Contents

1 Welcome

Welcome to the Hacking Session! This workshop is designed as an exercise to go with the Toolkit course. If you did not attend said course, we suggest you have a look at its slides, which can be found on our website. However, you can of course also solve them independently. Please make sure to read this page carefully before starting.

1.1 A Word About Organisation

Exclusion of liability

By taking part in this exercise, you acknowledge that you alone are responsible for your computer. TheAlternative and its parent organisation, the Student Sustainability Commission, will not be held liable for any damages or loss of data.

Ask

If anything is unclear in the exercises feel free to raise your hand and one of our staff members will come and help you.

Difficulty

The exercises are designed to be solved through self-initiative. For most exercises there are no step-by-step guides, but we will give you hints that point you in the right direction. In addition, the usage of manual pages and search engines is actively encouraged. The exercises are generally not meant to be completely solved within the 2 hours given to you in the session, as we tried to present a wide range of different topics with varying degrees of difficulty. Feel free to skip some exercises if they're not challenging or interesting enough to you, or if you get stuck at something.

1.2 Getting What You Need

In addition to this exercise sheet, you will need to download some files for some of the exercises. Those files can be found on our Github repository. Do not worry if you do not know yet what `git` is and how to use it. Simply follow the steps described below:

- Open the terminal on your computer
- Navigate to the directory you wish to download the files to
- Type `git clone https://github.com/TheAlternativeZurich/HackingSession.git`

Now you have all the necessary files and directories and you are ready to start.

Have fun!

2 Basics

This section aims to introduce you to, and train the usage of basic concepts and commands.

2.1 Explanation of Basic Concepts

Here you will find a short explanation of the main concepts to get you started. If you're tired of reading or already know most of this from the Toolkit course, then skip down to the `Exercises` section to test your skills at some practical tasks.

Terminal and Commands

The Linux terminal, also called console, is the standard command line interface we use to enter commands. It is found on every Linux distribution, and is the most versatile tool at your disposal. It lets you use a so-called `shell`, which is in simple terms a program that takes commands written by the user and executes them. Don't worry much about the exact definitions of a shell and a terminal. Being able to differentiate the two is irrelevant for an every-day user.

To run a command, type its name in the console and then press `Enter`. Should you ever need to cancel a running command, you can use the keyboard combination `Ctrl+C` to interrupt it. Note that the shell (by default) is case-sensitive, so capitalization matters.

man

`man` is a command you will need throughout all of the following exercises. It stands for "manual" and shows you what almost any given command can do and which options are available for it. Almost all commands provide such a "manpage", which is typically written by the developers themselves. They are hence reliable, and are the single most important tool to get comfortable with using your shell.

Some man pages are absolutely huge and may make you feel quite overwhelmed when you first open them. This is completely normal! The information you need for actually understanding how to run a program/command will in 90% of the cases be described in the first few lines of the page. The bulk of the manual usually describes special options (flags) that change how the command behaves when executed. Don't worry if you can't remember the details of a command or what its various options do. The most important thing to take away from a manual is a rough idea of *what* a command does: With that, you will know in which situations you can use it. The details can still be looked up in the manual and you will learn them by heart as time goes by.

Try it yourself! The `man` command itself has its own manpage. Type `man man` to access it, use the arrow keys to scroll, and press Q to exit.

An alternative to using the man pages is to try and see if a command has a help option. This is usually displayed by appending the option `--help` after the name of the command (don't forget to put spaces between the name of the command and the option), which will often display much shorter and concise instructions on how to use a command. For example, the `man` command also has a `--help` option. Simply type `man --help`, and the command will display information directly in your terminal.

Flags

Flags are options you can add to a command to change the way it behaves. The option `-h` or `--help` described above is one of them, and is a good example as it causes the command to display information about itself instead of performing its main function. There are many more flags like this, and their name and purpose usually differs between all programs and commands. Single character options are always preceded by `-`, while those with multiple letters are preceded by `--`. It is also possible to use multiple different flags together at once, and you can also string them together for quick execution. For example, for the command `ls` described below, you can type `ls -lah` instead of `ls -l -a -h`. (to see what these flags do, consult the man page by typing `man ls`!)

Wildcards

Wildcards are placeholder symbols that are replaced when a command is read by the shell.

There are three different types of wild cards:

- The star wildcard `*` acts as a placeholder for any number of consecutive symbols. For example, writing `*.jpg` addresses all files and folders that end with `.jpg`. Writing `D*s` addresses all files and folders that start with a `D` and end with a `s`, and so on.
- The question mark wildcard `?` acts as a placeholder for a single symbol only. For example: `???.png` will address all files that end with `.png` that have 3 letters.
- The bracket wildcard `[...]` can be used as a placeholder for the symbols we specify inside. [1-9] for example acts as a placeholder for the numbers 1 to 9, and [ABC] acts as a placeholder for those 3 letters only.

Furthermore, there is also a concept called "Regular Expressions" which fulfills a similar purpose. We will not elaborate on this here however.

sudo

Similar to Windows or MacOS, in Linux we differ between user accounts that have different access permissions. Users usually do not have permission to simply install new packages or alter files outside of their home directory. The exception to this is root, who has permission to alter whatever he wants, install whatever he wants, and destroy whatever he wants.

In order for a user to run single commands and programs as root, we make use the command `sudo`. Example: `sudo echo "Hello World"`

If the user who executes the command is in the sudoers group (grants permission to use `sudo`), the shell then asks the user for his password. Once it is entered, the command will run just as it normally

would – only that now, it is being ran by root instead of the user account. If you don't want to enter your password every time you need to use sudo, you can use the command `sudo su` to become root inside the current shell. This will allow you to run all commands with no confirmation needed (We strongly advise against doing this for now however!). Type `exit` if you want to return to the user account.

Note that most commands that require you to use `sudo` usually have a good reason to do so, as they may have the potential to cause serious damage to your system. Upon first entering `sudo`, you will be met with the quote `With great power comes great responsibility.`. Take this to heart, as the majority of commands on Linux will not ask twice, even when you're about to do something foolish.

2.2 Exercises

Here you will get to try things yourself!

We don't give you any step-by-step instructions however. You will have to find things out yourself! (or ask our helpers should you get stuck)

Exercise 0: man

If you haven't done so before, you should definitely make yourself familiar with the `man` command, as you will need it for the following tasks. Type `man man` in the shell and hit enter. You don't have to read the whole thing, but skimming over it is definitely a good idea. Also, while the manual of `man` is open, you can hit `h` to make a window with keyboard shortcuts pop up. This is a good opportunity to take a look at them, because it is vital that you know how to search a manpage and how to move around within it quickly!

Exercise 1: ls

`ls` stands for "list".

a) List all files (including hidden files) in your current directory.
b) List all files in your `/bin` directory.
c) List all files in your `/bin` directory whose names consist of only two letters.
d) Find the size of your bash executable (`/usr/bin/bash` or `/bin/bash`).

Exercise 2: mv

`mv` stands for "move".

a) Make a file `hello` using `touch hello`. Now rename this file to `world`.
b) Move this file into a different directory. (e.g. your `Downloads` folder).

Exercise 3: cd

`cd` stands for "change directory".

a) From your home directory, change to `./Documents`. Then change back to the `home` directory.

b) Change into the `/root` directory
c) Change back to your home directory without writing out the full path.

Exercise 4: cat

`cat` stands for "con*cat*enate files and print to the standard output".

a) Look at the output of whatever file you desire.
b) Move the contents of one file into another, overwriting it. (Hint: Look up the redirection operator)

Exercise 5: mkdir and rmdir

`mkdir` stands for "make directory", `rmdir` for "remove directory"

a) Make a directory called `hello` in your home directory, and then remove it.
b) Can you remove your home directory with rmdir?

Exercise 6: rm

`rm` stands for "remove". It directly removes files, it doesn't put them in the trash. Any data that you remove with `rm` is gone forever.

a) Create a file called `world`, then delete it.
b) Create a directory called `foo`. Change to this directory, then create 3 files with names of your choice inside this directory. Remove them all at once.

Exercise 7: cp

`cp` stands for "copy".

a) Make a `Backup` directory and copy your `.bashrc` (bash configuration file) there. Check if it worked with `ls`.
b) Copy a directory into `Backup/` (e.g. your `Downloads` directory). What is the difference to copying a file?
c) Remove the `Backup/` directory you just created.

Exercise 8: clear

`clear` stands for "clear the terminal"

a) After you worked through the previous exercises, clear the terminal

3 Files

This section introduces you to basic commands related to files.

wc

`wc` stands for "wordcount". It prints newline-, word-, and byte-counts for each file, and a summary line if more than one FILE is specified.

Syntax:

```
wc [OPTION]... [FILE]...
```

diff

`diff` stand for "difference", compare files line by line.

Syntax:

```
diff [OPTION]... [FILE]...
```

chown

`chown` stands for "change owner". It can also change group owner.

Syntax:

```
chown [OPTION]... [OWNER][:[GROUP]] FILE...
```

chmod

`chmod` stands for "change file mode bits" and controls the following permissions on any given file:

- Read: Who can see the file data.
- Write: Who can modify the file.
- Execute: Who is allowed to run the file (like a program or a script)

Permissions can be read using `ls -l`.

Syntax:

```
chmod [OPTION]... MODE[,MODE]... FILE...
```

echo

`echo` - output the given line of text (useful for displaying the content of variables)

Syntax:

```
echo [SHORT-OPTION]... [STRING]...
```

touch

`touch` - change file timestamps (access and modification times) to the current time or to a specific one defined through the options. Most useful for creating new, empty files.

Syntax:

```
touch [OPTION]... FILE...
-t STAMP`  use [[CC]YY]MMDDhhmm[.ss] instead of current time
```

Note: a FILE argument that does not exist is created empty (this mode of operation was already used in the section *basics*).

ln

`ln` - make links between files.

Syntax to create a link to `TARGET` with the name `LINK_NAME`:

```
ln [OPTION]... [-T] TARGET LINK_NAME
```

Note: If you use `ln` like this, it will create a so-called 'hardlink'. It is generally advisable to use softlinks (symlinks) instead, because they will also work across file systems. To do so, you will need to pass the `-s` flag like this: `ln -s TARGET LINK_NAME`. Be aware that you will have to use the *absolute* path for `TARGET`.

file

`file` - determine file type.

Syntax:

```
file [OPTION]... FILE...
```

less

`less` - A file viewer. `less` allows to scroll forward and backward in a file with the arrow keys. The current file can be opened in an editor by pressing `v`, and `less` can be quit by pressing `q`.

Syntax:

```
less [OPTION]... [filename]...
```

which (whereis)

`which` - shows the full path of (shell) commands.

Syntax:

```
which [options] [--] programname [...]
```

`whereis` - locate the binary, source, and manual page files for a command.

Syntax:

```
whereis [OPTIONS] name...
```

Exercise 1

a) For this set of exercises, create the temporary working directory `TempDir` in your home directory. Now, make it your current working directory.
b) Create the file `capture.txt` containing some text typed through your keyboard. *Hint:* Use a console text editor such as nano
c) How many lines, words and characters does this file have?
d) Make a copy of the file and compare both files.
e) Append text to the copied file and compare with the original.
f) Assign user `root` and group `users` as new owner of the original file and check the change. *Hint:* Check the current owners of the files first
g) Change the file attributes to make it write protected and try to append text again.
h) Generate a new line in the terminal which contains "Hello World!".
i) Change the date and time of the file `capture.txt` to 31.01.2017 00:01 and check the result.
j) Create a link to the `capture.txt` file in the desktop, minimize the terminal and look for the new link on the desktop. Can the text editor open the file?
k) Check the file types in the following directories: your test directory, downloads, pictures and music. Make a list of all those file types creating the `file-types.log` file.
l) Explore the log file `file-types.log` with the command `less`.
m) Obtain the location of the command `ls`, as well as the location of it's manual.
n) Delete the content of `TempDir` directory and check it was deleted. Then delete the directory.

4 Jobs

This section deals with processes. You will learn how to manage programs in the console. Especially, you will move them to back- and foreground and learn how to force terminate a blocked program.

Exercise 1: fg, bg, &, top, htop, kill, pkill, killall

a) Install `htop` if you do not already have it.
b) Navigate to the `jobs` directory.
c) Run the `infiniteloop.sh` script. Now your terminal should become unresponsive.
d) Find a way to stop the script without closing the window.

Now we want to learn how to kill processes running in the background. This is helpful when you have programs that have hung themselves up and have to be shut down. Collectively, the commands you will be using are known to provide "process control", that is, functionality similar to task managers known from Windows/OS X. So we will basically learn how to use Ctrl+Alt+Delete under Linux ;)

e) Run the script used in exercise c) again, but in a way such that it becomes a background job.
f) Now turn it into a foreground job while it's still running. Do you notice a difference?
g) Now move it back to be a background job. Can you still kill it as before?
h) Try and find out the PID (process id) of your started process (you might want to do this in a different terminal).
i) Use the kill command with the PID to get rid of the process.
j) Finally, start 3 instances of `infiniteloop.sh`, each of them running in the background, then use the `jobs` command. Now kill them in the following order, as presented by `jobs`: 2, 1, 3

Exercise 2: ps, kill, killall, pkill

Do the same as above but this time without using top or htop and use a different method to kill it. You also do not need to do all that background foreground stuff again.

Can you find the parent of the process you started?

5 Remote

In this section you will learn how to work on a machine which you have no physical access to (i.e. you cannot connect a keyboard, mouse or screen). Examples for such machines are: routers, Raspberry Pis, a computer that is 200km away, ETH's supercomputers et cetera... In particular, you will find out how to securely connect to those machines, run commands on them, how to copy files back and forth and see how many users are logged in.

Exercise 1: ssh, scp

In this exercise you will learn how to access a compute cluster and do stuff on it and get results. For this please first find the C program called `supercoolNumercialSimulation.c`. `ssh` and `scp` are the most important commands you will use within this exercise. `ssh` is used to connect to a machine remotely, while `scp` is used to copy files. For this exercise, it's probably best if you don't lose yourself in trying to understand how `ssh`, resp. `scp` works exactly, because there is quite a bit of cryptography and complicated network stuff involved. To get a general idea of how it works: `ssh` stands for "Secure Shell" and `scp` stands for "Secure Copy". Both make use of the cryptographic library `openSSL` in order to be able to exchange your data securely. If you are interested, you may also use the utility `sftp` for transferring files. If you do this, you need to check online for documentation (or run `man sftp`) since the solution to this exercise shows the `scp` method.

a) Connect to `pterodactyl.vsos.ethz.ch` with the user `hacker` via ssh (the password is `hacker`). If you are doing this from home, chances are that we already disabled ssh access, due to security reasons.
b) Make yourself familiar on the remote machine and make a directory with your name as the directory name.
c) Return to your local machine.
d) Copy the C program you found before to your directory on the remote machine.
e) Again access the machine and compile the program.
f) Run it.
g) Find its output and copy it to your `remoteExercise` directory on your local machine.

Exercise 2: tmux

This exercise is about using the terminal multiplexer `tmux`. Using it gives you several advantages, here are a few:

- If the connection gets lost, running processes on the remote machine will keep working independently.

- You can turn off your local computer without interrupting the remote calculation. You may re-connect to the remote tmux session any time to check how your calculations are doing.
- tmux allows you to tile your terminals and have several terminals open through the same ssh connection.
- You may collaborate with someone else and you both see the same content.

Consider looking up tmux keyboard shortcuts online.

a) Start tmux on your local machine.
b) Split your tmux horizontally (so that you have one terminal on top of the other).
c) Split the upper half vertically and try to navigate into each of the three parts.
d) Open up another terminal and connect to tmux (Caution: this is not the same thing as starting tmux).
e) Verify that whatever you do in one terminal is also reproduced in the other one.
f) Close both terminal windows, open up a terminal and attach. Verify that tmux kept running in the background and that it's still in the state in which you left it..
g) Close every bit of tmux until tmux is not running any more.
h) Connect to `pterodactyl.vsos.ethz.ch` as in Exercise 1a.
i) Type the command `w`, then hit Enter to see how many users are currently logged in.
j) Attach to tmux (if none is running, start it). Note that there might be an arbitrary number of people already logged in and all of you type in the same window. Have fun ;-)
k) Detach from tmux, but not by closing the terminal nor by exiting tmux. Check online for how to do this.
l) Log out from the remote server.

6 Searching

These exercises cover the `grep` *and* `find` *commands in detail.*

We're going look at two of the most important commands that perform "search-like" operations. Their names are `grep` and `find`. `find` is a command that is used for finding certain files. `grep` is a bit more complex and allows you to search a file's contents.

I would recommend that you do the exercises on both `grep` and `find` - they are very powerful when used in combination. If you have to choose only one, I recommend `grep` because I use it much more often.

Exercise 1: `grep`

`grep` searches the text using so-called Regular Expressions (commonly referred to as Regex). Regexes are textual patterns that encode a set of strings. In other words, a Regex is simply an efficient way to specify many strings. For example, `un(b|f)ounded` is a Regex that stands for both "unbounded" and "unfounded". There are many Regex standards, and they may differ in the special characters they use. It's probably a good idea to read https://en.wikipedia.org/wiki/Regular_expression#Basic_concepts. After that, the `grep` manual provides some further information.

Fun fact: `grep`'s name comes from *g*lobally search a *r*egular *e*xpression and *p*rint. This is a wordplay on the `ed` command g/re/p, which searches for a Regex and prints it. `ed` is one of the very first text editors created. Nowadays, no one uses `ed` anymore because there is no reason to do so (maybe apart from bragging rights), but `ed` has influenced a *lot* of commands commonly used in Unix.

a) Take a look at the file `HackingSessionExercises/grep_exercise.md`. In this exercise, you will search the file for various patterns using `grep`. The goal is that you use `grep` to output *only* what is written in the exercises.
b) Find the word "contains".
c) Come up with a pattern such that the output are the three lines `Dog`, `Dig` and `Dug`.
Hint: Take a look at the -E or the -P flag for `grep`. You will need one of them here.
d) Look for all lines beginning with `##`.
e) Print all lines beginning with `##`, but exclude the ones that starts with `###`.
f) Search for the lines `Cat`, `Rat`, `Sat` and so on, up until `Bat`, but without the line `Latin`.
g) Produce the same lines as in e), but this time with the line `Latin` included.
h) Try for the lines `Brat` up until `Bat`, but without the line `Latin`.
i) Adjust your command so that it lists the lines `Brat` up until `Bat`, this time with `Latin`.
j) Check for the three list elements (these are the lines beginning with `*`).
k) Investigate for the entire block of Java code (the one at the end of the file).
l) Within that code, output the class that contains the method `out`.
m) Pry for all animals that consist of exactly two words.

n) Probe for all animals that begin with `Danger`. *Note:* Take particular care that "Booplesnoot" and "Trash Panda" aren't part of your output. This makes it more tricky.
o) Finally, if you're still motivated: All animals that contain the word `Danger` *or* `danger`.

Exercise 2: `find`

In this exercise you'll use the `find` command to do some complicated file finding. You can "chain" filters, so for searching for a file that is called "asdf" and has size less than 50MB, you can do `find -name asdf -size -50M`. As usual, there are some more examples at the bottom of the manpage.

a) Navigate to your home directory.
b) Using `find`, search for all files that were modified less than 5 days ago.
c) Using `find`, search for all the files that you have write access for.
d) Search for all files that start with the letter `d`.
e) Search for all files smaller than 1MB, but larger than 10KB.
f) Find all files in the directory `Desktop` (or some different directory).
g) Search for all files starting with the letter `d` and execute `cat` on them.

Exercise 3

Combining the knowledge of `grep` and `find`, we can search for some incredibly specific stuff! Look at the `exec` flag in the manual for `find`.

a) Navigate to your etc directory (`/etc`, on most distributions).
b) Within that directory, search for all lines of text containing the word "root".
c) If you received a lot of error messages in exercise b), try to limit your search to files which you have read permissions for.
d) Search for all files containing an IPv4 address. (Ignore the fact that the adress ranges only from 0 to 255)
e) Output all comments (lines starting with a `#`) of all files with a name ending in `.conf`.

7 System Management

In this section, you will learn how to turn off your computer from the console, how to check the available disk space and how to manage users and groups from the shell. Note that when you use the shell for modifying a user, the system settings panel will also reflect these changes.

Exercise 1: Power on/off

You can use the shell to do various system management tasks from the command line. For example, you can use the command "poweroff" to shut down your computer immediately. This exercise explores some more examples: (you might need to use `sudo`)

a) Using your terminal, try to put your computer into suspend, and wake it back up. (*Hint:* Use the command `systemctl`)
b) Schedule your laptop to shut down in 10 minutes. Then cancel it.
c) Reboot your computer from the command line.

Exercise 2: Space Left on Disk

Useful commands: `df`, `du`

a) Find out what size (in MB) the exercise directory has.
b) Find out how much space is used by your root directory `/`.

Exercise 3: User Management

Useful commands: `useradd`, `userdel`, `groupadd`, `groupdel`, `gpasswd`, `passwd`

a) Add a new user called `ExerciseUser`.

b) Add a new group called `ExerciseGroup`.

c) Add your new user to the new group.

d) Remove the user and the group again.

Hint: If you are confused by `gpasswd` and `passwd`, consider the following: One of them is used to set the password for a user, the other is used to add a user to a group.

8 Block devices and file systems

This exercise is a bit more advanced! It will involve creating/editing partitions from the command line and manipulating block devices and file systems. Instead of using real disks, we will be working with files that "pretend" to be disks.

WARNING: If any of the following exercises require you to run them as root, you should not be doing that!

Exercise 1: Partition a disk

Useful tools: `cfdisk`, `gdisk`

For the purpose of this exercise, we provided a file called `TestDisk`. You can treat it like a regular block device (like `/dev/sda`), but using this file you will not damage your live system.

a) Create two partitions: One with 1MB size, one with the rest of the available space. Both should be of type `Linux filesystem`.

Exercise 2: Create a file system

Useful tools: `mkfs.ext4`, `mkfs.vfat`

For the purpose of this exercise, we provided two files: `partition1` and `partition2`. This will again not change your actual system.

a) Create a FAT file system on `partition1`.
b) Create a ext4 file system on `partition2`.

You can check if this worked by looking at the output of `file partition1` and `file partition2`.

Exercise 3: Using dd

WARNING: Don't do the following to your actual file systems! You can destroy your file systems! Especially don't run any of the commands as root.
With `dd` you can write images to a disk, like an image of a Linux distribution.

a) Write `image.iso` to `TestDisk` using `dd`.
You can check if this worked by looking at the output of `cat TestDisk`.

9 Software management

This exercise is about installing and managing software.

Exercise 1: Package manager

a) Refresh the sources of your package manager
b) Install updates for your system
c) If you are under Ubuntu, remove old software by using `autoremove` (otherwise skip this step)
d) Search for a package called `youtube-dl`
e) Install the package you found in the previous step
f) Run `youtube-dl` to see if it works. It should display something like "Usage: ..."
g) Open the man page for `youtube-dl`
h) Remove `youtube-dl` from your system
i) Look at the disk usage of your system partition (Hint: `df`). Then clean the cache of your package manager. Look at the disk usage again: some space should have been freed.

Exercise 2: Ubuntu only: Installing software from PPA

Skip this exercise if your distribution is not Ubuntu-based.

If a program is not in the official package sources, you might still be able to install it using your package manager. Under Ubuntu, a developer can create a Personal Package Archive (PPA), sort of a personal software channel which you can integrate into your sources list. After refreshing your sources, you may then install and upgrade software from the PPA as if it was an official source.

Caution: Anyone can own a PPA and distribute arbitrary software over it. Make sure you trust the developer!

In this exercise, we are going to install a video transcoder called Handbrake.

a) Google for "Handbrake PPA" and pick the Launchpad link
b) Use the command `add-apt-repository` to add the PPA to your sources as described on Launchpad
c) Update your software sources
d) Install `handbrake-gtk` using your package manager
e) In your start menu, search for "Handbrake" and open the program. Its logo shows a pineapple. The program should open.
f) Uninstall `handbrake-gtk`
g) We are now going to remove the PPA from the system. There is no command to do this. Edit the file `/etc/apt/sources.list` and remove the two lines that contain "stebbins".

h) Refresh your software sources again.
i) Attempt to install `handbrake-gtk`. There should be no such package.

Exercise 3: Installing a package manually

Even if there is no package available in the software source and there is no PPA available, there might be a package available online that you can download manually and install. This means that you download the package file using your browser or the `wget` command and then tell your package manager to install it. This way, the package can be removed **but not upgraded automatically** by your package manager.

Note: In this method you need to trust the developer since the package manager only verifies the integrity of the package, not the security.

In this exercise we are going to install the software TeamViewer. Note that this is a proprietary (non-free) software.

a) In your browser, go to https://www.teamviewer.com and visit the Downloads page
b) Select the version suitable for your OS and architecture (32- vs. 64-bit) and download it
c) Open a terminal, navigate to your Downloads folder and check that the file exists
d) Tell your package manager to install the file (use Tab completion when typing the filename teamviewer...)
e) Start `teamviewer` to check that the program works
f) Use your package manager to remove the package.

Exercise 4: Compile from source using git

Sometimes there is no package at all available for the software you are looking for. In this case, you need to go online and download a program manually. For Free and Open Source Software, you often get the sources and compile the program suitable for your OS and architecture.

In this exercise, we are going to download the source of a software called `f3` from Github, compile it into an executable program and install it. **Note that programs installed that way are not managed by your package manager.** You will need to update and uninstall them manually. Only use this option if you are sure that there is no package with your software for your distro. **Note:** When installing software this way, you will download and run code from a source that is not verified by your distro's community. **Only do this with software you trust.**

Before you start, you need to make sure your system has the required utilities to compile a program:

- Ubuntu-based: `apt-get install build-essential`
- OpenSUSE: `sudo zypper in --type pattern devel_basis`

a) In your browser, visit the GitHub page of `f3`: https://github.com/AltraMayor/f3

b) In the top right, there is a green button saying "Clone or Download". Copy the https URL to your clipboard.

c) In a terminal, go to your Downloads folder and type `git clone URL` where you *replace* `URL` by the URL you just copied in the previous step. This will create a new directory `f3` with the contents of the Github repository you just cloned.

d) Go into the `f3` directory and follow the instructions of the README to compile (make) and install the main software without the extras (you will type 2-3 commands).

e) Type `f3read` to verify that the program has been installed correctly. It should complain "f3read: The disk path was not specified"

f) We will now uninstall f3. Unfortunately, the developer has not included a script to uninstall f3, meaning that we must manually delete the files that were created by the install step. You probably get a feeling now why the package manager is always the preferrable option to install software. To locate the files we want to delete, we look at the output of the previously run install command. To help you, there are 4 files to be deleted:

- 2 files in `/usr/local/bin/`
- 2 files in `/usr/local/share/man/man1/`

Note: Every `sudo make install` is different and you need to figure out on your own how to undo its effects. Nice developers provide an option `sudo make uninstall`.

Exercise 5: Installing from a self-containing install script

It is possible to build setups for Linux. This means that you download some file (typically the file name ends with `.run`) and blindly execute it. The file self-contains all the data it needs and installs the program on your computer. Just like under Windows, **you don't know and have no control over what's going to happen when running such a file**. Outdated software may destroy your computer and you need to trust the developer.

In this exercise, we will install PostgreSQL from a `.run` archive. PostgreSQL can be installed with the package manager on most distros, but for the sake of training, we'll do the `.run` Method here.

a) In your browser, visit https://www.enterprisedb.com/downloads/postgres-postgresql-downloads#linux and download the newest version of the installer for your distro and architecture.
b) Open a terminal and navigate to your Downloads folder
c) Use `chmod` to make the freshly downloaded file executable
d) Attempt to run the installer as an unprivileged user (using `./postgres<TAB>`) where TAB is autocompletion by pressing the Tabulator key on your keyboard
e) Do the same thing as root. **Note that now you are executing an unknown program with super user privileges. This is bad practice and should be done as your very last choice!**
f) Click yourself through the installer. Leave default values as they are. Type any password when asked. Do not run Stack Builder.
g) The software is now installed in your `/opt` directory. To run it, execute the program `postgresql` which you can find in `/opt/PostgreSQL/YOUR.VERSION/bin/`. It should complain that it does not know where to find the server configuration file.
h) It's cumbersome to have to type the full path every time we want to start up the program. We can either add PostgreSQL's bin folder to the $PATH variable (out of scope of this exercise) or create a symlink to it. For the sake of the exercise, create a symlink to the executable under `usr/bin/postgres`
i) Verify that the program can now be run without typing the entire path
j) PostgreSQL comes with an uninstaller. You can find it right above the `bin/` directory, it's called `uninstall-postgresql`. Run it as root.
k) The installer tells you that the data directory and service user account have not been removed. Remove `opt/PostgreSQL` manually.
l) Remove the user `postgres` that the installer created

Note: Every `.run` is different and you need to figure out on your own what needs to be done to undo it (if possible).

10 License