Software and packages

In Windows we have several options to install software. We usually use installers that we find on discs and websites or even the Windows app store where we can find software in an online catalog.

Remember that latest hyped video game that you preordered and turned out to be a total bust? When installing that game it might prompt you saying/asking that you need to install the lastest version of DirectX or Visual C++ Redistributable? These are other pieces of software that are needed to run the initial application or game. We call these pieces of software *dependencies*. Most of the times the original installer installs these for us but sometimes we have to manually find and install these dependencies ourselves.

Installing software on Linux systems hasn't always been easy. Back in the days we had to download source code and compile applications ourselves, put them in the right folders and make sure we have all the needed dependencies to run the application. We might come across this process in the present day, but most of the time we are gonna install software using *package managers*. These are tools that run through a database to find the application that we want to install. If it finds an application matching the specified name it will install the application as well as all the required dependencies. If we remove an application it will also remove all dependencies that are no loger required. Another benefit is that the package manager will also manage updates of all our applications and dependencies.

installing, removing and updating software (apt)

When installing software packages in Ubuntu we often use the `apt` (advanced package system) command. The man page gives us all the info we need to use the command:

```bash
student@linux-ess:~$ man apt
APT(8)                                                    APT

NAME
       apt - command-line interface

SYNOPSIS
       apt [-h] [-o=config_string] [-c=config_file] [-t=target_release] [
           update | install pkg [{=pkg_version_number | /target_release}]
           full-upgrade | edit-sources | {-v | --version} | {-h | --help}

DESCRIPTION
       apt provides a high-level commandline interface for the package ma
       user interface and enables some options better suited for interact
       specialized APT tools like apt-get(8) and apt-cache(8).
```

As described above `apt` (or it's predecessor `apt-get`) is a package manager. We can use this tool to install packages (read: software) on our Linux machine. Note that `apt` is the specific package manager for Ubuntu. There are several alternatives available such as `dpkg`, `pacman`, `rpm`, `yum`, `dnf` ... which might come pre-installed with a specific linux distribution.

Repositories

An important thing to note about `apt` is that it uses a database of available packages. We can update this list of packages by running the command below:

```bash
student@linux-ess:~/globbing$ sudo apt update
[sudo] password for student:
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Hit:2 http://archive.ubuntu.com/ubuntu focal InRelease
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security/main amd64 Package
Get:5 http://security.ubuntu.com/ubuntu focal-security/main Translation-e
Get:6 http://security.ubuntu.com/ubuntu focal-security/main amd64 c-n-f M
```

```
Get:7 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 P
Get:8 http://security.ubuntu.com/ubuntu focal-security/restricted Transla
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 c
Get:10 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Pa
Get:11 http://security.ubuntu.com/ubuntu focal-security/universe Translat
Get:12 http://security.ubuntu.com/ubuntu focal-security/universe amd64 c-
...
Get:28 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 c-
Get:29 http://archive.ubuntu.com/ubuntu focal-backports/main amd64 Packag
Get:30 http://archive.ubuntu.com/ubuntu focal-backports/main Translation-
Get:31 http://archive.ubuntu.com/ubuntu focal-backports/main amd64 c-n-f
Get:32 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Pa
Get:33 http://archive.ubuntu.com/ubuntu focal-backports/universe Translat
Get:34 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 c-
Fetched 8672 kB in 2s (3651 kB/s)
```

ℹ Note that we used the `sudo` command. This is needed because `apt` is a system wide command that impacts the entire system (installing/removing/updating software). Therefore we cannot run it as a user with default permissions.

We can see that this command gets data from a bunch of servers. These servers are called *repositories* (servers with a collection of packages). When we install software later, it will use the database based on the repositories to check if the package that we want to install is available.

The list of repositories that `apt` uses can be found in the file `/etc/apt/sources.list` . We can manually add more repositories to this file or use the command `add-apt-repository` but we will not go into this for now.

Installing software (apt install)

Imagine we would like to install the `zip` package. We could simply run the command below:

bash

```
student@linux-ess:~$ sudo apt install zip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  unzip
The following NEW packages will be installed:
  unzip zip
0 upgraded, 2 newly installed, 0 to remove and 175 not upgraded.
Need to get 336 kB of archives.
After this operation, 1231 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu jammy/main amd64 unzip amd64 6.0-2
Get:2 http://archive.ubuntu.com/ubuntu jammy/main amd64 zip amd64 3.0-11b
Fetched 336 kB in 1s (444 kB/s)
Selecting previously unselected package unzip.
(Reading database ... 32259 files and directories currently installed.)
Preparing to unpack .../unzip_6.0-25ubuntu1_amd64.deb ...
Unpacking unzip (6.0-25ubuntu1) ...
Selecting previously unselected package zip.
Preparing to unpack .../zip_3.0-11build1_amd64.deb ...
Unpacking zip (3.0-11build1) ...
Setting up unzip (6.0-25ubuntu1) ...
Setting up zip (3.0-11build1) ...
Processing triggers for man-db (2.9.1-1) ...
```

If we analyse the output of the command we can see a couple of this happening:

- It reads the package list to find the `zip` package. After this is done it builds a dependency tree to see what dependencies the `zip` package needs.
- It points out that it will install an additional package called `unzip`. This is a dependency of the `zip` command.
- It points out what new packages will be installed and if any packages will be updated.
- It asks for a confirmation if we are sure we want to continue. After this confirmation it will install / update all the packages mentioned in the output

above.

ℹ️ You might notice that you can use both the `apt` and `apt-get` command. Both commands are very similar (with minor differences, but we won't get into that now).

After running the command above, we can succesfully use the `zip` command:

bash

```
student@linux-ess:~$ zip
Copyright (c) 1990-2008 Info-ZIP - Type 'zip "-L"' for software license.
Zip 3.0 (July 5th 2008). Usage:
zip [-options] [-b path] [-t mmddyyyy] [-n suffixes] [zipfile list] [-xi
  The default action is to add or replace zipfile entries from list, whic
  can include the special name - to compress standard input.
  If zipfile and list are omitted, zip compresses stdin to stdout.
```

ℹ️ You might wonder where the `zip` executable is located and how the shell knows how to run that exact executable. We can check this by running the `which zip` command. This will tell us that, when we run the `zip` command it will actually run the `zip` binary (=executable) located at `/usr/bin/zip`. `/usr/bin` is one of the folders where binaries (=executable files) are stored. You could compare this to the folder `c:\program files` in Windows systems. Linux has multiple places where binaries are stored. These are often bundled in the `$PATH` variable which we will learn to use in a later chapter.

Imagine we made a typo in our command and try to install a package that doesn't exist:

bash

```
student@linux-ess:~$ sudo apt install zap
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

```
E: Unable to locate package zap
```

The output indicates that it cannot locate the package `zap`. This means that it checked all its repositories for a package with the name `zap` but it could not be found. If the package does exists we would have to add the repository containing the package and run `sudo apt update`.

Removing software (apt remove)

To remove software we can simply use the command below:

bash

```
student@linux-ess:~$ sudo apt remove zip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer requir
 unzip
Use 'sudo apt autoremove' to remove it.
The following packages will be REMOVED:
 zip
0 upgraded, 0 newly installed, 1 to remove and 175 not upgraded.
After this operation, 638 kB disk space will be freed.
Do you want to continue? [Y/n] y
(Reading database ... 32291 files and directories currently installed.)
Removing zip (3.0-11build1) ...
Processing triggers for man-db (2.9.1-1) ...
```

Notice how it doesn't automatically remove the dependencies we talked about earlier. This is because there might be other packages that require this dependency. We can use the `sudo apt autoremove` command to remove unused dependencies.

We also have a somewhat more agressive command to remove packages: `sudo apt purge <packagename>`. The difference between `remove` and `purge` is that when using `purge` it will remove any config files linked to that

application as well. When using `remove` those config files will stay on the system.

> ℹ To get a list of all possible software packages we can use `apt list`. To get a list of installed software packages we can use `apt list --installed`. Warning: this list can be very very long!

Updating software (apt upgrade)

For updating our software we have a couple of options. To start of we can individually update certain packages by simply using the `sudo apt install <packagename>` command. When you use this command with a package that has already been installed, it will try to update it to the latest version.

> Mind that before updating or even installing software you might want to update your database by running `sudo apt update` so you are certain we use the latest and correct repositories. *This might also be confusing at first because the `update` command doesn't actually update packages!*

When we want to update all the packages on our system we can use the `sudo apt upgrade` command. This will update all the installed packages.

gzip and tar

gzip

If we compress a file, we keep the content, but the filesize will become smaller. We can compress a file with gzip.

bash

```
student@ubuntu-server:~$ cd
student@ubuntu-server:~$ cp /var/lib/apt/lists/be.archive.ubuntu.com_ubun
student@ubuntu-server:~$ ls -lh be.archive*
```

```
-rw-r--r-- 1 student student 62M Oct 10 19:40 be.archive.ubuntu.com_ubunt
student@ubuntu-server:~$ gzip be.archive.ubuntu.com_ubuntu_dists_jammy_un
student@ubuntu-server:~$ ls -lh be.archive*
-rw-r--r-- 1 student student 17M Oct 10 19:40 be.archive.ubuntu.com_ubunt
```

The file has an extension of gz and is a lot smaller now, but you cannot see its contents. The original file is gone and we only have the compressed file now.

gunzip

If we want to decompress the file we can use gunzip.

bash

```
student@ubuntu-server:~$ ls -lh be.archive*
-rw-r--r-- 1 student student 17M Oct 10 19:40 be.archive.ubuntu.com_ubunt
student@ubuntu-server:~$ gunzip be.archive.ubuntu.com_ubuntu_dists_jammy_
student@ubuntu-server:~$ ls -lh be.archive*
-rw-r--r-- 1 student student 62M Oct 10 19:40 be.archive.ubuntu.com_ubunt
```

The file regained its original size and the content can be viewed/edited again. The zipped file is gone again and we only have the original file.

tar

If we want to put multiple files and folders together in one file, we can use tar. We use the option -c to create the tarfile.

bash

```
student@ubuntu-server:~$ tree
.
├── Downloads
│   ├── logo.png
│   └── Steam
│       └── games
│           └── pacman
├── emptyfile
```

```
student@ubuntu-server:~$ tar -cf Downloads.tar Downloads     # create tar
student@ubuntu-server:~$ tar -tf Downloads.tar               # view conte
Downloads/
Downloads/logo.png
Downloads/Steam/
Downloads/Steam/games/
Downloads/Steam/games/pacman
```

tarball

We can also zip the tar file when we create it.

bash

```
student@ubuntu-server:~$ tree
.
├── Downloads
│   ├── logo.png
│   └── Steam
│       └── games
│           └── pacman
├── emptyfile
student@ubuntu-server:~$ tar -czf Downloads.tar.gz  Downloads
student@ubuntu-server:~$ tar -tzf Downloads.tar.gz
Downloads/
Downloads/logo.png
Downloads/Steam/
Downloads/Steam/games/
Downloads/Steam/games/pacman
```

Extract a tar

We can extract a tarfile (.tar) or tarball (.tar.gz).

bash

```
student@ubuntu-server:~$ tree
.
```

```
├── Downloads
│   ├── logo.png
│   └── Steam
│       └── games
│           └── pacman
├── Downloads.tar
├── Downloads.tar.gz
├── emptyfile

student@ubuntu-server:~$ mkdir backup
student@ubuntu-server:~$ tar -xzf Downloads.tar.gz  -C backup      # zipp
student@ubuntu-server:~$ tar -xf Downloads.tar -C /tmp             # tarf
student@ubuntu-server:~$ tree
.
├── backup
│   └── Downloads
│       ├── logo.png
│       └── Steam
│           └── games
│               └── pacman

...

student@ubuntu-server:~$ tree /tmp/Downloads/
/tmp/Downloads/
├── logo.png
└── Steam
    └── games
        └── pacman

2 directories, 2 files
```

Installing packages via a tarball

Sometimes you might encounter a `tar` or `tar.gz` file for installing some software. A `tar.gz` file is also known as a `tarball` . it's a compressed file containing all kinds of files & folders. You can compare these to `zip` or `rar`

files. We need to extract the files/folders inside this file before we can use them. To do so we can use the `tar` command. The options we use will vary depending on the case. Do we want to extract a `tar` file or a `tar.gz` file:

bash

```
student@linux-ess:~/tarexample$ ls
app.tar.gz  docs.tar
student@linux-ess:~/tarexample$ tar -xf docs.tar
student@linux-ess:~/tarexample$ mkdir app
student@linux-ess:~/tarexample$ tar -xzf app.tar.gz -C ./app
student@linux-ess:~/tarexample$ ls
app  app.tar.gz  docs  docs.tar
```

The options we use can be found in the manpage, but below you can find a small summary of the most important ones:

- `-x` : Stands for extract, this will unpack all the contents of the file.
- `-f` : Stands for file. This uses the argument after the options as the path of the archive we want to extract.
- `-z` : This makes sure we run the file through `gzip` , another archive tool. The extention `tar.gz` hints that it is processed through `gzip` .
- `-C` : This changes the directory before extracting to the path supplied as an argument. This means the contents will be extracted in this folder.
- `-t` : Will only list the contents of the archive to the screen. It will not unpack anything.

dpkg

`dpkg` was the first package manager for Debian-based systems. Where Ubuntu nowadays uses `apt` as the default package manager, we can also use `dpkg` . `dpkg` was the predecessor of `apt-get` and `apt` . `dpkg` doesn't make use of repositories, so we have to have the installer file before using the command. It also doesn't download dependencies automatically.

That's why they used to call it *the dependancy hell* We can use `dpkg` to install / remove / ... `.deb` files. The example below installs the package `yourpackage` :

```bash
student@linux-ess:~$ sudo dpkg -i yourpackage.deb
```

`dpkg` can also be used to (re)configure particular packages. We can use this for example to change the keyboard layout on our server by running:

```bash
student@linux-ess:~$ sudo dpkg-reconfigure keyboard-configuration
```

snap

One of the relative new players is snap. A snap is a bundle of the software we want to install with all of its dependencies stored in one file and executed in its own bubble. This means that two snaps cannot interfere with eachother. This for example makes it possible to install two different versions of the same software at the same time and running them together. Snaps have their own filesystem but can work with files on your systems too. You can find snaps in the snap store on a Desktop or with `snap search` on a server. Snaps are used more and more because they are distro independant. If you can install the snap daemon on the distro it will run all snaps.

The commands for working with snap are almost similar as apt. We have:
snap search
snap list (installed apps)
snap list --all (shows all the installed versions)
snap revert --revision (go back to a previous version of the snap)
snap info
snap install
snap remove
snap refresh (upgrades the snaps, happens every day automatically)
snap changes (history of changes)

snap version

snap update does not exist. the snap daemon checks for updates 4 times a day.

You can install from different channels (like stable, beta,...) but the standard is stable and we keep it that way.

bash

```
student@linux-ess:~$ snap search mapscii
Name     Version  Publisher  Notes  Summary
mapscii  0.3.1    nhaines    -      The whole world in your console.
student@linux-ess:~$ snap info mapscii
name:      mapscii
summary:   The whole world in your console.
publisher: Nathan Haines (nhaines)
store-url: https://snapcraft.io/mapscii
contact:   https://github.com/nathanhaines/mapscii
license:   MIT
description: |
  A node.js based Vector Tile to Braille and ASCII renderer for
  xterm-compatible terminals.
snap-id: YmktiCRE0Dg3FvsSwXiNFHeygSzRtyIo
channels:
  latest/stable:    0.3.1 2020-08-12 (14) 29MB -
  latest/candidate: ↑
  latest/beta:      ↑
  latest/edge:      0.3.1 2020-08-15 (14) 29MB -
student@linux-ess:~$ sudo snap install mapscii
[sudo] password for student:
mapscii 0.3.1 from Nathan Haines (nhaines) installed
student@linux-ess:~$ mapscii
```

zip vs gzip

`gzip` compresses only one file and therefore is used with `tar` which brings multiple files into one file.
`zip` is used when you want to compress a bunch of files into one file in one

step.

```bash
student@linux-ess:~$ ls -a
.   ..   .bash_history   .bash_logout   .bashrc   .profile   .ssh
student@linux-ess:~$ zip myzipfile.zip .bashrc .profile
  adding: .bashrc (deflated 54%)
  adding: .profile (deflated 51%)
student@linux-ess:~$ ls -l
total 2440
-rw-rw-r-- 1 student student    2440 Oct 14 15:00 myzipfile.zip
student@linux-ess:~$ unzip -l myzipfile.zip     # only list the files
Archive:  myzipfile.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     3771  2022-01-06 16:23   .bashrc
      807  2022-01-06 16:23   .profile
---------                     -------
     4578                     2 files
student@linux-ess:~$ rm .bashrc .profile
student@linux-ess:~$ ls -a
.   ..   .bash_history   .bash_logout   .ssh
student@linux-ess:~$ unzip myzipfile.zip
Archive:  myzipfile.zip
  inflating: .bashrc
  inflating: .profile
student@linux-ess:~$ ls -a
.   ..   .bash_history   .bash_logout   .bashrc   .profile   .ssh
```

So why do we use `tar` and `gzip` if we can use `zip` instead? This is because `tar` will also keep the ownerships and the rights on each file. We will compare the two with an example (if you don't really understand the ownership in the example, mind that it will be explained in a later lesson):

```bash
student@linux-ess:~$ ls -lh .bashrc
-rw-r--r-- 1 student student 3.7K Jan  6  2022 .bashrc        # mind the
student@linux-ess:~$ zip zipdemo.zip .bashrc
```

```
  adding: .bashrc (deflated 54%)
student@linux-ess:~$ tar -czf tardemo.tgz .bashrc
student@ulinux-ess:~$ ls -lh zipdemo.zip tardemo.tgz
-rw-rw-r-- 1 student student 1.9K Oct 14 15:21 tardemo.tgz
-rw-rw-r-- 1 student student 1.9K Oct 14 15:19 zipdemo.zip
student@ulinux-ess:~$ cp zipdemo.zip tardemo.tgz /tmp
student@ulinux-ess:~$ cd /tmp
student@ulinux-ess:/tmp$ sudo unzip zipdemo.zip              # sudo -> do
student@ulinux-ess:/tmp$ ls -lh .bashrc
-rw-r--r-- 1 root root 3.7K Jan  6  2022 .bashrc             # root is th
student@ulinux-ess:/tmp$ sudo rm .bashrc
student@ulinux-ess:/tmp$ sudo tar -xzf tardemo.tgz            # sudo -> d
student@ulinux-ess:/tmp$ ls -lh .bashrc
-rw-r--r-- 1 student student 3.7K Jan  6  2022 .bashrc       # student is
```

Previous
5 File contents

Next
Lab