Snap for dummies

Crownstone B.V

Ilhan Delic
0914619
september 2019

Contents

1   introduction

This document is a manual on how to start making snap packages on ubuntu and how to publish it. The main reason for doing this is to run a selfmade snap on a RaspberryPi 3 with ubuntu core installed. Ubuntu core exclusively uses snaps. The reason for choosing for using core over another ubuntu flavour is because of the advantages for use in IoT. Snaps are individually installed and updated. In order for us to use ubuntu core we need an understanding in making and publishing snaps.

2 installation

The first thing to do is to open up the terminal and update ubuntu with the command:

"sudo apt update"

the next thing to do is to install build essential:

"sudo apt install build-essential"

we will now install snapcraft, the utility for building snaps:

"sudo snap install - -classic snapcraft"

NOTE:
- if the snap command is not available, install the snapdpackage via APT
- the –classic switch enables the installation of a snap that uses classic confinement. snap confinement will be discus later on.
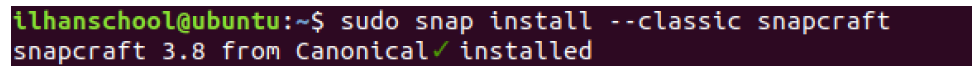

Figure 1: install snapcraft

3 Snap basics

This chapter covers the basic operations you can execute with snap. First if you're using a dektop ubuntu make sure you've installed snapd.
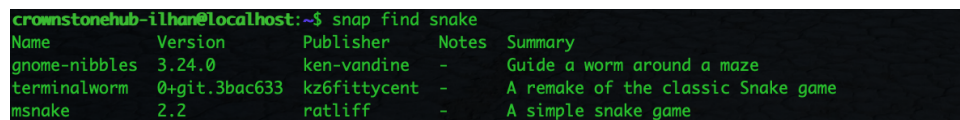
"sudo apt install snapd"

If you're using ubuntu core you don't need to install snapd because you already have snapd, you can not use apt-get in core unless you install the "classic" snap this will be discuss later on. For now i'm just showing a couple of basics snap operators.

3.1 Finding and installing snap

Finding a snap:

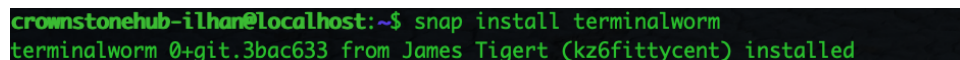"snap find< $searchtext$ >"

```
crownstonehub-ilhan@localhost:~$ snap find snake
Name            Version        Publisher     Notes  Summary
gnome-nibbles   3.24.0         ken-vandine   -      Guide a worm around a maze
terminalworm    0+git.3bac633  kz6fittycent  -      A remake of the classic Snake game
msnake          2.2            ratliff       -      A simple snake game
```

Figure 2: snap find

Installing snaps can be done by using the following command:

"sudo snap install< $package$ >"

```
crownstonehub-ilhan@localhost:~$ snap install terminalworm
terminalworm 0+git.3bac633 from James Tigert (kz6fittycent) installed
```

Figure 3: snap install

3.2 Keeping track of your snaps

If you'd like to take a look at the snaps you've downloaded use:

"snap list"

Figure 4: snap list

To take a look at the history of the changes you've made enter the following command:

"snap changes"

Figure 5: snap change

3.3 Snap info

If you'd want to run a snap but don't know how you can check the commands of the snap and more information:

"snap info $< snapname >$"

Figure 6: snap info

3.4 Upgrading and downgrading your snap

You could also choose to upgrade or downgrade the snaps you downloaded(of it's possible) to update use:

"sudo snap refresh $< package >$"

To see which snap update is ready to be installed type in:

"sudo snap refresh - -list"

Figure 7: snap refresh list

If for some reason you'd like to undo a update and return to a previous version of the snap use the command:

"sudo snap revert $< package >$"

3.5 Removing a snap

When you're done playing with your snap and feel like you don't need it you can just remove it from your device by using the following line:

"sudo snap remove $< package >$"
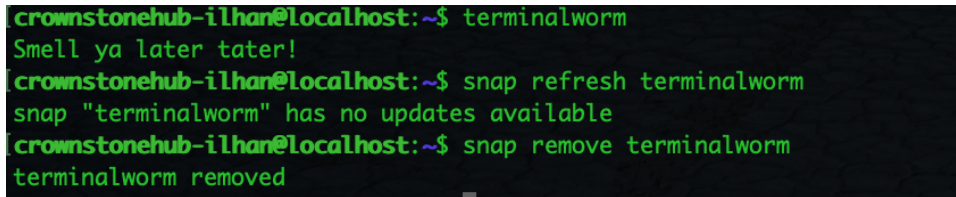
Figure 8: snap remove

3.6 Snap channels

Snap also has a feature called channels. By default, Snap packages are installed from the 'stable' channel. But there are few other channels that give you access to the development version of a program. It's like switching branches in git, if you are familiar with software development.
These channels are:

stable: The latest stable release of an application

candidate: The release candidate of an application that is reaching the stable version

beta: Unstable version that has reached a certain milestone

edge: Daily/nightly build of an application under development

Needless to say that you should stay on the Stable channel but if you really want to change to another channel, you can use Snap command in the following fashion:

"sudo snap refresh $< package >$ - -channel=$< channelname >$"

Once you changed the channel, your installed package will get updates from that channel. You can switch back to the old channel either by using the refresh command as shown above or simply use the revert command shown above.

4 Starting the snap

To start of you have to create a general snaps directory followed bij a working directory for this specific snap project. Using the following commands:

<div align="center">

"mkdir -p / name directory / name snap "

e.g. "mkdir -p /mysnaps/hello"

"cd mysnaps/hello"

</div>

It is from within this hello directory where we will invoke all subsequent commands.
Get started by initialising your snap:

<div align="center">

"snapcraft init"

</div>
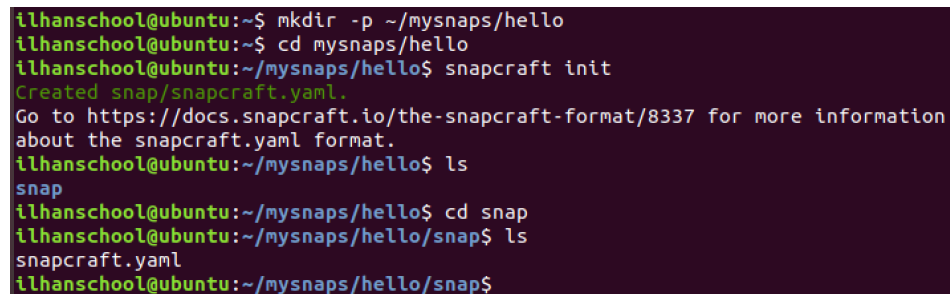
This created a snapcraft.yaml in which you declare how the snap is built and which properties it exposes to the user. We will edit this later.
The directory structure looks like this:
mysnaps/
hello/
snap/
snapcraft.yaml
NOTE:
Any future snap would be put in its own directory under mysnaps.
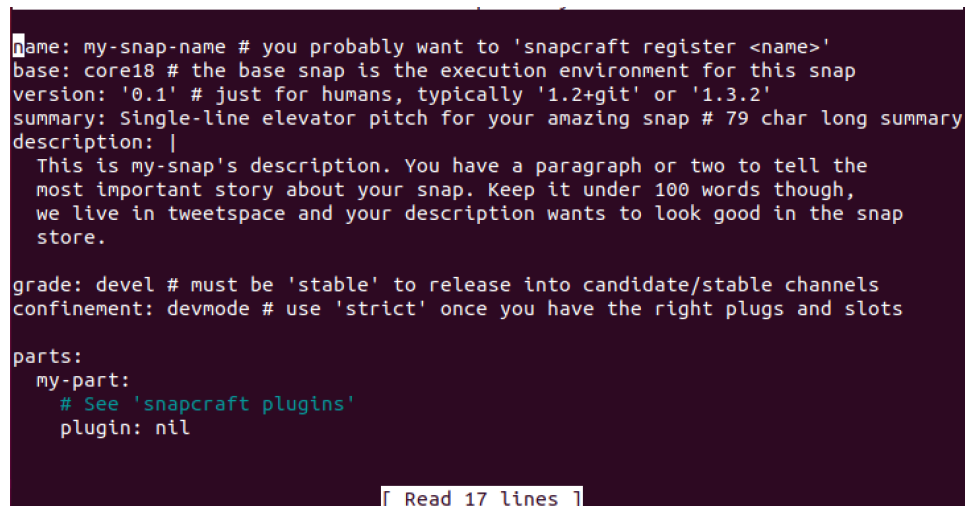
```
ilhanschool@ubuntu:~$ mkdir -p ~/mysnaps/hello
ilhanschool@ubuntu:~$ cd mysnaps/hello
ilhanschool@ubuntu:~/mysnaps/hello$ snapcraft init
Created snap/snapcraft.yaml.
Go to https://docs.snapcraft.io/the-snapcraft-format/8337 for more information
about the snapcraft.yaml format.
ilhanschool@ubuntu:~/mysnaps/hello$ ls
snap
ilhanschool@ubuntu:~/mysnaps/hello$ cd snap
ilhanschool@ubuntu:~/mysnaps/hello/snap$ ls
snapcraft.yaml
ilhanschool@ubuntu:~/mysnaps/hello/snap$
```

<div align="center">

Figure 9: snapcraft directory

</div>

4.1 describing the snap

Let's take a look at the top part of your snapcraft.yaml file. Use "nano snapcraft.yaml" while you're in the mysnaps/hello/snap directory. It should look somewhat as shown below:


```
name: my-snap-name # you probably want to 'snapcraft register <name>'
base: core18 # the base snap is the execution environment for this snap
version: '0.1' # just for humans, typically '1.2+git' or '1.3.2'
summary: Single-line elevator pitch for your amazing snap # 79 char long summary
description: |
  This is my-snap's description. You have a paragraph or two to tell the
  most important story about your snap. Keep it under 100 words though,
  we live in tweetspace and your description wants to look good in the snap
  store.

grade: devel # must be 'stable' to release into candidate/stable channels
confinement: devmode # use 'strict' once you have the right plugs and slots

parts:
  my-part:
    # See 'snapcraft plugins'
    plugin: nil
```
```
[ Read 17 lines ]
```

Figure 10: default text

This part of snapcraft.yaml is mandatory and is basic metadata for the snap. Let's go through this line by line:

NAME: The name of the snap.

VERSION: The current version of the snap. This is just a human readable string. All snap uploads will get an incremental snap revision, which is independent from this version. It's separated so that you can upload multiple times the same snap for the same architecture with the same version. See it as a string that indicates to your user the current version, like "stable", "2.0", etc.

SUMMARY: A short, one-line summary or tag-line for your snap.
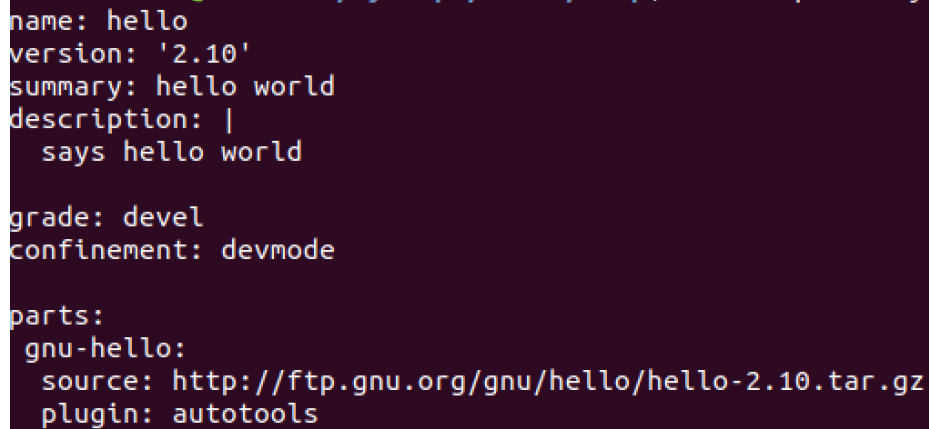
DESCRIPTION: A longer description of the snap. It can span over multiple lines if prefixed with the '—' character.

GRADE: Can be used by the publisher to indicate the quality confidence in the build. The store will prevent publishing 'devel' grade builds to the 'stable' channel.

CONFINEMENT: Can be either 'devmode', 'strict', or 'classic'. A newly-developed snap should start out in devmode. Security requirements can get in the way during development and 'devmode' eases these requirements. Security aspects like confinement can be addressed once the snap is working. If there are technical issues that obstruct the confinement of a snap, it may also be developed and released using classic confinement. Classic imposes no additional restrictions and effectively grants device ownership to the snap. Consequently, snaps using classic confinement require a manual review before being released to the store (see Classic confinement review process for further details).However, in this tutorial we will only focus on devmode and strict confinement.

5   Customisation

That's it for the basics, now you're going to customise your snap.
Use the command "nano snapcraft.yaml" to edit the text to make
snapcraft.yaml look like this:

```
name: hello
version: '2.10'
summary: hello world
description: |
  says hello world

grade: devel
confinement: devmode

parts:
 gnu-hello:
  source: http://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
  plugin: autotools
```

Figure 11: snapcraft.yaml 1.0

Note:
- Make sure you set a space after the colon(":").
- Use the spacebar instead of tabs to set the outline
- Version information is for snap user consumption only, and has no effect on
snap updates. It's defined within quotes, ('2.10'), because it needs to be a
YAML string rather than a floating-point number. Using a string allows for
non-numeric version details, such as 'myfirstversion' or '2.3-git'.
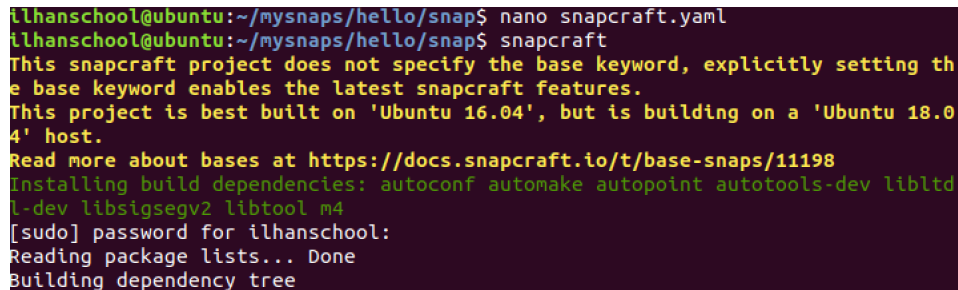
5.1 Adding a part

A snap can consist of multiple parts. Here are a few examples of this:
Snaps with separate logical parts, e.g. a server snap, which contains a web
server, a database and the application itself or a game which ships the game
engine and game data for three different games, each one being defined in its
own part. Snaps with parts which come from different locations. Parts which
are built in a different way. Our hello snap will be nice and simple. It will
consist of only one part for now. In the following pages we are going to
gradually extend it.

Two must-haves for every part are the 'source' and 'plugin' definition. Think
of these as the "what" and the "how", respectively. As source you can, for
example, pick a source repository (like git), a tarball, or a local directory.
Snapcraft supports many plugins, allowing you to build a wide variety of
project types (e.g. autotools, cmake, go, maven, nodejs, python2, python3).
To build hello the snapcarft.yaml file needs to look like this: 10
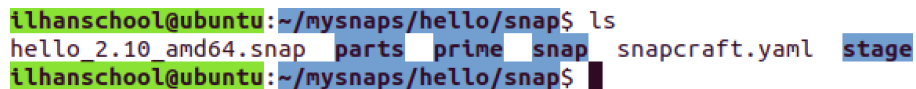
To build the snap use the command:

"snapcraft"

now you should get this:

Figure 12: snapcraft command

Figure 13: the snap directory after the first build

11

Next install the snap you've just build:

$$\text{"sudo snap install - -devmode} < name|_s nap > 11\text{"}$$

Like so:

Figure 14: snap install 1

Snap list shows the information given in the snapcraft.yaml file. Now we're able to build the snap but it doesn't have any commands to run. We'll need to add another part to the snapcraft.yaml

Open the snapcraft.yaml file and make it look like this to add an command to run:

```
name: hello
version: '2.10'
summary: hello world
description: |
  says hello world

grade: devel
confinement: devmode

apps:
 hello:
  command: bin/hello

parts:
 gnu-hello:
  source: http://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
  plugin: autotools
```

Figure 15: snapcraft.yaml 2.0

Now the snap can finally be build. Use the following commands:

"snapcraft prime"

What we did was tell snapcraft to run the build up until the "prime" step. That is, we are omitting the "pack" step (see lower down for an explanation of each step in a snapcraft lifecycle 14). What this invocation gives therefore are the unpacked contents of a snap.
We can then provide this content to "snap try":

"sudo snap try - -devmode prime"

```
ilhanschool@ubuntu:~/mysnaps/hello/snap$ snapcraft prime
This snapcraft project does not specify the base keyword, explicitly setting the base keyword
enables the latest snapcraft features.
This project is best built on 'Ubuntu 16.04', but is building on a 'Ubuntu 18.04' host.
Read more about bases at https://docs.snapcraft.io/t/base-snaps/11198
Skipping pull gnu-hello (already ran)
Skipping build gnu-hello (already ran)
Skipping stage gnu-hello (already ran)
Skipping prime gnu-hello (already ran)
The requested action has already been taken. Consider
specifying parts, or clean the steps you want to run again.
ilhanschool@ubuntu:~/mysnaps/hello/snap$ sudo snap try --devmode prime
hello 2.10 mounted from /home/ilhanschool/mysnaps/hello/snap/prime
ilhanschool@ubuntu:~/mysnaps/hello/snap$ hello
Hello, world!
ilhanschool@ubuntu:~/mysnaps/hello/snap$
```
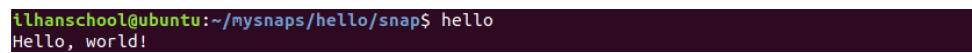
Figure 16: snap try output

13

Note:
The different steps of the snapcraft lifecycle are: "pull" (download source for all parts), "build", "stage"
(consolidate installed files of all parts), "prime" (distill down to just the desired files), and "pack" (create a snap out of the prime/ directory). Each step depends on the successful completion of the previous one.

Things should be working now, test it by using the following:

"snapcraft prime"

```
ilhanschool@ubuntu:~/mysnaps/hello/snap$ hello
Hello, world!
```

Figure 17: hello output

Important:
If hello does not run and you get the error cannot change current working directory to the original directory: No such file or directory then most likely you are developing the snap in a directory other than your home directory. An example of a directory that would generate this error, is the /tmp/ directory. Fix it by uninstalling the snap with sudo snap remove hello and starting over.

5.2 Adding another part

In order to make your snap do more than one thing you should add more parts to it so you can run more than 1 command. Open de snapcraft.yaml and make it look like this:

```
name: hello
version: '2.10'
summary: hello world
description: |
  says hello world

grade: devel
confinement: devmode

apps:
 hello:
  command: bin/hello
 bash:
  command: bash

parts:
 gnu-hello:
  source: http://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
  plugin: autotools
 gnu-bash:
  source: http://ftp.gnu.org/gnu/bash/bash-4.3.tar.gz
  plugin: autotools
  configflags: ["--infodir=/var/bash/info"]
```

Figure 18: snapcraft.yaml 3.0

Make sure you add the last line to avoid this error:
"Failed to stage: Parts 'gnu-bash' and 'gnu-hello' have the following files, but with different contents: share/info/dir"
What does this mean? Both our gnu-hello and gnu-bash parts want to ship a version of share/info/dir with differing contents. The two most common ways to rectify this kind of problem are:

- Instruct only one of the two parts to place content in this directory. We can also tell both to supress the content. Which solution depends on the necessity of said content. Either is achieved by influencing the 'snap' and 'stage' steps.
- Change the directory location for one of the two parts.

By adding the last line in de snapcraft.yaml you avoid this error by changing the directory of the new part.

6 Remove devmode

One last thing you might want to do before the snap is ready for wider consumption is to remove the devmode status.

Important:
Users of snaps using devmode, will need to pass –devmode during the installation, so they explicitly agree to trust you and your snap. Another benefit of removing devmode is that you will be able to ship your snap on the 'stable' or 'candidate' channels (you can only release to the other channels, like 'beta' or 'edge' as your snap is less trusted) and users will be able to search for it using snap find.

For this to be declared in your snap, let's set confinement to "strict" in the snapcraft.yaml.

```
name: hello-crownstonehub-ilhan
version: '2.10'
summary: hello world
description: |
  says hello world

grade: stable
confinement: strict

apps:
 hello:
  command: bin/hello
 bash:
  command: bash

parts:
 gnu-hello:
  source: http://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
  plugin: autotools
 gnu-bash:
  source: http://ftp.gnu.org/gnu/bash/bash-4.3.tar.gz
  plugin: autotools
  configflags: ["--infodir=/var/bash/info"]
```

Figure 19: snapcraft.yaml 4.0

```
ilhanschool@ubuntu:~/mysnaps/hello/snap$ sudo snap install hello_2.10_amd64.snap --dangerous
hello 2.10 installed
ilhanschool@ubuntu:~/mysnaps/hello/snap$ hello
Hello, world!
ilhanschool@ubuntu:~/mysnaps/hello/snap$ hello.bash
bash-4.3$ exit
exit
ilhanschool@ubuntu:~/mysnaps/hello/snap$ snap info hello
name:      hello
summary:   hello world
publisher: -
license:   unset
description: |
  says hello world
commands:
  - hello.bash
  - hello
refresh-date: today at 03:34 PDT
channels:
  stable:    2.10    2019-04-17 (38) 98kB -
  candidate: 2.10    2017-05-17 (20) 65kB -
  beta:      2.10.1  2017-05-17 (29) 65kB -
  edge:      2.10.42 2017-05-17 (34) 65kB -
installed:   2.10             (x4)  3MB -
ilhanschool@ubuntu:~/mysnaps/hello/snap$
```

Figure 20: - -dangerous

If you don't add the - -dangerous in the line the it will not install, because the snap isn't signed by the Snap Store yet. With the - -dangerous added at the end of the line it can install without being signed by the Snap Store. Now try the commands "hello" and "hello.bash" as shown in the figure above.

7 Push to the store

For this part you'll need an Ubuntu SSO account to proceed. If you have an
Ubuntu SSO account go to the snapcraft dashboard:
https://dashboard.snapcraft.io/
This is the place to manage all the snaps you've build and published. At the
top of the webpage you'll see an add snap button click it and you'll be able to
choose a name for the snap you've made.

Figure 21: snap name

Click register name.

Before you start

Make sure the snapcraft command knows about you by logging in using the
email address attached to your account (0914619@hr.nl).

```
$ snapcraft login
```

Push your snap to the store

1. Open a terminal window (CTRL+ALT+t)

2. CD to the directory containing your snap file

3. Run the following command, with the specific **version string** and **architecture**:

```
$ snapcraft push hello-crownstonehub-ilhan3_<version>_<arch>.snap
```

Need more help?

For more information, please see:

```
$ snapcraft push --help
```

You can also learn more in the upload documentation.

Back

Figure 22: snap name 2.0

If you don't want to be on a desktop and stay in your comfortable
terminal.The next steps would need to be executed on the terminal:

"snapcraft login"
Enter your ubuntu SSO email and password
"snapcraft register $< somename >$"

Now you'll see the following:

Figure 23: snap publish

You'll be asked if the name you chose is the right one if so answer "y". The next thing to do is to change the name in your snapcraft.yaml to the name you've just accepted. and change the grade from strict to stable like: 16 Rebuild the snap using the command as shown in figure above:

"snapcraft"

Now you can release the snap on the 'candidate' channel for now:

"snapcraft push $< namesnap > \_ < version > \_ < architecture >$.snap --release=candidate "
"name_version_architecture.snap"

Now you can release the snap on the 'candidate' channel for now:

"sudo snap install $< name >$ - -channel=candidate"

In order to make your snap available in the store run this commandline:

"snapcraft release $< namesnap >< revision >< channel >$

like so:

Figure 24: to the store

With the other commands shown in the figure you'll be able to find out of you downloaded the snap.

8 Crosscompile amd64 to armhf

The biggest issue I faced while making a snap for a raspberryPi on an linux computer was the fact that I couldn't find the snap in the store while i was using ubuntu core on the raspberryPi.
One way to make the snap work on the raspberryPi is to actually build it on the raspberry. In order to do this you'd have to install a couple of other additional snaps.

NOTE:
You'd only need 1 raspberryPi to build it and push it to the snap store from here you can download the snap on any other raspberry with ubuntu (core) and not have the additional snaps installed.

8.1 Building on the raspberryPi

The first way I fixed the snap on raspberry issue is to build the snap on a Pi. I had to install a couple of additional snaps to make this work. We'll need to edit files and run snapcraft. Snapcraft can not be installed on ubuntu core! Luckily you can install a traditional ubuntu mode called "classic". So we're going to install classic first. To do so run:

"sudo snap install classic –edge –devmode"

To run classic and install snaps or packages that can't be installed on ubuntu core run:

"sudo classic"

We also need to edit files because of this we'll install nano:

"sudo apt-get install nano"

The last thing to install is snapcraft. (2)
Now that everything is installed we can start making the snap. Make a new directory (7) and make the snapcraft.yaml file like in the tutorial above. When you're done with the yaml file run the command snapcraft(11) and push the snap to the store (18). When the snap is pushed into the store successfully you can download the snap on other amrhf devices without any other snaps. It will look like this (5).

8.2   Convert with linux desktop

I've tried a snap builder by Launchpad at the moment this page is under construction because i don't have this bit fully working as need be