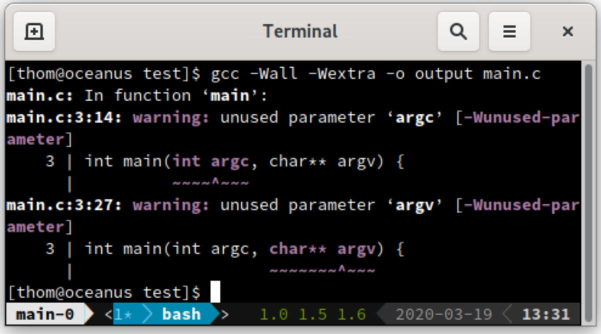


```
[thom@oceanus test]$ ls
directory  file  main.c  someother_file
[thom@oceanus test]$ gcc main.c
[thom@oceanus test]$ ls
a.out  directory  file  main.c  someother_file
[thom@oceanus test]$ 
```

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



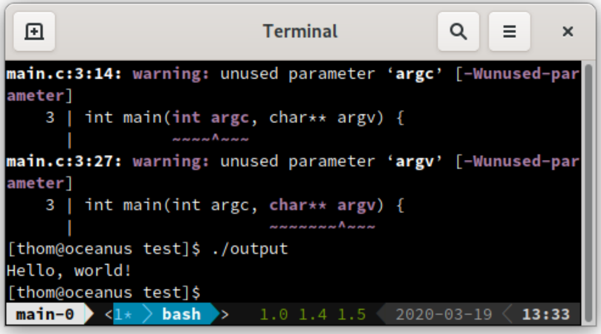
```
[thom@oceanus test]$ gcc -Wall -Wextra -o output main.c
main.c: In function ‘main’:
main.c:3:14: warning: unused parameter ‘argc’ [-Wunused-par
ameter]
    3 | int main(int argc, char** argv) {
      |          ~~~~~^~~~
main.c:3:27: warning: unused parameter ‘argv’ [-Wunused-par
ameter]
    3 | int main(int argc, char** argv) {
      |                    ~~~~~~~^~~~
[thom@oceanus test]$ 
 main-0 > <1* > bash >>   1.0 1.5 1.6 < 2020-03-19 < 13:31
```

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.



```
main.c:3:14: warning: unused parameter ‘argc’ [-Wunused-par
ameter]
    3 | int main(int argc, char** argv) {
      |          ~~~~^~~~
main.c:3:27: warning: unused parameter ‘argv’ [-Wunused-par
ameter]
    3 | int main(int argc, char** argv) {
      |                    ~~~~~~~^~~~
[thom@oceanus test]$ ./output
Hello, world!
[thom@oceanus test]$
 main-0  <1*  bash >   1.0 1.4 1.5  2020-03-19  13:33
```

Figure: Run the `output` program

Table of Contents

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

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

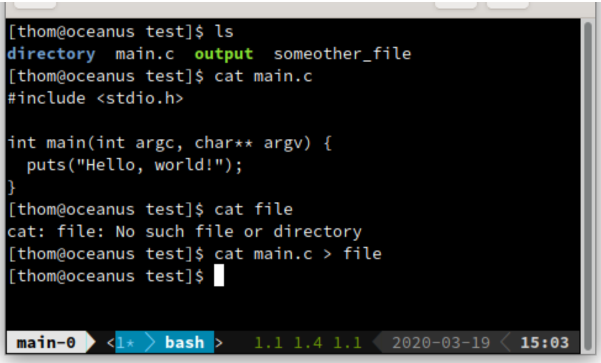
When a program reads in (e.g. using `gets`, `cin`, `readline`, . . .), it reads from the terminal. This is called *standard input* or `stdin`.

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`.

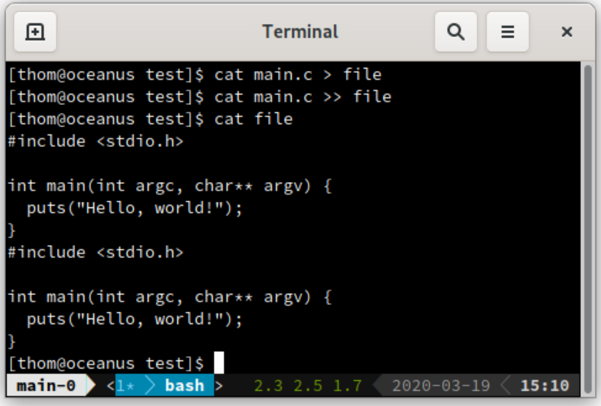


```
[thom@oceanus test]$ ls
directory  main.c  output  someother_file
[thom@oceanus test]$ cat main.c
#include <stdio.h>

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`.

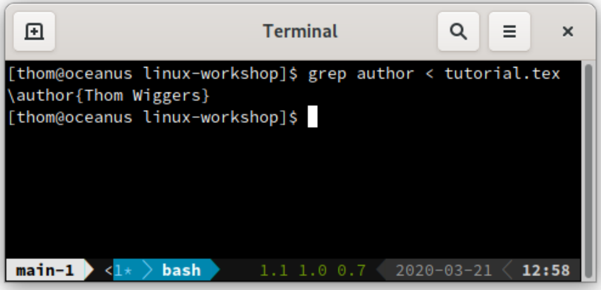


```
[thom@oceanus test]$ cat main.c > file
[thom@oceanus test]$ cat main.c >> file
[thom@oceanus test]$ cat file
#include <stdio.h>

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

int main(int argc, char** argv) {
  puts("Hello, world!");
}
[thom@oceanus test]$ 
 main-0  <1*  bash >   2.3 2.5 1.7   2020-03-19 < 15:10
```

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`.

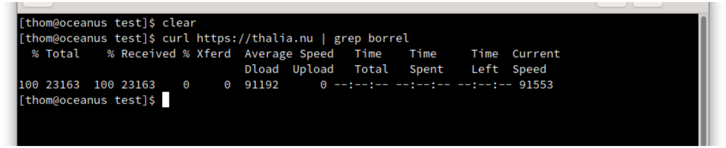


```
[thom@oceanus linux-workshop]$ grep author < tutorial.tex
\author{Thom Wiggers}
[thom@oceanus linux-workshop]$
```

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`.

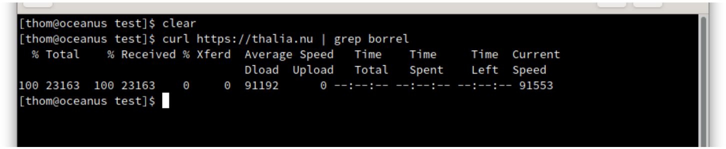


```
[thom@oceanus test]$ clear
[thom@oceanus test]$ curl https://thalia.nu | grep borrel
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 23163  100 23163    0     0  91192      0 --:--:--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

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 **u**ser 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**x**ecute

Owners and permissions

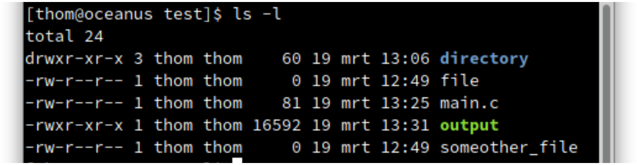
- Files and folders have an owner and a group
- Permissions are usually set on three levels
  - What can the owning **u**ser do?
  - What can the **g**roup members do?
  - What can the **o**thers do?
- The three permissions are:
  - **r**ead
  - **w**rite
  - e**x**ecute
- `ls -l` shows this information

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
  - e**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
  - e**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

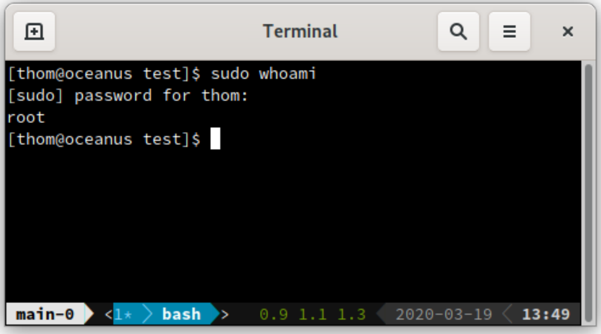
Changing owners and permissions

- 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]$ 
```

Figure: Runing 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

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
O RLY?
@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!