

Bitácora Linux Mint Ulyana 20 / Ulyssa 20.1

Sergio Alvariño salvari@gmail.com

junio-2020

Resumen

Bitácora de mi portatil
Solo para referencia rápida y personal.

Índice general

1	Introducción	7
2	Programas básicos	8
2.1	Linux Mint	8
2.2	Firmware	8
2.3	Control de configuraciones con git	8
2.3.1	Instalación de etckeeper	8
2.3.2	Controlar dotfiles con git	9
2.4	Parámetros de disco duro	10
2.4.1	Ajustar <i>Firefox</i>	10
2.5	Fuentes (tipográficas) adicionales	11
2.6	Firewall	11
2.7	Aplicaciones variadas	12
2.8	Algunos programas de control del sistema	13
2.9	Programas de terminal	13
2.9.1	tmux	13
2.10	Dropbox	14
2.11	Chrome	14
2.12	Varias aplicaciones instaladas de binarios	14
2.12.1	Freeplane	15
2.12.2	Treesheets	15
2.12.3	Telegram Desktop	15

2.12.4	Tor browser	15
2.12.5	Brave browser	15
2.12.6	TiddlyDesktop	16
2.12.7	Joplin	16
2.13	Terminal y shells	16
2.13.1	bash-git-promt	16
2.13.2	zsh	16
2.14	Codecs	19
2.15	Syncthing	19
3	Utilidades	19
3.1	Utilidades variadas	19
3.2	time-tracking	20
3.2.1	Activity Watcher	20
3.2.2	go for it	20
3.3	autokeys	21
4	Internet	21
4.1	Rclone	21
4.1.1	Recetas rclone	21
4.1.2	Referencias	22
4.2	Palemoon	22
4.3	LibreWolf	22
4.3.1	Plugins instalados	22
4.4	Netsurf	23
4.5	Lagrange	23
4.6	Castor	23
4.7	Sengi: Cliente de Mastodon	23
5	Window Managers adicionales	23
5.1	i3wm	23
5.2	qtile	23
5.2.1	Configurar Qtile en <i>virtualenv</i> para arrancar desde el <i>Lightdm</i>	24
5.3	Lanzar Window Managers con Xephyr	25
6	Comunicación con dispositivos Android	26
6.1	scrcpy	26
6.2	Heimdall	26
7	Documentación	26

7.1	Vanilla LaTeX	26
7.1.1	Falsificando paquetes	27
7.1.2	Fuentes	27
7.2	Tipos de letra	28
7.3	Fuentes Adicionales	29
7.4	Pandoc	29
7.5	Algunos editores adicionales	29
7.6	Calibre	29
7.7	Scribus	30
7.7.1	Cambiados algunos valores por defecto	31
7.7.2	Solucionados problemas de <i>hyphenation</i>	31
7.8	Foliate: lector de libros electrónicos	31
7.9	Zotero: Gestor de referencias bibliográficas	31
8	Desarrollo software	32
8.1	Paquetes esenciales	32
8.2	Git	32
8.3	Emacs	33
8.4	Lenguaje de programación D (D programming language)	33
8.4.1	D-apt e instalación de programas	33
8.4.2	DCD	34
8.4.3	gdc	34
8.4.4	ldc	34
8.4.5	Emacs para editar D	34
8.5	C, C++	35
8.5.1	Instalación de Gnu Global	35
8.6	Rust	35
8.6.1	RLS	36
8.6.2	Algunas utilidades de sistema escritas en Rust	36
8.7	golang	36
8.7.1	Instalación de <i>gopls</i> un servidor de LSP para editores:	37
8.7.2	golint	37
8.8	Processing	37
8.9	openFrameworks	37
8.10	Python	38
8.10.1	Paquetes de python instalados	38
8.10.2	Instalación de bpython y ptpython	39
8.10.3	Jupyter	39
8.10.4	Instalamos python3.9	39
8.10.5	pyenv	39
8.10.6	poetry	40

8.11	neovim	40
8.12	Firefox developer edition	43
8.13	Navegadores cli	43
8.14	MariaDB	43
8.15	Squirrel SQL Client	44
8.16	R y R-studio	44
8.16.1	R-studio	44
8.17	Octave	45
9	Desarrollo hardware	45
9.1	Arduino IDE	45
9.1.1	Añadir soporte para <i>Feather M0</i>	47
9.1.2	Añadir soporte para <i>Circuit Playground Express</i>	47
9.1.3	Añadir soporte para STM32	47
9.1.4	Añadir soporte para ESP32 y ESP8266	47
9.1.5	Añadir biblioteca de soporte para Makeblock	48
9.2	Pinguino IDE	49
9.3	stm32 cubeide	49
9.4	esp-idf	49
9.5	KiCAD	50
9.5.1	Actualizar a KiCAD 6.0	51
9.6	Analizador lógico	51
9.6.1	Sigrok	51
9.6.2	Sump logic analyzer	52
9.6.3	OLS	52
9.7	IceStudio	52
9.8	PlatformIO	52
9.8.1	VS Code	52
9.8.2	Incluir platform.io CLI en el PATH	53
9.8.3	vscodium	53
9.8.4	Editor Atom	53
9.9	RepRap	54
9.9.1	OpenScad	54
9.9.2	Slic3r	54
9.9.3	Slic3r Prusa Edition	54
9.9.4	ideaMaker	54
9.9.5	Ultimaker Cura	54
9.9.6	Pronterface	54
9.10	Cortadora de vinilos	55
9.10.1	Inkcut	55
9.10.2	Plugin para inkscape	55

9.11 Drone	55
9.11.1 qgroundcontrol	55
9.11.2 missionplanner	55
9.12 node-red	56
9.12.1 Instalación de node.js	56
9.12.2 Instalación de mosquitto	56
9.12.3 Instalación de Influxdb	57
9.12.4 Instalación de node-red	57
10 Aplicaciones de gráficos	57
10.1 LibreCAD	57
10.2 FreeCAD	58
10.3 Inkscape	58
10.4 Gimp	58
10.4.1 Plugins de Gimp	59
10.5 Krita	59
10.6 MyPaint	59
10.7 Alchemy	59
10.8 Capturas de pantalla	59
10.9 Reoptimizar imágenes	60
10.9.1 ImageMagick	60
10.9.2 Imagine	60
10.10 dia	60
10.11 Blender	60
10.12 Structure Synth	60
10.13 Heron animation	60
10.14 Stopmotion	60
10.15 Instalación del driver digiment para tabletas gráficas Huion	61
11 Sonido	61
11.1 Spotify	61
11.2 Audacity	61
11.3 Clementine	62
12 Video	62
12.1 Shotcut	62
12.2 kdenlive	62
12.3 Openshot	62
12.4 Avidemux	62
12.5 Handbrake	63
12.6 Grabación de screencast	63

12.6.1	Vokoscreen, Kazam y SimpleScreenRecorder	63
12.6.2	OBS	63
12.7	Grabación de podcast	63
12.7.1	Mumble	63
12.8	Cliente de youtube smtube	63
13	Fotografía	64
13.1	Rawtherapee	64
13.2	Darktable	64
13.3	Digikam	64
13.4	Webcamoid	64
14	Seguridad	64
14.1	Autenticación en servidores por clave pública	64
14.2	Claves gpg	65
14.3	Seahorse	65
14.4	Conexión a github con claves ssh	65
14.4.1	Claves ssh	65
14.5	Signal	66
14.6	Element (cliente de matrix.org)	66
14.7	Lector DNI electrónico	67
15	Virtualizaciones y contenedores	68
15.1	Instalación de <i>virtualBox</i>	68
15.1.1	Instalación desde Ubuntu	69
15.1.2	Instalación desde repos oficiales	69
15.2	qemu	70
15.2.1	Referencias	70
15.3	Docker	71
15.3.1	docker-compose	72
15.3.2	Kitematic	72
16	Utilidades para mapas y cartografía	72
16.1	josm	72
16.2	MOBAC	73
16.2.1	Referencias	73
16.3	QGIS	73
16.3.1	Referencias	73
17	Recetas variadas	74
17.1	Añadir las claves GPG de un repo	74

17.2 Solucionar problemas de menús duplicados usando menulibre .	74
17.3 Formatear memoria usb	74
17.4 Copiar la clave pública ssh en un servidor remoto	75
17.5 ssh access from termux	75
17.6 SDR instalaciones varias	75
17.7 Posible problema con modemmanager y micros programables .	75
17.8 Programar los nanos con chip ch340 o ch341	76
17.9 Linux Mint 20 es <i>python agnostic</i>	76

1 Introducción

Mis portatiles son:

- Un ordenador Acer 5755G con las siguientes características:
 - Core i5 2430M 2.4GHz
 - NVIDIA Geforce GT 540M (+ intel integrada)
 - 8Gb RAM
 - 750Gb HD

Este portátil equipa una tarjeta *Nvidia Geforce GT540M* que resulta pertenecer a una rama muerta en el árbol de desarrollo de Nvidia.

Esta tarjeta provocaba todo tipo de problemas de sobrecalentamiento, pero en las últimas versiones de Linux instalando el driver de Nvidia parece funcionar correctamente.

- Un Lenovo Legion
 - Core i7-9750H
 - Nvidia GTX1650-4Gb (+ intel integrada)
 - 16Gb RAM
 - 512Gb SSD + 1Tb HDD

2 Programas básicos

2.1 Linux Mint

Linux Mint incluye `sudo` y las aplicaciones que uso habitualmente para gestión de paquetes por defecto (*aptitude* y *synaptic*).

Interesa tener instalado el paquete *ppa-purge* (`sudo apt install ppa-purge`). Sirve para eliminar ppas junto con los programas instalados desde ese ppa.

Tampoco voy a enredar nada con los orígenes del software (de momento), es decir no voy a cambiar al depósito regional.

2.2 Firmware

Ya no es necesario instalar los paquetes de *microcode* la instalación de Ulyana se encargó de instalar:

- `amd64-microcode`
- `intel-microcode`

Instalamos el driver de nvidia recomendado, el *Mint* nos avisa de que tenemos que revisar la instalación de los drivers.

El driver de Nvidia viene muy mejorado. Merece la pena ver todas las opciones.

Una vez instalado el driver de nvidia, el comando `prime-select query` debe indicarnos la tarjeta activa y podremos cambiar de tarjeta ejecutando `prime-select [nvidia|intel]`

2.3 Control de configuraciones con git

Una vez instalado el driver de nvidia y antes de seguir con la instalación instalamos el `git` y el `etckeeper` para que todos los cambios que se produzcan en el directorio `/etc` durante nuestra instalación queden reflejados en el `git`.

Yo nunca almaceno esta información en la nube, pero me permite tener controlados los cambios de configuración y ayuda en caso de problemas.

2.3.1 Instalación de etckeeper

¡Ojo!, nos hacemos `root` para ejecutar:


```
sudo su -  
git config --global user.email xxxxx@whatever.com  
git config --global user.name "Name Surname"  
apt install etckeeper
```

etckeeper hara un control automático de tus ficheros de configuración en */etc*

Para echar una mirada a los *commits* creados puedes ejecutar:

```
cd /etc  
sudo git log
```

2.3.2 Controlar dotfiles con git

Vamos a crear un repo de git para controlar nuestros ficheros personales de configuración.

Creamos el repo donde queramos

```
mkdir usrcfg  
cd usrcfg  
git init  
git config core.worktree "/home/salvari"
```

Y ya lo tenemos, un repo que tiene el directorio de trabajo apuntando a nuestro *\$HOME*.

Podemos añadir los ficheros de configuración que queramos al repo:

```
git add .bashrc  
git commit -m "Add some dotfiles"
```

Una vez que tenga añadidos los ficheros que quiero tener controlados pondré *** en el fichero *.git/info/exclude* de mi repo para que ignore todos los ficheros de mi *\$HOME*.

Cuando instalo algún programa nuevo añadido a mano los ficheros de configuración al repo.

Yo no tengo información confidencial en este repositorio (claves ssh por ejemplo) así que no tengo problemas en almacenarlo en la nube. Facilita mucho las cosas en casos de upgrade del sistema o copiar configuraciones entre ordenadores.

2.4 Parámetros de disco duro

Tengo un disco duro `ssd` y otro `hdd` normal.

El area de intercambio la hemos creado en el disco duro `hdd`, no se usará mucho (mejor dicho: no se usará nunca) pero evitamos multiples operaciones de escritura en el disco `ssd` en caso de que se empiece a tirar del `swap`.

Añadimos el parámetro `noatime` para las particiones de `root` y `/home`, que si que se han creado en el `ssd`.

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point> <type> <options> <dump> <pass>
# / was on /dev/sda5 during installation
UUID=d96a5501-75b9-4a25-8ecb-c84cd4a3fff5 / ext4 noatime,errors=remount-
ro 0 1
# /home was on /dev/sda7 during installation
UUID=8fcde9c5-d694-4417-adc0-8dc229299f4c /home ext4 defaults,noatime 0
# /store was on /dev/sdc7 during installation
UUID=0f0892e0-9183-48bd-aab4-9014dc1bd03a /store ext4 defaults 0 2
# swap was on /dev/sda6 during installation
UUID=ce11ccb0-a67d-4e8b-9456-f49a52974160 none swap sw 0 0
# swap was on /dev/sdc5 during installation
UUID=11090d84-ce98-40e2-b7be-dce3f841d7b4 none swap sw 0 0
```

Una vez modificado el `/etc/fstab` no hace falta arrancar, basta con ejecutar lo siguiente:

```
mount -o remount /
mount -o remount /home
mount
```

2.4.1 Ajustar *Firefox*

Las diferencias de rendimiento del `Firefox` con estos ajustes son bastante notables.

Seguimos [esta referencia](#)

Visitamos `about:config` con el navegador.

Cambiamos

- `browser.cache.disk.enable` **false**
- `browser.cache.memory.enable` **true**
- `browser.cache.memory.capacity` **204800**
- `browser.sessionstore.interval` **15000000**

TODO: Comprobar *trim* en mi disco duro. Y mirar [esto](#)

Y siguiendo [esta otra referencia](#) cambiamos:

- `gfx.x11-egl.force-enabled` **true**

Reiniciamos nuestro *Firefox* (podemos visitar `about:restartRequired`)

2.5 Fuentes (tipográficas) adicionales

Instalamos algunas fuentes desde los orígenes de software:

```
sudo apt install ttf-mscorefonts-installer
sudo apt install fonts-noto
```

Y la fuente [Mensch](#) la bajamos directamente al directorio `~/.local/share/fonts`

Puede ser conveniente instalar el paquete *font-manager* (`sudo apt install font-manager`), sólo lo uso para visualizar fuentes, no para hacer configuración.

Instaladas varias de las [nerd-fonts](#) en `~/.local/share/fonts`. Simplemente descargamos las fuentes interesantes y ejecutamos `sudo fc-cache -f -v`

Fuentes interesantes:

- [nerd-fonts](#)
- [Powerline fonts](#)
- [Programming fonts](#)

2.6 Firewall

`ufw` y `gufw` vienen instalados por defecto, pero no activados.

```
aptitude install ufw
ufw default deny
ufw enable
ufw status verbose
aptitude install gufw
```

Nota: Ojo con el log de `ufw`, tampoco le sienta muy bien al `ssd` esa escritura masiva. Yo normalmente lo dejo desactivado excepto cuando valido una nueva configuración.

2.7 Aplicaciones variadas

Nota: Ya no instalamos *menulibre*, Linux Mint tiene una utilidad de edición de menús.

Keepass2 Para mantener nuestras contraseñas a buen recaudo

Gnucash Programa de contabilidad, la versión de los repos está bastante atrasada habrá que probar la nueva que puede instalarse desde la web o desde el flathub.

Deluge Programa de descarga de torrents (acuérdate de configurar tus cortafuegos)

rsync, grsync Para hacer backups de nuestros ficheros

Descompresores variados Para lidiar con los distintos formatos de ficheros comprimidos

mc Midnight Comander, gestor de ficheros en modo texto

most Un `less` mejorado

tree Para ver estructuras de directorios

neofetch Este solo vale para presumir de ordenador creo ٩_٩

fasd Para cambiar entre directorios rápidamente

silversearcher-ag Una alternativa mejorada al clásico *grep*

ack Otro `grep` mejorado

mate-tweak Para *customizar* nuestro escritorio Mate

filezilla Un interfaz gráfico para transferencia de ficheros

rofi Un conmutador de ventanas capaz de automatizar muchas tareas

```
sudo apt install keepass2 gnucash deluge rsync grsync rar unrar \  
zip unzip unace bzip2 lzop p7zip p7zip-full p7zip-rar \  

```

```
most mc tree neofetch fasd silversearcher-ag ack mate-tweak filezilla \  
rofi
```

Chromium Como Chrome pero libre, ahora en Ubuntu solo está disponible como snap así que tendremos que dar un rodeo.

```
sudo add-apt-repository ppa:xalt7x/chromium-deb-vaapi  
cat <<EOF | sudo tee /etc/apt/preferences.d/pin-xalt7x-chromium-deb-  
vaapi  
Package: *  
Pin: release o=LP-PPA-xalt7x-chromium-deb-vaapi  
Pin-Priority: 1337  
EOF
```

```
sudo apt update  
sudo apt install chromium-browser chromium-codecs-ffmpeg chromium-  
codecs-ffmpeg-extra
```

2.8 Algunos programas de control del sistema

Son útiles para control de consumo.

```
sudo apt install tlp tlp-rdw htop powertop
```

2.9 Programas de terminal

Dos imprescindibles:

```
sudo apt install guake terminator
```

terminator lo dejamos como aplicación terminal preferida del sistema.

TODO: asociar *Guake* a una combinación apropiada de teclas.

También instalo *rxvt* para tener una alternativa ligera al *terminator*.

```
sudo apt install rxvt-unicode
```

2.9.1 tmux

tmux combinado por ejemplo con *rxvt* nos da la misma funcionalidad que *Terminator*, además merece la pena aprender a usarlo por que instalado en servidores remotos es increíblemente útil.

```
sudo apt install tmux
```

- [El tao de tmux](#)
- [rxvt customizations](#)

2.10 Dropbox

Lo instalamos desde el software manager.

2.11 Chrome

No lo he instalado.

Puede instalarse desde [la página web de Chrome](#)

2.12 Varias aplicaciones instaladas de binarios

Lo recomendable en un sistema POSIX es instalar los programas adicionales en `/usr/local` o en `/opt`. Yo soy más chapuzas y suelo instalar en `~/apt` por que el portátil es personal e intrasferible. En un ordenador compartido es mejor usar `/opt`.

En general cuando instalo en el directorio `~/apps` sigo los siguientes pasos:

1. Descargamos los binarios o *appimage* desde la web
2. Descomprimo en un nuevo directorio para la aplicación, tomamos como ejemplo *freeplane*, así que el directorio se llamará: `~/apps/mi_aplicacion`
3. Creamos enlace simbólico al que llamamos *current*. Esto es para no editar los ficheros `.desktop` cada vez que actualicemos la versión del programa. El enlace puede apuntar a un directorio o a un binario, depende de como obtengamos la aplicación. En el caso de *freeplane* yo tengo la siguiente estructura

```
freeplane
├─ current -> freeplane-1.7.11
├─ freeplane-1.7.10
└─ freeplane-1.7.11
```

Vemos que el enlace apunta a la versión más reciente de *freeplane*.

4. Añadimos la aplicación a los menús, al hacer esto se creará un fichero `.desktop` en el directorio `~/.local/share/applications`

2.12.1 Freeplane

Para hacer mapas mentales, presentaciones, resúmenes, apuntes... La versión incluida en LinuxMint está un poco anticuada.

2.12.2 Treesheets

Está bien para hacer chuletas rápidamente. Descargamos el *appimage* desde [la web](#)

2.12.3 Telegram Desktop

Cliente de Telegram, descargado desde la [página web](#). El programa de instalación de Telegram ya se encarga de crear el fichero `.desktop`

2.12.4 Tor browser

Descargamos desde la [página oficial del proyecto](#) Descomprimos en `~/apps/` y ejecutamos desde terminal:

```
cd ~/apps/tor-browser
./start-tor-browser.desktop --register-app
```

Tor se encarga tanto de crear el fichero `.desktop` como de mantenerse actualizado a la última versión.

2.12.5 Brave browser

Instalamos siguiendo las instrucciones de la [página web oficial](#)

```
curl -s https://brave-browser-apt-release.s3.brave.com/brave-core.asc | sudo apt-key --keyring /etc/apt/trusted.gpg.d/brave-browser-release.gpg add -
```

```
echo "deb [arch=amd64] https://brave-browser-apt-release.s3.brave.com/ stable main" | sudo tee /etc/apt/sources.list.d/brave-browser-release.list
```

```
sudo apt update
```

```
sudo apt install brave-browser
```

2.12.6 TiddlyDesktop

Descargamos desde la [página web](#), descomprimos y generamos la entrada en el menú.

2.12.7 Joplin

Una herramienta libre para mantener notas sincronizadas entre el móvil y el portátil.

Instalamos siguiendo las instrucciones de la [página web](#)

```
wget -O - https://raw.githubusercontent.com/laurent22/joplin/master/Joplin_install_and_configure.sh
```

Joplin se instala en el directorio `~/joplin` y crea su propia entrada en el menú.

2.13 Terminal y shells

Por defecto tenemos instalado bash.

2.13.1 bash-git-prompt

Para dejar configurado el *bash-git-prompt* seguimos las instrucciones de [este github](#)

2.13.2 zsh

Nos adelantamos a los acontecimientos, pero conviene tener instaladas las herramientas de entornos virtuales de python antes de instalar *zsh* con el plugin para *virtualenvwrapper*.

```
apt install python-all-dev
apt install python3-all-dev
apt install virtualenv virtualenvwrapper python3-virtualenv
```

zsh viene por defecto en mi instalación, en caso contrario:

```
apt install zsh
```

Para *zsh* vamos a usar [antigen](#), así que nos lo clonamos en `~/apps/`


```
cd ~/apps
git clone https://github.com/zsh-users/antigen
```

También vamos a usar [zsh-git-prompt](#), así que lo clonamos también:

```
cd ~/apps
git clone https://github.com/olivierverdier/zsh-git-prompt
```

Y editamos el fichero `~/.zshrc` para que contenga:

```
# This line loads .profile, it's experimental
[[ -e ~/.profile ]] && emulate sh -c 'source ~/.profile'

source ~/apps/zsh-git-prompt/zshrc.sh
source ~/apps/antigen/antigen.zsh

# Load the oh-my-zsh's library.
antigen use oh-my-zsh

# Bundles from the default repo (robbyrussell's oh-my-zsh).
antigen bundle git
antigen bundle command-not-found

# must install autojump for this
#antigen bundle autojump

# extracts every kind of compressed file
antigen bundle extract

# jump to dir used frequently
antigen bundle z

#antigen bundle pip

antigen bundle common-aliases

antigen bundle robbyrussell/oh-my-zsh plugins/virtualenvwrapper

antigen bundle zsh-users/zsh-completions

# Syntax highlighting bundle.
antigen bundle zsh-users/zsh-syntax-highlighting
```

```
antigen bundle zsh-users/zsh-history-substring-search ./zsh-history-  
substring-search.zsh
```

```
# Arialdo Martini git needs awesome terminal font  
#antigen bundle arialdomartini/oh-my-git  
#antigen theme arialdomartini/oh-my-git-themes oppa-lana-style
```

```
# autosuggestions  
antigen bundle tarruda/zsh-autosuggestions
```

```
#antigen theme agnoster  
antigen theme gnzh
```

```
# Tell antigen that you're done.  
antigen apply
```

```
# Correct rm alias from common-alias bundle  
unalias rm  
alias rmi='rm -i'
```

Para usar *virtualenvwrapper* hay que decidir en que directorio queremos salvar los entornos virtuales. El obvio sería `~/` *virtualenvs* la alternativa sería `~/` *local/share/virtualenvs*.

El que escojamos lo tenemos que crear y añadirlo a nuestro `~/` *profile* con las líneas:

```
# WORKON_HOME for virtualenvwrapper  
if [ -d "$HOME/.virtualenvs" ] ; then  
    WORKON_HOME="$HOME/.virtualenvs"  
fi
```

Después de seguir estos pasos basta con arrancar el *zsh*

Antigen ya se encarga de descargar todos los plugins que queramos utilizar en *zsh*. Todos el software se descarga en `~/` *antigen*

Para configurar el [zsh-git-prompt](#), que inspiró el *bash-git-prompt*.

He modificado el fichero `zshrc.sh` de *zsh-git-prompt* cambiando la linea 'echo "\$STATUS":

```
#echo "$STATUS"  
if [[ "$__CURRENT_GIT_STATUS" == ": 0 0 0 0 0 0" ]]; then
```

```
    echo ""
else
    echo "$STATUS"
fi
```

También he cambiado el fichero del tema *gnzh* en `~/ .antigen/bundles/robbyrussell/oh-my-zsh/themes/gnzh.zsh-theme` por que me interesa ver la versión python asociada a cada virtualenv.

2.14 Codecs

```
sudo apt-get install mint-meta-codecs
```

2.15 Syncthing

Añadimos el ppa:

```
curl -s https://syncthing.net/release-key.txt | sudo apt-key add -
echo "deb https://apt.syncthing.net/ syncthing stable" | sudo tee /etc/apt/sources.list
sudo apt-get update
sudo apt-get install syncthing
```

3 Utilidades

3.1 Utilidades variadas

gpick con *Agave* y *pdftk* ya no existen, nos pasamos a *gpick* y *poppler-utils*

```
sudo apt install gpick
```

graphviz Una utilidad de generación de gráficos que uso a veces. También es útil para web2py y para 'org-roam

```
sudo apt install graphviz
```

sqlite3 Un motor de bases de datos sencillo que se uso a menudo

```
sudo apt install sqlite3
```

cheat Chuletas de comandos habituales, se instala bajando el ejecutable desde [su github](#)

cheat.sh Echa una mirada a su página web: <http://cheat.sh/>, es casi idéntico al anterior pero disponible desde cualquier ordenador con conexión.

gparted Instalamos *gparted* para poder formatear memorias usb

```
sudo apt install gparted
```

wkhtmltopdf Para pasar páginas web a pdf

```
sudo apt install wkhtmltopdf
```

lsd ls potenciado, instalamos el paquete desde [la página de releases del github del proyecto](#)

bat cat potenciado, instalamos el paquete desde [la página de releases del github del proyecto](#)

nmap ndiff ncat nmap nos permite realizar mapeos de subredes en nuestras redes locales. Por ejemplo para localizar dispositivos enganchados a nuestra red. ndiff nos permite comparar escaneos realizados con nmap y ncat hace todo tipo de cosas (mira en la red)

```
sudo apt install nmap ndiff ncat
```

rofi El super conmutador de ventanas (y muchas más cosas).

Creamos el fichero `~/.config/rofi/config.rasi` con el siguiente contenido

```
configuration {  
  modi: "drun,run,ssh,combi";  
  theme: "fancy";  
  font: "mensch 16";  
  combi-modi: "window,drun,ssh";  
}
```

Asociamos un atajo de teclado al comando: `rofi -show drun`

3.2 time-tracking

3.2.1 Activity Watcher

Instalado desde la web

En realidad no lo uso para nada.

3.2.2 go for it

Este programa no para de escribir en el disco continuamente. He dejado de usarlo por que me sobra con el org-mode de emacs.

Si de todas formas lo quieres instalar, aquí tienes los comandos:

```
sudo add-apt-repository ppa:go-for-it-team/go-for-it-daily && sudo apt-get update
sudo apt-get install go-for-it
```

3.3 autokeys

Instalamos [autokey](#) siguiendo [las instrucciones para instalarlo con paquetes debian](#)

Me he descargado la última estable en el Legion y la beta en el Acer.

Después de descargar los paquetes Debian ejecutamos los siguientes comandos (para la estable):

```
VERSION=0.95.10-0
sudo dpkg --install autokey-common_0.95.10-0_all.deb autokey-gtk_0.95.10-0_all.deb
sudo apt --fix-broken install
```

El único paquete que se instala a mayores en el python del sistema sería `python3-pyinotify`

Si estamos usando un entorno virtual con `pyenv` creo que sería mejor usar la instalación con `pip` ([ver documentación](#))

4 Internet

4.1 Rclone

Instalamos desde la página web(<https://rclone.org/>), descargando el fichero `.deb`.

```
curl https://rclone.org/install.sh | sudo bash
```

4.1.1 Recetas rclone

Copiar directorio local en la nube:

```
rclone copy /localdir hubic:backup -vv
```

Si queremos ver el directorio en la web de Hubic tenemos que copiarlo en *default*:

```
rclone copy /localdir hubic:default/backup -vv
```

Sincronizar una carpeta remota en local:

```
rclone sync hubic:directorio_remoto /home/salvari/directorio_local -vv
```

4.1.2 Referencias

- [Como usar rclone \(blogdelazaro\)](#)
- [y con cifrado \(blogdelazaro\)](#)
- [Documentación](#)

4.2 Palemoon

Un fork de *Firefox* con menos chorradas. Instalado con el paquete deb descargado de su [página web](#)

4.3 LibreWolf

Otro fork de *Firefox* centrado en la privacidad. Instalado como *appimage* descargado desde su [página web](#)

UPDATE: Ya está disponible el repo para Mint:

```
echo "deb [arch=amd64] http://deb.librewolf.net $(lsb_release -sc) main" | sudo tee /  
sudo wget http://deb.librewolf.net/keyring.gpg -O /etc/apt/trusted.gpg.d/librewolf.gp  
sudo apt update  
sudo apt install librewolf -y
```

4.3.1 Plugins instalados

Conviene estudiar la documentación de los *add-ons* recomendados, disponible [aquí](#).

- KeepassXC-Browser
 - Necesitamos instalar el KeepassXC, el que viene en los repos es un poco antiguo podemos instalar desde PPA
 - Es imprescindible hacer un link con `ln -s ~/.mozilla/native-messaging-hosts ~/.librewolf/native-messaging-hosts`
- Clear URLs
- Mozilla Multiaccount Containers

4.4 Netsurf

Un navegador ultraligero (aunque no funciona con muchas páginas, solo para webs austeras) Instalado via flathub con `flatpak install netsurf`

4.5 Lagrange

Un navegador para el protocolo *Gemini*. Instalado con la *appimage* desde su [página web](#)

4.6 Castor

Otro navegador para el protocolo *Gemini* programado en *Rust*. Instalado desde las fuentes siguiendo instrucciones de su [página web](#)

4.7 Sengi: Cliente de Mastodon

Instalada *appimage* desde su [github](#)

5 Window Managers adicionales

5.1 i3wm

Añadimos el repo:

```
cd ~/tmp
/usr/lib/apt/apt-helper download-file https://debian.sur5r.net/i3/pool/main/s/sur5r-k
sudo dpkg -i ./keyring.deb
sudo echo "deb http://debian.sur5r.net/i3/ $(grep '^DISTRIB_CODENAME=' /etc/lsb-relea
sudo apt update
sudo apt install i3
```

5.2 qtile

Vamos a describir la instalación del *Qtile* en un *virtualenv* dedicado. Si te bajas la versión estable de *Qtile* desde su [página web](#) en el paquete vienen un montón de script auxiliares que te permiten hacer la instalación aislada en un *virtualenv* pero lo voy a hacer a mano para tenerlo controlado con *pyenv*.

- Creamos el *virtualenv* qtile basado en la versión 3.10.0 que teníamos instalada previamente:

```
# Creamos el directorio de trabajo
mkdir <workPath>/qtile
cd <workPath>/qtile

# OPCIONAL: Descargamos el qtile estable de la página web
# sólo si quieres curiosear los scripts auxiliares
wget https://github.com/qtile/qtile/archive/v0.18.1.tar.gz
tar xvzf v0.18.1.tar.gz
rm v0.18.1.tar.gz

# Creamos el entorno virtual
pyenv virtualenv 3.10.0 qtile
pyenv local qtile
# Instalamos los paquetes iniciales (comunes a mis entornos)
pip install --upgrade pip setuptools wheel

# instalamos los requisitos
pip install --no-cache-dir xcffib
pip install --no-cache-dir cairocffi

# Instalamos la versión estable de qtile
pip install --no-cache-dir qtile
```

Con esto ya estamos listos, podríamos arrancar *Qtile* con `qtile start`, pero no puede funcionar claro. Para que arranque correctamente, tenemos que lanzarlo en un servidor X. (ver el punto “[Lanzar Window Managers con Xephyr](#)”)

5.2.1 Configurar Qtile en *virtualenv* para arrancar desde el *Lightdm*

Como root nos creamos un script `launch_qtile` en `/usr/local/bin`, con el siguiente contenido

```
#!/bin/bash
source '/home/user/.pyenv/versions/3.10.0/envs/qtile/bin/activate'
qtile start
```

Le damos permisos de ejecución con `chmod 755 launch_qtile` (ojo a los permisos para *all* que si no son estos no le gusta a *Lightdm*)

También como root creamos el fichero `/usr/share/xsessions/qtile.desktop`

con el contenido:

```
[Desktop Entry]
Name=Qtile
Comment=Qtile Session
Exec=launch_qtile
Type=Application
Keywords=wm;tiling
```

Y con esto tendremos Qtile disponible en *Lightdm*.

5.3 Lanzar Window Managers con Xephyr

Para probar (o configurar) los *Window Managers* sin salir de nuestra sesión de Mate podemos usar Xephyr, si no lo tienes instalado ejecuta:

```
sudo apt update
sudo apt install xserver-xephyr
```

Para lanzar un *Xserver* usaríamos un comando como:

```
Xephyr -ac -screen 800x600 -br -reset -terminate 2> /dev/null :1 &
```

-ac Autorizar conexiones de clientes indiscriminadamente (*disable access restrictions*)

-screen Especificar la geometría de la pantalla.

-br La ventana raíz tendrá fondo negro

-reset Reset al terminar el último cliente

-terminate Finalizar cuando se resetee el servidor

2> /dev/null Mandar los mensajes de error al limbo (alias **NE** en nuestro pc)

:1 Arrancar el server en el DISPLAY=1

Así que si queremos arrancar por ejemplo el *i3wm* podríamos hacer un script con las siguientes líneas:

```
Xephyr -ac -screen 800x600 -br -reset -terminate 2> /dev/null :1 &
export DISPLAY=:1
i3
```

Para *Qtile* bastaría con cambiar *i3* por *qtile start*

6 Comunicación con dispositivos Android

6.1 scrcpy

```
sudo apt install scrcpy
```

6.2 Heimdall

Para flashear roms en moviles

```
sudo apt install heimdall-flash heimdall-flash-frontend
```

7 Documentación

7.1 Vanilla LaTeX

Para instalar la versión más reciente de LaTeX hago la instalación desde [ctan](#)

Una vez instalado usamos *equivs* para generar un paquete deb y que nuestro sistema sepa que tenemos *texlive* instalado.

```
cd ~
mkdir tmp
cd tmp
wget http://mirror.ctan.org/systems/texlive/tlnet/install-tl-unx.tar.gz
tar xzf install-tl-unx.tar.gz
cd install-tl-xxxxxx
```

La parte xxxxxx varía en función del estado de la última versión de LaTeX disponible.

```
sudo ./install-tl
```

Una vez lanzada la instalación podemos desmarcar las opciones que instalan la documentación y las fuentes. Eso nos obligará a consultar la documentación on line pero ahorrará prácticamente el 50% del espacio necesario. En mi caso sin doc ni src ocupa 2,3Gb

```
mkdir -p /opt/texbin
sudo ln -s /usr/local/texlive/2020/bin/x86_64-linux/* /opt/texbin
```

Por último para acabar la instalación añadimos `/opt/texbin` al *PATH*. Para *bash* y *zsh* basta con añadir al fichero `~/.profile` las siguientes líneas:

```
# adds texlive to my PATH
if [ -d "/opt/texbin" ] ; then
    PATH="$PATH:/opt/texbin"
fi
```

En cuanto a *fish* (si es que lo usas, claro) tendremos que modificar (o crear) el fichero `~/.config/fish/config.fish` y añadir la siguiente línea:

```
set PATH $PATH /opt/texbin
```

7.1.1 Falsificando paquetes

Ya tenemos el *texlive* instalado, ahora necesitamos que el gestor de paquetes sepa que ya lo tenemos instalado.

```
sudo apt install equivs --no-install-recommends
mkdir -p /tmp/tl-equivs && cd /tmp/tl-equivs
equivs-control texlive-local
```

Alternativamente para hacerlo más fácil podemos descargarnos un fichero *texlive-local* ya preparado, ejecutando:

```
wget http://www.tug.org/texlive/files/debian-equivs-2018-ex.txt
/bin/cp -f debian-equivs-2020-ex.txt texlive-local
```

Editamos la versión (si queremos) y procedemos a generar el paquete *deb*.

```
equivs-build texlive-local
```

El paquete que hemos generado tiene una dependencia: *freeglut3*, hay que instalarla previamente.

```
sudo apt install freeglut3
sudo dpkg -i texlive-local_2020-1_all.deb
```

Todo listo, ahora podemos instalar cualquier paquete *debian* que dependa de *texlive* sin problemas de dependencias, aunque no hayamos instalado el *texlive* de *Debian*.

7.1.2 Fuentes

Para dejar disponibles las fuentes *opentype* y *truetype* que vienen con *texlive* para el resto de aplicaciones:

```
sudo cp $(kpsewhich -var-value TEXMFSYSVAR)/fonts/conf/texlive-
fontconfig.conf /etc/fonts/conf.d/09-texlive.conf
```

```
sudo nano /etc/fonts/conf.d/09-texlive.conf
```

Borramos la línea:

```
<dir>/usr/local/texlive/20xx/texmf-dist/fonts/type1</dir>
```

Y ejecutamos:

```
sudo fc-cache -fsv
```

Actualizaciones Para actualizar nuestro *latex* a la última versión de todos los paquetes:

```
sudo /opt/texbin/tlmgr update --self
```

```
sudo /opt/texbin/tlmgr update --all
```

También podemos lanzar el instalador gráfico con:

```
sudo /opt/texbin/tlmgr --gui
```

Para usar el instalador gráfico hay que instalar previamente:

```
sudo apt-get install perl-tk --no-install-recommends
```

Lanzador para el actualizador de *texlive*:

```
mkdir -p ~/.local/share/applications
```

```
/bin/rm ~/.local/share/applications/tlmgr.desktop
```

```
cat > ~/.local/share/applications/tlmgr.desktop << EOF
```

```
[Desktop Entry]
```

```
Version=1.0
```

```
Name=TeX Live Manager
```

```
Comment=Manage TeX Live packages
```

```
GenericName=Package Manager
```

```
Exec=gksu -d -S -D "TeX Live Manager" '/opt/texbin/tlmgr -gui'
```

```
Terminal=false
```

```
Type=Application
```

```
Icon=system-software-update
```

```
EOF
```

7.2 Tipos de letra

Creamos el directorio de usuario para tipos de letra:

```
mkdir ~/.local/share/fonts
```

7.3 Fuentes Adicionales

Me he descargado de internet la fuente [Mensch](#) el directorio de usuario para los tipos de letra: `~/ .local/share/fonts`

Además he clonado el repo [Programming Fonts](#) aunque parece que las fuentes están un poco anticuadas.

```
cd ~/wherever
git clone https://github.com/ProgrammingFonts/ProgrammingFonts
cd ~/ .local/share/fonts
ln -s ~/wherever/ProgrammingFonts/Menlo .
```

La fuente Hack la he instalado directamente desde el [sitio web](#)

7.4 Pandoc

Pandoc es un traductor entre formatos de documento. Está escrito en Python y es increíblemente útil. De hecho este documento está escrito con *Pandoc*.

Instalado el *Pandoc* descargando paquete `.deb` desde [la página web del proyecto](#).

Además descargamos plantillas de Pandoc desde [este repo](#) ejecutando los siguientes comandos:

```
mkdir ~/ .pandoc
cd ~/ .pandoc
git clone https://github.com/jgm/pandoc-templates templates
```

Las plantillas no son imprescindibles pero si quieres aprender a usarlas o hacer alguna modificación viene bien tenerlas.

7.5 Algunos editores adicionales

Dos editores para hacer pruebas:

Obsidian Instalado con *appimage* descargado desde [la página web](#)

Zettlr Instalado con fichero `.deb` descargado desde [su página web](#)

7.6 Calibre

La mejor utilidad para gestionar tu colección de libros electrónicos.

Ejecutamos lo que manda la página web:

```
sudo -v && wget -nv -O- https://download.calibre-ebook.com/linux-installer.sh | sudo sh /dev/stdin
```

El programa queda instalado en `/opt/calibre`. Se puede desinstalar con el comando `sudo calibre-uninstall`.

Para usar el calibre con el Kobo Glo:

- Desactivamos todos los plugin de Kobo menos el Kobo Touch Extended
- Creamos una columna MyShelves con identificativo `#myshelves`
- En las opciones del plugin:
 - En la opción `Collection columns` añadimos las columnas `series,#myshelves`
 - Marcamos las opciones `Create collections` y `Delete empty collections`
 - Marcamos *Modify CSS*
 - `Update metadata on device` y `Set series information`

Algunos enlaces útiles:

- (<https://github.com/jgoguen/calibre-kobo-driver>)
- (<http://www.lectoreselectronicos.com/foro/showthread.php?15116-Manual-de-instalaci%C3%B3n-y-uso-del-plugin-Kobo-Touch-Extended-para-Calibre>)
- (<http://www.redelijkheid.com/blog/2013/7/25/kobo-glo-ebook-library-management-with-calibre>)
- (<https://www.netogram.com/kobo.htm>)

7.7 Scribus

Scribus es un programa libre de composición de documentos. con Scribus puedes elaborar desde los folletos de una exposición hasta una revista o un poster.

Instalamos desde los depósitos oficiales de Mint.

Se podría instalar desde ppa cuando lo actualicen para incluir Ubuntu 20 con los siguientes comandos:

```
sudo add-apt-repository ppa:scribus/ppa
sudo apt update
sudo apt install scribus scribus-ng scribus-template scribus-ng-doc
```

7.7.1 Cambiados algunos valores por defecto

He cambiado los siguientes valores en las dos versiones, no están exactamente en el mismo menú pero no son difíciles de encontrar:

- Lenguaje por defecto: **English**
- Tamaño de documento: **A4**
- Unidades por defecto: **milimeters**
- Show Page Grid: **Activado**
- Dimensiones de la rejilla:
 - Mayor: **30 mm**
 - Menor: **6mm**
- En opciones de salida de *pdf* indicamos que queremos salida a impresora y no a pantalla. Y también que no queremos *spot colors*, que serían sólo para ciertas impresoras industriales, así que activamos la opción *Convert Spot Colors to Process Colors*.

Siempre se puede volver a los valores por defecto sin mucho problema (hay una opción para ello)

Referencia [aquí](#)

7.7.2 Solucionados problemas de *hyphenation*

Scribus no hacía correctamente la separación silábica en castellano, he instalado los paquetes:

- hyphen-es
- hyphen-gl

Y ahora funciona correctamente.

7.8 Foliote: lector de libros electrónicos

Instalado el paquete deb desde [su propio github](#)

7.9 Zotero: Gestor de referencias bibliográficas

Por cortesía de [Emiliano Heyns](#) tenemos disponible el paquete de Zotero para Debian y Ubuntu.

```
wget -q0- https://apt.retorque.re/file/zotero-apt/install.sh | sudo bash  
sudo apt update
```

```
sudo apt install zotero
```

8 Desarrollo software

8.1 Paquetes esenciales

Estos son los paquetes esenciales para empezar a desarrollar software en Linux.

```
sudo apt install build-essential checkinstall make automake cmake autoconf \
git git-core git-crypt dpkg wget
```

8.2 Git

NOTA: Si quieres instalar la última versión de git, los git developers tienen un ppa para ubuntu, si quieres tener el git más actualizado:

```
sudo add-apt-repository ppa:git-core/ppa
sudo apt update
sudo apt upgrade
```

Control de versiones distribuido. Imprescindible. Para *Linux Mint* viene instalado por defecto.

Configuración básica de git:

```
git config --global ui.color auto
git config --global user.name "Pepito Pérez"
git config --global user.email "pperez@mikasa.com"

git config --global alias.cl clone

git config --global alias.st "status -sb"
git config --global alias.last "log -1 --stat"
git config --global alias.lg "log --graph --pretty=format:'%Cred%h%Creset -
%C(yellow)%d%Creset %s %Cgreen(%cr) %Cblue<an>%Creset' --abbrev-
commit --date=relative --all"
git config --global alias.dc "diff --cached"
```



```
git config --global alias.unstage "reset HEAD --"
```

```
git config --global alias.ci commit  
git config --global alias.ca "commit -a"
```

```
git config --global alias.ri "rebase -i"  
git config --global alias.ria "rebase -i --autosquash"  
git config --global alias.fix "commit --fixup"  
git config --global alias.squ "commit --squash"
```

```
git config --global alias.cp cherry-pick  
git config --global alias.co checkout  
git config --global alias.br branch  
git config --global core.editor emacs
```

8.3 Emacs

Instalado emacs desde los repos:

```
sudo aptitude install emacs
```

8.4 Lenguaje de programación D (D programming language)

El lenguaje de programación D es un lenguaje de programación de sistemas con una sintaxis similar a la de C y con tipado estático. Combina eficiencia, control y potencia de modelado con seguridad y productividad.

8.4.1 D-apt e instalación de programas

Configurado *d-apt*, instalados todos los programas incluidos

```
sudo wget http://master.dl.sourceforge.net/project/d-apt/files/d-  
apt.list -O /etc/apt/sources.list.d/d-apt.list  
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys EBCF975E5BA24D5E  
sudo apt update
```

Instalamos todos los programas asociados excepto *textadept* que no uso.

```
sudo apt install dmd-compiler dmd-tools dub dcd dfix dfmt dscanner
```

8.4.2 DCD

Una vez instalado el DCD tenemos que configurarlo creando el fichero `~/.config/dcd/dcd.conf` con el siguiente contenido:

```
/usr/include/dmd/druntime/import
/usr/include/dmd/phobos
```

Podemos probarlo con:

```
dcd-server &
echo | dcd-client --search toImpl
```

8.4.3 gdc

Instalado con:

```
sudo aptitude install gdc
```

8.4.4 ldc

Instalado con:

```
sudo aptitude install ldc
```

Para poder ejecutar aplicaciones basadas en VibeD, necesitamos instalar:

```
sudo apt-get install -y libssl-dev libevent-dev
```

8.4.5 Emacs para editar D

Instalados los siguientes paquetes desde Melpa

- d-mode
- flymake-d
- flycheck
- flycheck-dmd-dub
- flycheck-d-unitest
- auto-complete (desde melpa)
- ac-dcd

Referencias * (<https://github.com/atilaneves/ac-dcd>) * (<https://github.com/Hackerpilot/DCD>)

8.5 C, C++

8.5.1 Instalación de Gnu Global

Para instalar las dependencias, previamente instalamos:

```
sudo apt install ncurses-dev id-utils exuberant-ctags python-pygments
```

Con `ctags --version` nos aseguramos de que se llama a Exuberant y no el `ctags` que instala Emacs. Si no es así habrá que revisar la definición del `PATH`

`python-pygments` no es necesario para C o C++, pero añade funcionalidad a Global (hasta 25 lenguajes de programación más)

No podemos instalar Global desde los repos de Ubuntu, está muy anticuado y genera bases de datos enormes y lentas. Tendremos que compilarlo.

Nos bajamos las fuentes del programa desde [la página oficial](#) En el momento de escribir esto se trata de la versión 6.6.4.

Descomprimos los fuentes y los compilamos con:

```
./configure --prefix=/usr/local --with-exuberant-ctags=/usr/bin/ctags  
make  
sudo make install
```

He comprobado que `make uninstall` funciona correctamente, las librerías quedan instaladas en `/usr/local/lib/gtags` y los ejecutables en `/usr/local/bin`

8.6 Rust

Instalamos siguiendo las instrucciones de [aquí](#) (Hacemos la instalación por defecto)

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

```
stable installed - rustc 1.47.0 (18bf6b4f0 2020-10-07)
```

Rust is installed now. Great!

To get started you need Cargo's bin directory (`$HOME/.cargo/bin`) in your `PATH` environment variable. Next time you log in this will be done automatically.

To configure your current shell run `source $HOME/.cargo/env`

Yo comento la linea del fichero `.zprofile` por que mi `.zshrc` ya lee el fichero `.profile`

Para desinstalar rust bastaría con ejecutar `rustup self uninstall`

Para actualizar la instalación: `rustup update`

8.6.1 RLS

Seguimos las instrucciones de la página oficial de [LSP](#) y de [RLS](#)

```
rustup update
```

```
rustup component add rls rust-analysis rust-src
```

8.6.2 Algunas utilidades de sistema escritas en Rust

Instalamos:

```
cargo install tealdeer # tldr tldr
cargo install du-dust  # tldr dust
cargo install fd-find  # tldr fd
cargo install exa      # tldr exa
cargo install tokei    # tldr tokei
cargo install gitui    # gitui --help
cargo install ripgrep  # tldr rg
```

8.7 golang

Bajamos el paquete con la última versión desde [la página oficial](#).

Descomprimos como *root* en `/usr/local/`

```
sudo tar -C /usr/local -xvzf go1.15.7.linux-amd64.tar.gz
```

Añadimos al path con las siguientes lineas en `~/.profile`:

```
#golang
if [ -d "/usr/local/go" ] ; then
    export GOROOT="/usr/local/go"
    PATH="$PATH:$GOROOT/bin"
fi
if [ -d "$HOME/work/development/gocode" ] ; then
    export GOPATH="$HOME/work/development/gocode"
```

```
PATH="$PATH:$GOPATH/bin"  
fi
```

8.7.1 Instalación de *gopls* un servidor de LSP para editores:

Desde un directorio **que no sea el GOPATH**

```
G0111MODULE=on go get golang.org/x/tools/gopls@latest
```

8.7.2 *golint*

```
go get -u golang.org/x/lint/golint
```

8.8 Processing

Bajamos el paquete de la [página web](#), descomprimimos en ~/apps/, en las nuevas versiones incorpora un script de instalación que ya se encarga de crear el fichero *desktop*.

La última versión incorpora varios modos de trabajo, he descargado el modo *Python* para probarlo.

8.9 openFrameworks

Nos bajamos los fuentes para linux 64bits desde [la página web del proyecto](#), y las descomprimos en un directorio para proceder a compilarlas.

No hay más que seguir [las instrucciones de instalación para linux](#).

La instalación no es demasiado intrusiva si tienes Ubuntu 18 o mayor y una versión reciente del gcc.

En la primera pregunta que nos hace es necesario contestar que no. De lo contrario falla la compilación.

Añade los siguientes paquetes a nuestro sistema

```
installing OF dependencies  
OF needs to install the following packages using apt-get:  
curl libjack-jackd2-0 libjack-jackd2-dev freeglut3-dev libasound2-  
dev libxmu-dev libxxf86vm-dev g++ libgl1-mesa-dev libglu1-mesa-  
dev libraw1394-dev libudev-dev libdrm-dev libglew-dev libopenal-  
dev libsndfile-dev libfreeimage-dev libcairo2-dev libfreetype6-
```

```
dev libssl-dev libpulse-dev libusb-1.0-0-dev libgtk-3-dev libopencv-
dev libassimp-dev librtaudio-dev libboost-filesystem-dev libgstreamer1.0-
dev libgstreamer-plugins-base1.0-dev gstreamer1.0-libav gstreamer1.0-
pulseaudio gstreamer1.0-x gstreamer1.0-plugins-bad gstreamer1.0-
alsa gstreamer1.0-plugins-base gstreamer1.0-plugins-good gdb libglfw3-
dev liburiparser-dev libcurl4-openssl-dev libpugixml-dev libgconf-2-
4 libgtk2.0-0 libpoco-dev
Do you want to continue? [Y/n]
```

No te olvides de compilar también el *Project Generator*.

8.10 Python

De partida tenemos instalado dos versiones: *python2* y *python3*

Parece que Linux Mint no viene con ningún python por defecto. Si invocamos el comando `python` el sistema nos indicará que no existe.

Para escoger un python por defecto tenemos dos paquetes que podemos instalar: `python-is-python2` y `python-is-python3`

En principio yo no quería instalar ninguno para averiguar que paquetes no funcionaban, pero la instalación de VirtualBox hizo que se instalara automáticamente el paquete `python-is-python2`.

```
python2 -V
Python 2.7.18rc1
```

```
python3 -V
Python 3.8.2
```

8.10.1 Paquetes de python instalados

Son los que ya comentamos en la sección de instalación de zsh, como ya dijimos conviene que instalemos los paquetes de desarrollo:

```
sudo apt install python2-dev
sudo apt install python-all-dev
sudo apt install python3-dev
sudo apt install python3-all-dev
sudo apt install virtualenv virtualenvwrapper python3-virtualenv
```

Ademas añadimos las siguientes lineas al fichero `~/.profile`:

```
# WORKON_HOME for virtualenvwrapper
if [ -d "$HOME/.virtualenvs" ] ; then
WORKON_HOME="$HOME/.virtualenvs"
fi
```

Aquí tenemos la referencia de comandos de *virtualenvwrapper*.

8.10.2 Instalación de bpython y ppython

bpython instalado desde repos `sudo apt install bpython bpython3`

ppython instalado en un virtualenv para probarlo

8.10.3 Jupyter

Una instalación para pruebas.

```
mkvirtualenv -p /usr/bin/python3 jupyter
python -m pip install jupyter
```

8.10.4 Instalamos python3.9

python3.9 está ya disponible en los repos oficiales. Para dejarla instalada:

```
sudo apt install python3.9 python3.9-dev python3.9-venv
```

8.10.5 pyenv

Instalamos los pre-requisitos:

```
sudo apt-get update
sudo apt-get install --no-install-recommends make build-essential \
libssl-dev zlib1g-dev libbz2-dev libreadline-dev libsqlite3-dev \
wget curl llvm libncurses5-dev xz-utils tk-dev \
libxml2-dev libxmlsec1-dev libffi-dev liblzma-dev
```

Podemos clonar el repo en nuestro home:

```
git clone https://github.com/pyenv/pyenv.git ~/.pyenv
```

Pero también podemos añadir el *bundle* para *Antigen* en el fichero `.zshrc` y dejar que se encargue de todo.

```
antigen bundle mattberther/zsh-pyenv
```

Añadimos al fichero ~/.profile:

```
# pyenv
if [ -d "$HOME/.pyenv" ] ; then
    export PYENV_ROOT="$HOME/.pyenv"
    export PATH="$PYENV_ROOT/bin:$PATH"
fi
```

Y configuramos en ~/.bashrc y en ~/.zshrc (aunque en el último no es necesario tocar nada si usamos el *bundle* para *Antigen*):

```
if command -v pyenv 1>/dev/null 2>&1; then
    eval "$(pyenv init -)"
fi
```

Podemos probar que todo funciona con: `pyenv install -l`

Una vez instalado hay que estudiarse [la referencia de comandos](#)

8.10.6 poetry

Instalamos con:

```
curl -sSL https://raw.githubusercontent.com/python-poetry/poetry/master/get-poetry.py
```

Dejamos la activación solo en el fichero ~/.profile, añadiendo las siguientes líneas:

```
# poetry
if [ -d "$HOME/.poetry" ] ; then
    PATH="$HOME/.poetry/bin:$PATH"
fi
```

Las actualizaciones se realizan con `poetry self update`

8.11 neovim

Vamos a probar *neovim*, ahora mismo la versión de los repos de Ubuntu está actualizada a la penúltima versión (0.4.3). También podemos descargar el *apimage* desde [la página web](#)

Es de esperar que alguna vez vuelvan a tener el *neovim* disponible en los repos de la aplicación:

```
sudo apt-add-repository ppa:neovim-ppa/stable
sudo apt update
```



```
sudo apt install neovim
```

Para instalar los módulos de python creamos un *virtualev* que más tarde añadiremos al fichero *init.vim*.

```
mkvirtualenv -p /usr/bin/python3 neovim3
sudo pip install --upgrade neovim
deactivate
```

Revisar [esto](#)

NOTA: El siguiente paso ya no parece necesario, las alternativas se han actualizado con la instalación del *neovim*.

Para actualizar las alternativas:

```
sudo update-alternatives --install /usr/bin/vi vi /usr/bin/nvim 60
sudo update-alternatives --config vi
sudo update-alternatives --install /usr/bin/vim vim /usr/bin/nvim 60
sudo update-alternatives --config vim
```

8.11.0.1 Install *vim-plug* Ejecutamos:

```
curl -fLo ~/.local/share/nvim/site/autoload/plug.vim --create-dirs \
  https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
```

Configuramos el fichero de configuración de *nvim* (*~/.config/nvim/init.vim*):

```
" Specify a directory for plugins
" - For Neovim: ~/.local/share/nvim/plugged
" - Avoid using standard Vim directory names like 'plugin'
call plug#begin('~/.local/share/nvim/plugged')

if has('nvim')
  Plug 'Shougo/deoplete.nvim', { 'do': ':UpdateRemotePlugins' }
else
  Plug 'Shougo/deoplete.nvim'
  Plug 'roxma/nvim-yarp'
  Plug 'roxma/vim-hug-neovim-rpc'
endif

Plug 'deoplete-plugins/deoplete-jedi'
```

```
" Initialize plugin system
call plug#end()

let g:deoplete#enable_at_startup = 1

" set python enviroments
let g:python_host_prog = '/full/path/to/neovim2/bin/python'
let g:python3_host_prog = '/home/salvari/.virtualenvs/neovim3/bin/python'
```

La primera vez que abramos *nvim* tenemos que instalar los plugin porn comando ejecutando: `:PlugInstall`

Instalación de dein

Nota:

Solo hay que instalar uno de los dos o *dein* o *plug-vim*. Yo uso *plug-vim* así que esto es sólo una referencia.

<https://github.com/Shougo/dein.vim>

```
" Add the dein installation directory into runtimepath
set runtimepath+=~/config/nvim/dein/repos/github.com/Shougo/dein.vim

if dein#load_state('~/config/nvim/dein')
  call dein#begin('~/config/nvim/dein')

  call dein#add('~/config/nvim/dein/repos/github.com/Shougo/dein.vim')
  call dein#add('Shougo/deoplete.nvim')
  call dein#add('Shougo/denite.nvim')
  if !has('nvim')
    call dein#add('roxma/nvim-yarp')
    call dein#add('roxma/vim-hug-neovim-rpc')
  endif

  call dein#end()
  call dein#save_state()
endif

filetype plugin indent on
syntax enable
```

8.12 Firefox developer edition

El rollo de siempre, descargar desde [la página web](#) descomprimir en ~/apps y crear un lanzador.

8.13 Navegadores cli

Herramientas útiles para depuración web

```
sudo apt install httpie links
```

8.14 MariaDB

Instalamos la última estable para Ubuntu Fossa desde los repos oficiales.

Primero añadimos los repos.

Añadimos la clave de firma:

```
sudo apt-key adv --fetch-keys 'https://mariadb.org/mariadb_release_signing_key.asc'
```

Ahora tenemos dos opciones:

Podemos ejecutar:

```
sudo add-apt-repository 'deb [arch=amd64,arm64,ppc64el] http://mariadb.mirror.liteserv
```

O podemos crear un fichero /etc/apt/apt.sources.list.d/MariaDB con el siguiente contenido (yo dejo las fuentes comentadas):

```
# MariaDB 10.5 [Stable] repository list - created UTC
# https://mariadb.org/download-test/
deb [arch=amd64] http://mariadb.mirror.liteserver.nl/repo/10.5/ubuntu focal main
# deb-src http://mariadb.mirror.liteserver.nl/repo/10.5/ubuntu focal main
```

Y ya solo nos queda lo de siempre:

```
sudo apt update
sudo apt upgrade
sudo apt install mariadb-server
```

Podemos comprobar con `systemctl status mariadb`

También podemos hacer login con el usuario root:

```
sudo mariadb -u root
```

Ojo, hay que hacer sudo, el comando `mariadb -u root` no funciona.

Y ahora aseguramos la instalación con:

```
sudo mysql_secure_installation
```

Yo diría que tienes que decir que si a todas las preguntas, excepto quizás al *unix_socket_authentication*.

Por último sólo nos queda decidir si el servicio mariadb debe estar ejecutándose permanentemente o no.

Si queremos pararlo y que no se arranque automáticamente al arrancar el ordenador:

```
sudo systemctl stop mariadb
sudo systemctl disable mariadb
```

8.15 Squirrel SQL Client

Bajamos el zip de estándar desde [la página web de Squirrel](#) (yo prefiero no usar el instalador)

Como de costumbre descomprimos en `~/apps` y creamos una entrada en nuestro menú de aplicaciones.

Nos descargamos también el *java connector* para MariaDB. Desde la página oficial. Nos interesa el fichero `mariadb-java-client-2.6.0.jar`

Configuramos el driver para que sepa donde está el fichero `.jar` y ya estamos listos para trabajar.

8.16 R y R-studio

Primero instalamos la última versión de R en nuestro pc:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E298A3A825C0D65DFD57CBB65
sudo add-apt-repository 'deb https://cloud.r-project.org/bin/linux/ubuntu focal-
cran40/'
sudo apt update
sudo apt install r-base
```

8.16.1 R-studio

Descargamos la última versión disponible de *R-studio* desde [la página web](#)

Instalamos con *gdebi* (basta con clicar sobre el fichero *.deb*)

8.17 Octave

Instalado desde flatpak

```
sudo flatpak install flathub org.octave.Octave
```

9 Desarrollo hardware

9.1 Arduino IDE

Bajamos los paquetes de la página [web](#), descomprimimos en `~/apps/arduino`.

La distribución del IDE incluye ahora un fichero `install.sh` que se encarga de hacer la integración del IDE en los menús de Linux.

Además también incluye un script (`arduino-linux-setup.sh`) para crear las *devrules* y que además desinstala el driver *modemmanager* y crea grupos nuevos en el sistema si no existen.

No tengo claro lo de desinstalar el driver así que creamos las *devrules* a mano mirando por el fichero.

Hay que añadir nuestro usuario a los grupos *tty*, *dialout*, *uucp* y *plugdev* (no hay que crear grupos nuevos, ya tenemos todos en el sistema)

```
sudo gpasswd --add <username> tty
sudo gpasswd --add <username> dialout
sudo gpasswd --add <username> uucp
sudo gpasswd --add <username> plugdev
```

Creamos los siguientes ficheros en el directorio `/etc/udev/rules.d`

Fichero `90-extraacl.rules` mete mi usuario en el fichero de reglas (↵↵)

```
# Setting serial port rules
```

```
KERNEL=="ttyUSB[0-9]*", TAG+="udev-acl", TAG+="uaccess", OWNER="salvari"
KERNEL=="ttyACM[0-9]*", TAG+="udev-acl", TAG+="uaccess", OWNER="salvari"
```

Fichero `98-openocd.rules`

```
# Adding Arduino M0/M0 Pro, Primo UDEV Rules for CMSIS-DAP port
```

```
ACTION!="add|change", GOTO="openocd_rules_end"
SUBSYSTEM!="usb|tty|hidraw", GOTO="openocd_rules_end"
```

```
#Please keep this list sorted by VID:PID
```

```
#CMSIS-DAP compatible adapters
ATTRS{product}=="*CMSIS-DAP*", MODE="664", GROUP="plugdev"
```

```
LABEL="openocd_rules_end"
```

```
Fichero avrisp.rules
```

```
# Adding AVRisp UDEV rules
```

```
SUBSYSTEM!="usb_device", ACTION!="add", GOTO="avrisp_end"
# Atmel Corp. JTAG ICE mkII
ATTR{idVendor}=="03eb", ATTRS{idProduct}=="2103", MODE="660", GROUP="dialout"
# Atmel Corp. AVRISP mkII
ATTR{idVendor}=="03eb", ATTRS{idProduct}=="2104", MODE="660", GROUP="dialout"
# Atmel Corp. Dragon
ATTR{idVendor}=="03eb", ATTRS{idProduct}=="2107", MODE="660", GROUP="dialout"
```

```
LABEL="avrisp_end"
```

```
Fichero 40-defuse.rules:
```

```
# Adding STM32 bootloader mode UDEV rules
```

```
# Example udev rules (usually placed in /etc/udev/rules.d)
# Makes STM32 DfuSe device writeable for the "plugdev" group
```

```
ACTION=="add", SUBSYSTEM=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="df11", MO
```

```
Fichero 99-arduino-101.rules:
```

```
# Arduino 101 in DFU Mode
```

```
SUBSYSTEM=="tty", ENV{ID_REVISION}=="8087", ENV{ID_MODEL_ID}=="0ab6", MODE="0666", ENV
SUBSYSTEM=="usb", ATTR{idVendor}=="8087", ATTR{idProduct}=="0aba", MODE="0666", ENV{ID
```

Yo añado el fichero 99-arduino.rules que se encarga de inhibir el modemmanager para que no capture al *CircuitPlayground Express*:

```
# for arduino brand, stop ModemManager grabbing port
```

```
ATTRS{idVendor}=="2a03", ENV{ID_MM_DEVICE_IGNORE}="1"  
# for sparkfun brand, stop ModemManager grabbing port  
ATTRS{idVendor}=="1b4f", ENV{ID_MM_DEVICE_IGNORE}="1"
```

9.1.1 Añadir soporte para *Feather M0*

Arrancamos el IDE Arduino y en la opción de *Preferences::Additional Board Managers URLs* añadimos la dirección https://adafruit.github.io/arduino-board-index/package_adafruit_index.json, si tenemos otras URL, simplemente añadimos esta separada por una coma.

Ahora desde el *Board Manager* instalamos:

- Arduino SAMD Boards
- Adafruit SAMD Boards

9.1.2 Añadir soporte para *Circuit Playground Express*

Bastaría con instalar *Arduino SAMD Boards*

9.1.3 Añadir soporte para STM32

Tenemos varias URL posibles para configurar en las preferencias del IDE Arduino:

- http://dan.drown.org/stm32duino/package_STM32duino_index.json (recomendada por Tutoelectro)
- https://github.com/stm32duino/BoardManagerFiles/raw/master/STM32/package_stm_index.json (parece la oficial)

He optado por añadir los dos ficheros json al IDE, la oficial tiene buena pinta pero parece que no soporta st-link. Con la otra podremos usarlo sin problemas.

Instalamos la biblioteca `stm32_cores` que corresponde al origen de software oficial y la biblioteca `STM32F1xx/GD32F1xx`, esta última es la que nos dará soporte explícito para el st-link

Lo probamos con el *Blink* y funciona perfectamente con las opciones de la [Figura 1](#)

9.1.4 Añadir soporte para ESP32 y ESP8266

Añadimos las URL:



Figura 1: Opciones Arduino para STM32 con st-link

- https://dl.espressif.com/dl/package_esp32_index.json
- http://arduino.esp8266.com/stable/package_esp8266com_index.json

Añadimos la librería:

- ESP32 (de espressif)

9.1.5 Añadir biblioteca de soporte para Makeblock

Nota: Pendiente de instalar

Clonamos el [repo oficial en github](#).

Una vez que descarguemos las librerías es necesario copiar el directorio Makeblock-Libraries/makeblock en nuestro directorio de bibliotecas de Arduino. En mi caso ~/Arduino/libraries/.

Una vez instaladas las bibliotecas es necesario reiniciar el IDE Arduino si estaba arrancado. Podemos ver si se ha instalado correctamente simplemente echando un ojo al menú de ejemplos en el IDE, tendríamos que ver los ejemplos de *Makeblock*.

Un detalle importante para programar el Auriga-Me es necesario seleccionar el micro Arduino Mega 2560 en el IDE Arduino.

9.2 Pinguino IDE

Nota: Pendiente de instalar

Tenemos el paquete de instalación disponible en su página [web](#)

Ejecutamos el programa de instalación. El programa descargará los paquetes Debian necesarios para dejar el IDE y los compiladores instalados.

Al acabar la instalación he tenido que crear el directorio `~/Pinguino/v11`, parece que hay algún problema con el programa de instalación y no lo crea automáticamente.

El programa queda correctamente instalado en `/opt` y arranca correctamente, habrá que probarlo con los micros.

9.3 stm32 cubeide

Nos bajamos el instalador genérico. Tendremos que:

- aceptar un montón de acuerdos de licencias
- indicarle un directorio de instalación (en mi caso `'~/apps/st/st/stm32cubeide_1.4.0'`)
- darle la password de root para instalar ficheros de udev, concretamente:
 - `udev/rules.d/49-stlinkv1.rules`
 - `udev/rules.d/49-stlinkv2-1.rules`
 - `udev/rules.d/49-stlinkv2.rules`
 - `udev/rules.d/49-stlinkv3.rules`
 - `udev/rules.d/99-jlink.rules`

9.4 esp-idf

Instalamos las dependencias (cmake ya lo tenemos instalado)

NOTA: No es necesario instalar los paquetes de python que nos especifican en las instrucciones de instalación del *esp-idf*, se instalarán automáticamente en el siguiente paso.

```
sudo apt-get install gperf cmake ninja-build ccache libffi-dev libssl-dev
```

Ahora creamos un directorio para nuestro *tool-chain*:

```
mkdir ~/esp
cd ~/esp
git clone --recursive https://github.com/espressif/esp-idf
```

También es necesario que nuestro usuario pertenezca al grupo dialout, pero eso ya deberíamos tenerlo hecho de antes.

Una vez clonado el repo ejecutamos el script de instalación

```
cd ~/esp/esp-idf
./install.sh
```

Este script nos va a dejar instaladas todas las herramientas necesarias en el directorio `~/expressif`

Nota: para que funcione correctamente en Linux Mint es necesario que el script `tools/idf_tools.py` apunte al `python3` de nuestro sistema. Basta con editar la primera línea *shebang* del script.

Estas son las bibliotecas que deja instaladas:

```
Installing ESP-IDF tools
Installing tools: xtensa-esp32-elf, xtensa-esp32s2-elf, esp32ulp-elf, esp32s2ulp-elf, openocd-esp32
```

Para empezar a trabajar bastará con hacer un *source* del fichero `~/apps/esp/esp-idf/export.sh`:

```
. ~/apps/esp/esp-idf/export.sh
```

9.5 KiCAD

En la [página web del proyecto](#) nos recomiendan el ppa a usar para instalar la última versión estable:

```
sudo add-apt-repository --yes ppa:kicad/kicad-5.1-releases
sudo apt-get update
sudo apt install kicad
```

Paciencia, el paquete `kicad-packages3d` tarda un buen rato en descargarse.

Algunas librerías alternativas:

- [Freetronics](#) una librería que no solo incluye Shield para Arduino sino una completa colección de componentes que nos permitirá hacer proyectos completos. [Freetronics](#) es una especie de BricoGeek australiano, publica tutoriales, vende componentes, y al parecer mantiene una biblioteca para KiCAD. La biblioteca de Freetronics se mantiene en un repo de github. Lo suyo es incorporarla a cada proyecto, por que si la actualizas se pueden romper los proyectos que estes haciendo.
- [eklablog](#) Esta biblioteca de componentes está incluida en el github de KiCAD, así que teoricamente no habría que instalarla en nuestro disco duro.

9.5.1 Actualizar a KiCAD 6.0

Hacemos un `sudo apt purge` de todos los paquetes del antiguo KiCAD. Desactivamos el antiguo repo y ejecutamos:

```
sudo add-apt-repository --yes ppa:kicad/kicad-6.0-releases
sudo apt update
sudo apt install --install-recommends kicad
# If you want demo projects
# sudo apt install kicad-demos
```

No hay demos de momento en el nuevo KiCAD 6.0

9.6 Analizador lógico

Mi analizador es un OpenBench de Seedstudio, [aquí hay mas info](#)

9.6.1 Sigrok

Instalamos **Sigrok**, simplemente desde los repos de Debian:

```
sudo aptitude install sigrok
```

Al instalar **Sigrok** instalamos también **Pulseview**.

Si al conectar el analizador, echamos un ojo al fichero *syslog* vemos que al conectarlo se mapea en un puerto tty.

Si arrancamos **Pulseview** (nuestro usuario tiene que estar incluido en el grupo *dialout*), en la opción *File::Connect to device*, escogemos la opción *Openbench* y le pasamos el puerto. Al pulsar la opción *Scan for devices* reconoce el analizador correctamente como un *Sump Logic Analyzer*.

9.6.2 Sump logic analyzer

Este es el software recomendado para usar con el analizador.

Descargamos el paquete de la [página del proyecto](#), o más concretamente de [esta página](#) y descomprimos en `~/apps`.

Instalamos las dependencias:

```
sudo apt install librxtx-java
```

Editamos el fichero `~/apps/Logic Analyzer/client/run.sh` y lo dejamos así:

```
#!/bin/bash
```

```
# java -jar analyzer.jar $*
```

```
java -cp /usr/share/java/RXTXcomm.jar:analyzer.jar org.sump.analyzer.Loader
```

Y ya funciona.

9.6.3 OLS

Nota: Pendiente de instalar

[Página oficial](#)

9.7 IceStudio

Instalamos dependencias con `sudo apt install xclip`

Bajamos el *AppImage* desde el [github de IceStudio](#) y lo dejamos en `~/apps/icestudio`

9.8 PlatformIO

9.8.1 VS Code

Añadimos el origen de software:

```
curl https://packages.microsoft.com/keys/microsoft.asc | gpg --  
dearmor > packages.microsoft.gpg
```

```
sudo install -o root -g root -m 644 packages.microsoft.gpg /usr/share/keyrings/
```

```
sudo sh -c 'echo "deb [arch=amd64 signed-by=/usr/share/keyrings/packages.microsoft.gpg]"
```

E instalamos

```
sudo apt update
```

```
sudo apt install code # alternativamente code-insiders (es como la versión beta, se puede
```

Ahora

1. lanzamos el editor
2. abrimos el gestor de extensiones
3. buscamos el platformio ide
4. instalamos

Seguimos las instrucciones de [aquí](#)

9.8.2 Incluir platform.io CLI en el PATH

Esto es una malísima idea, **NO LO HAGAS**

Las instrucciones indican que hagamos lo siguiente para usar Platformio desde línea de comandos pero no es conveniente hacerlo.

Modificamos el fichero `~/.profile` añadiendo las siguientes líneas:

```
if [ -d "$HOME/.platformio/penv/bin" ] ; then
    PATH="$PATH:$HOME/.platformio/penv/bin"
fi
```

Si quieres usar Platformio desde línea de comandos, es mejor activar manualmente el entorno virtual con `source ~/.platformio/penv/bin/activate`

9.8.3 vsodium

```
wget -qO - https://gitlab.com/paulcarroty/vscodium-deb-rpm-repo/raw/master/pub.gpg | gpg --dearmor | sudo dd of=/etc/apt/trusted.gpg.d/vscodium.gpg
echo 'deb https://paulcarroty.gitlab.io/vscodium-deb-rpm-repo/debs/ vscodium main' | sudo tee -a /etc/apt/sources.list.d/vscodium.list
sudo apt update && sudo apt install codium
```

9.8.4 Editor Atom

NOTA: Parece que antes recomendaban instalar Atom para disponer del Platformio CLI, ahora en cambio recomiendan VS Code.

```
wget -q0 - https://packagecloud.io/AtomEditor/atom/gpgkey | sudo apt-  
key add -  
sudo sh -c 'echo "deb [arch=amd64] https://packagecloud.io/AtomEditor/atom/any/ any ma  
sudo apt update  
sudo apt install atom
```

9.9 RepRap

9.9.1 OpenScad

El OpenSCAD está disponible en los orígenes de software, así que `sudo apt install openscad`.

9.9.2 Slic3r

Descargamos la estable desde la [página web](#) y como de costumbre descomprimos en `~/apps` y creamos un lanzador con *MenuLibre*

9.9.3 Slic3r Prusa Edition

Una nueva versión del clásico *Slic3r* con muchas mejoras. Descargamos la *appimage* desde la [página web](#) y ya sabéis, descomprimir en `~/apps` y dar permisos de ejecución.

9.9.4 ideaMaker

Una aplicación más para generar gcode con muy buena pinta, tenemos el paquete *deb* disponible en su [página web](#). Instalamos con el gestor de software.

9.9.5 Ultimaker Cura

Descargamos el *AppImage* desde la [página web](#)

9.9.6 Pronterface

Seguimos las instrucciones para Ubuntu Focal:

Instalamos las dependencias:

Clonamos el repo:

```
cd ~/apps
git clone https://github.com/kliment/Printrun.git
cd Printrun
mkvirtualenv -p /usr/bin/python3 printrun
python -m pip install https://extras.wxpython.org/wxPython4/extras/linux/gtk3/ubuntu-
20.04/wxPython-4.1.0-cp38-cp38-linux_x86_64.whl
pip install -r requirements.txt
# sudo apt-get install libdbus-glib-1-dev libdbus-1-dev
```

Y ya lo tenemos todo listo para ejecutar.

9.10 Cortadora de vinilos

9.10.1 Inkcute

Instalado en un entorno virtual:

```
mkvirtualenv -p `which python3` inkcute

sudo apt install libxml2-dev libxslt-dev libcups2-dev

pip install PyQt5

pip install inkcute
```

9.10.2 Plugin para inkscape

Instalamos dependencias:

```
pip install python-usb
```

Instalamos el fichero .deb desde la web <https://github.com/fablabnbg/inksc-ape-silhouette/releases>

9.11 Drone

9.11.1 qgroundcontrol

Descargamos [el appimage](#)

9.11.2 missionplanner

Para usar *Mission Planner* en Linux Mint se recomienda instalar los paquetes:

```
sudo apt install mono-complete festival
```

Descargamos el MissionPlanner desde [aquí](#)

Descripción de la instalación

Una vez descomprimido el zip ejecutamos: `mono MissionPlanner.exe`

9.12 node-red

Para instalar node-red en linux necesitamos instalar primero node.js. Hay varias formas de instalar node.js, yo voy a optar por instalar nvm que es el **n**ode **v**ersion **m**anager.

Para ello ejecutamos el siguiente comando (la versión actual de nvm es la 0.37.0)

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.37.0/install.sh | bash
```

El script de instalación añade las siguientes líneas al fichero `~/.bashrc`, nosotros las movemos al fichero `~/.profile`

```
export NVM_DIR="$HOME/.nvm"  
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm  
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion" # This loads nvm
```

Para comprobar la instalación usamos `command -v nvm` que nos devolverá nvm. `which` no funciona en este caso por que es un script para aplicarle `source`

9.12.1 Instalación de node.js

Ahora que tenemos nvm instalado, ya podemos instalar fácilmente la versión o versiones que queramos de node.js

```
nvm ls-remote      # para listar las versiones disponibles  
nvm install node  # instala la última versión disponible
```

9.12.2 Instalación de mosquitto

mosquitto es un *mqtt broker* muy sencillo y completo, aunque no es capaz de aguantar cargas grandes, para aprender es perfecto.

```
sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa  
sudo apt-get update  
sudo apt install mosquitto mosquitto-clients
```


Con esto queda el servicio instalado y funcionando (`scs mosquitto` o `systemctl status mosquitto`)

Si queremos el servicio parado para arrancarlo nosotros manualmente:

```
scsp mosquitto.service
scd mosquitto.service
```

Y sin alias sería:

```
sudo systemctl stop mosquitto.service
sudo systemctl disable mosquitto.service
```

Para arrancarlo cuando lo necesitemos le damos un *start* con `scst mosquitto.service` o `sudo systemctl start mosquitto.service`

9.12.3 Instalación de Influxdb

Seguimos el método para ubuntu:

```
wget -q0- https://repos.influxdata.com/influxdb.key | sudo apt-key add -
#source /etc/lsb-release
```

```
DISTRIB_ID=ubuntu
```

```
DISTRIB_CODENAME=focal
```

```
echo "deb https://repos.influxdata.com/${DISTRIB_ID} ${DISTRIB_CODENAME} stable" | su
```

```
apt update
```

```
apt install influxdb
```

```
apt install telegraf
```

9.12.4 Instalación de node-red

Una vez instalado el `node.js` instalar el `node-red` es muy fácil

```
nvm use node
```

```
npm install -g --unsafe-perm node-red
```

10 Aplicaciones de gráficos

10.1 LibreCAD

Diseño en 2D

```
sudo apt install librecad
```

10.2 FreeCAD

No hay ppa disponible para Ubuntu 20.

Instalamos *ApplImage* desde [aquí](#)

Dejo la instalación desde ppa como recordatorio.

```
sudo add-apt-repository ppa:freecad-maintainers/freecad-stable
sudo apt update
sudo install freecad
```

NOTA: the ccx package brings CalculiX support to the FEM workbench, and needs to be installed separately.

10.3 Inkscape

El programa libre para creación y edición de gráficos vectoriales.

```
sudo add-apt-repository ppa:inkscape.dev/stable
sudo apt update
sudo apt install inkscape
```

10.4 Gimp

El programa para edición y retocado de imágenes.

Parece que ahora mismo los repos están más actualizados que el ppa. Así que bastaría con:

```
sudo apt install gimp gimp-data gimp-texturize \
gimp-data-extras gimp-gap gmic gimp-gmic
```

De todas formas dejo aquí las instrucciones para instalar desde el ppa por si hacen falta algún día:

```
sudo apt remove gimp gimp-data
sudo add-apt-repository ppa:otto-kesselgulasch/gimp
sudo apt update
sudo apt upgrade
sudo apt install gimp gimp-data gimp-texturize \
gimp-data-extras gimp-gap gmic gimp-gmic gimp-python
```

10.4.1 Plugins de Gimp

Para instalar los principales plugins basta con:

```
sudo apt install gimp-plugin-registry
```

10.5 Krita

La versión disponible en orígenes de software está bastante por detrás de la disponible en la web. Basta con descargar el *Appimage* desde la [página web](#)

Lo copiamos a `~/apps/krita` y creamos un lanzador con el editor de menús.

Alternativamente también lo tenemos disponible por ppa en <https://launchpad.net/~kriticalime/+archive/ubuntu/ppa>

10.6 MyPaint

Desde el [github](#) tenemos disponible la última versión en formato *appimage*. La descargamos la dejamos en `~/apps` y creamos un acceso con *Menulibre*, como siempre.

10.7 Alchemy

Igual que el *MyPaint* descargamos desde [la página web](#), descomprimos en `~/apps` y creamos un acceso con *Menulibre*.

10.8 Capturas de pantalla

El *flameshot* cubre el 99% de mis necesidades: `sudo apt install flameshot`

El *ksnip* por si tenemos que hacer una captura con retardo lo instalé con un *appimage*.

Shutter vuelve a estar disponible, al instalar desde este ppa ya queda con las opciones de edición habilitadas:

```
sudo add-apt-repository ppa:linuxuprising/shutter
sudo apt update
sudo apt install shutter
```

10.9 Reoptimizar imágenes

10.9.1 ImageMagick

Instalamos desde los repos, simplemente:

```
sudo apt install imagemagick
```

10.9.2 Imagine

Nos bajamos un *AppImage* desde el [github](#) de la aplicación

10.10 dia

Un programa para crear diagramas

```
sudo apt install dia dia-shapes gsfonds-x11
```

10.11 Blender

Bajamos el Blender linkado estáticamente de [la página web](#) y lo descomprimos en `~/apps/blender`.

El paquete incluye un fichero `blender.desktop` que podemos editar y copiar en `~/.local/share/applications`.

10.12 Structure Synth

Instalado desde repos, junto con sunflow para explorar un poco.

```
sudo apt install structure-synth sunflow
```

10.13 Heron animation

El proyecto parece abandonado. El software ya no funciona en el último linux.

10.14 Stopmotion

Primero probamos el del repo: `sudo apt install stopmotion`

10.15 Instalación del driver digiment para tabletas gráficas Huion

He intentado un par de veces instalar con el fichero deb pero parece que no funciona.

Para hacer la instalación via DKMS el truco está en:

- Dejar el código fuente en un directorio de la forma `/usr/src/<PROJECTNAME>-<VERSION>`
- Lanzar el build pero usando esta vez `<PROJECTNAME>/<VERSION>`

Descargamos los últimos drivers desde [la página oficial de releases](#), en el momento de escribir esto descargamos la versión V9.

Descomprimos en `/usr/src/digimend-9`

```
cd /usr/src
sudo xvzf <path-to-digimend-kernel-drivers-9> .
sudo dkms build digimend-kernel-drivers/9
sudo dkms install digimend/9
```

Para comprobar:

```
xinput --list
dkms status
```

Referencia:

- [Aquí](#)

11 Sonido

11.1 Spotify

Spotify instalado desde las opciones de Linux Mint via flatpak.

11.2 Audacity

El ppa de Audacity no permite instalar la última versión. Podemos instalarla via flatpak o simplemente instalar la de los repos (no es la última)

Es de esperar que al final la añadan al ppa así que dejamos aquí las instrucciones.

Añadimos ppa:

```
sudo add-apt-repository ppa:ubuntuhandbook1/audacity
sudo apt-get update
sudo apt install audacity
```

Instalamos también el plugin [Chris's Dynamic Compressor plugin](#)

11.3 Clementine

La version disponible en los orígenes de software parece al dia:

```
sudo apt install clementine
```

12 Video

12.1 Shotcut

Nos bajamos la *AppImage* para Linux desde la [página web](#).

La dejamos en ~/apps/shotcut y:

```
cd
chmod 744 Shotcutxxxxxx.AppImage
./Shotcutxxxxxx.AppImage
```

12.2 kdenlive

Está disponible [en la web](#) como ppa o como *appimage*. Lo he bajado como *appimage* para probarlo.

12.3 Openshot

También descargado desde [su web](#) como *appimage*, para probar. Tienen un ppa disponible.

12.4 Avidemux

Descargado *appimage* desde [la web](#)

12.5 Handbrake

Instalado como flatpak desde [su web](#).

12.6 Grabación de screencast

12.6.1 Vokoscreen, Kazam y SimpleScreenRecorder

Instalados desde los repos oficiales:

```
sudo apt update
sudo apt install vokoscreen vokoscreen-ng kazam simplescreenrecorder
```

Escoge el que más te guste.

12.6.2 OBS

Añadimos el repositorio

```
sudo add-apt-repository ppa:obsproject/obs-studio
sudo apt update
sudo apt install obs-studio
```

12.7 Grabación de podcast

12.7.1 Mumble

Podemos instalarlo desde flatpak o bajarnos [el paquete antiguo](#) (parece que funciona bien).

Mumble no está disponible desde el PPA, aunque dejo aquí las instrucciones por si lo corrigen.

```
sudo add-apt-repository ppa:mumble/release
sudo apt update
sudo apt install mumble
```

12.8 Cliente de youtube smtube

Instalado el ppa siguiendo instrucciones de [su página web](#).

```
sudo add-apt-repository ppa:rvm/smplayer
sudo apt-get update
sudo apt-get install smtube
```

13 Fotografía

13.1 Rawtherapee

Bajamos el AppImage desde la [página web](#) al directorio ~/apps/rawtherapee.

```
cd
chmod 744 RawTherapeexxxxxx.AppImage
./RawTherapeexxxxxx.AppImage
```

Al ejecutarla la primera vez ya se encarga la propia aplicación de integrarse en nuestro sistema.

13.2 Darktable

Instalamos ppa (ver [esta web](#))

```
echo 'deb http://download.opensuse.org/repositories/graphics:/darktable/xUbuntu_20.04
curl -fsSL https://download.opensuse.org/repositories/graphics:darktable/xUbuntu_20.04
-dearmor | sudo tee /etc/apt/trusted.gpg.d/graphics:darktable.gpg > /dev/null
sudo apt update
sudo apt install darktable
```

Se instala la última versión de Darktable (3.0.2)

OJO: Conviene renombrar el fichero de claves de darktable, a nuestro linux no le gustan los ficheros con un ':' Revisa /etc/apt/trusted.gpg.d/

13.3 Digikam

Instalado desde la [página web](#) de la aplicación con appimage.

13.4 Webcamoid

Descargada la appimage desde la [página web](#)

14 Seguridad

14.1 Autenticación en servidores por clave pública

Generar contraseñas para conexión servidores remotos:


```
cd ~
ssh-keygen -b 4096 [-t dsa | ecdsa | ed25519 | rsa | rsa1]
cat .ssh/
```

Solo resta añadir nuestra clave pública en el fichero `authorized_keys` del servidor remoto.

```
cat ~/.ssh/id_XXX.pub | ssh user@hostname 'cat >> .ssh/authorized_keys'
```

[¿Cómo funciona esto?](#)

14.2 Claves gpg

`gpg --gen-key` Para generar nuestra clave.

- **Siempre** hay que ponerle una fecha de expiración, la puedes cambiar más tarde.
- **Siempre** hay que escoger la máxima longitud posible

14.3 Seahorse

Para manejar todas nuestras claves con comodidad:

```
sudo apt install seahorse
```

14.4 Conexión a github con claves ssh

Usando este método podemos conectarnos a github sin tener que teclear la contraseña en cada conexión.

14.4.1 Claves ssh

Podemos echar un ojo a nuestras claves desde seahorse la aplicación de gestión de claves que hemos instalado. También podemos ver las claves que tenemos generadas:

```
ls -al ~/.ssh
```

En las claves listadas nuestras claves públicas aparecerán con extensión `.pub`

También podemos comprobar que claves hemos añadido ya a nuestro agente ssh con:

```
ssh-add -l
```

Para generar una nueva pareja de claves ssh:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Podremos dar un nombre distintivo a los ficheros de claves generados y poner una contraseña adecuada a la clave. Si algún día queremos cambiar la contraseña:

```
ssh-keygen -p
```

Ahora tenemos que añadir nuestra clave ssh en nuestra cuenta de github, para ello editamos con nuestro editor de texto favorito el fichero `~/.ssh/id_rsa.pub` y copiamos el contenido integro. Después pegamos ese contenido en el cuadro de texto de la web de github.

Para comprobar que las claves instaladas en github funcionan correctamente:

```
ssh -T git@github.com
```

Hi salvari! You've successfully authenticated, but GitHub does not provide shell access.

Este mensaje indica que todo ha ido bien.

Ahora en los repos donde queramos usar ssh debemos cambiar el remote:

```
git remote set-url origin git@github.com:$USER/$REPONAME.git
```

14.5 Signal

El procedimiento recomendado en la página oficial:

```
curl -s https://updates.signal.org/desktop/apt/keys.asc | sudo apt-key add -  
echo "deb [arch=amd64] https://updates.signal.org/desktop/apt xenial main" | sudo tee /etc/apt/sources.list.d/signal.list  
sudo apt update && sudo apt install signal-desktop
```

NOTA: Parece que no funciona. Lo he instalado via *flatpack*

14.6 Element (cliente de matrix.org)

Instalamos con:

```
sudo apt install -y wget apt-transport-https
```

```
sudo wget -O /usr/share/keyrings/riot-im-archive-keyring.gpg https://packages.riot.im
```

```
echo "deb [signed-by=/usr/share/keyrings/riot-im-archive-keyring.gpg] https://package
```

```
sudo apt update
```

```
sudo apt install element-desktop
```

14.7 Lector DNI electrónico

Descargamos la aplicación en formato .deb desde [la página de descargas del portal DNle](#).

Automáticamente nos instalará las dependencias: libccid, pcsc y pinentry-gtk2. A mayores instalamos:

```
sudo apt-get install pcsc-tools opensc
```

El opensc no es necesario para el DNle, pero nos permite usar otras tarjetas.

Como root ejecutamos pcsc_scan:

```
root@rasalhague:~# pcsc_scan
PC/SC device scanner
V 1.4.23 (c) 2001-2011, Ludovic Rousseau <ludovic.rousseau@free.fr>
Compiled with PC/SC lite version: 1.8.11
Using reader plug'n play mechanism
Scanning present readers...
Waiting for the first reader...
```

Si insertamos el lector veremos algo como esto:

```
root@rasalhague:~# pcsc_scan
PC/SC device scanner
V 1.4.23 (c) 2001-2011, Ludovic Rousseau <ludovic.rousseau@free.fr>
Compiled with PC/SC lite version: 1.8.11
Using reader plug'n play mechanism
Scanning present readers...
Waiting for the first reader...found one
Scanning present readers...
0: C3PO LTC31 v2 (11061005) 00 00
```

```
Wed Jan 25 01:17:20 2017
```

```
Reader 0: C3PO LTC31 v2 (11061005) 00 00
Card state: Card removed,
```

Si insertamos un DNI veremos que se lee la información de la tarjeta insertada:

```
Reader 0: C3P0 LTC31 v2 (11061005) 00 00  
Card state: Card inserted,
```

y mas rollo

Para abrir los certificados en el navegador Firefox, nos lo explica [esta página de la AEAT](#)

Como se puede ver el link de la AEAT, los pasos necesarios para Firefox son:

1. Vamos a preferencias y buscamos 'cert'
2. En el diálogo de certificados abrimos los Dispositivos de Seguridad (*Security Devices*)
3. Para dar de alta un nuevo dispositivo pulsamos el botón Cargar (*Load*)
4. Damos un nombre (p.ej. DNIE) y asociamos el driver: `/usr/lib/libpkcs11-dnie.so`
5. Adicionalmente podemos Cargar (crear), otro dispositivo con el driver `opensc`, no es necesario para el DNIE pero nos añade soporte para otras tarjetas. (Nombre: OtrasTarjetas, Driver: `'/usr/lib/x86_64-linux-gnu/pkcs11/opensc-pkcs11.so'`)

NOTA:

Para cada tarjeta puede hacer falta un driver diferente, tendrás que investigar con ayuda del `pcsc_scan` y herramientas similares.

15 Virtualizaciones y contenedores

15.1 Instalación de *virtualBox*

AVISO IMPORTANTE

Tenemos dos formas de instalar *Virtualbox*, desde los repos oficiales de la aplicación o desde los propios de Linux Mint (Ubuntu Focal Fossa)

Si descargamos los oficiales de *Virtualbox* se instalará el paquete `python-is-python2`, eso hara que el python por defecto de nuestro sistema sea el dos. A cambio tendremos la última versión de *Virtualbox*

Si lo instalamos con los repos de Ubuntu, podemos tener instalado el paquete `python-is-python3` (esta es mi opción favorita)

15.1.1 Instalación desde Ubuntu

```
sudo apt install virtualbox virtualbox-ext-pack virtualbox-guest-addition-iso
```

15.1.2 Instalación desde repos oficiales

Lo hacemos con los orígenes de software oficiales (alternativamente, podríamos hacerlo manualmente):

```
# Importamos la clave gpg
wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc -O-
| sudo apt-key add -
```

```
# Añadimos el nuevo origen de software
sudo add-apt-repository "deb [arch=amd64] https://download.virtualbox.org/virtualbox/d
release; echo "$UBUNTU_CODENAME" contrib"
```

```
# Actualizamos la base de datos de paquetes
sudo apt update
```

Ahora podemos comprobar que además del paquete *virtualbox* tenemos varios paquetes con número de versión (p.ej. `_virtualbox.6.1`), estos últimos son los que hemos añadido (compruébalo con `apt-cache policy [nombrepaquete]`)

Instalamos el que nos interesa:

```
sudo apt install virtualbox-6.1
```

ATENCIÓN

The following additional packages will be installed:

```
python-is-python2
```

Descargamos también el [VirtualBox Extension Pack](#), este paquete lo podemos instalar desde el propio interfaz de usuario del *VirtualBox*, o bien con el siguiente comando:

```
sudo VBoxManage extpack install ./Oracle_VM_VirtualBox_Extension_Pack-
6.1.2.vbox-extpack
```

Sólo nos queda añadir nuestro usuario al grupo `vboxusers`, con el comando `sudo gpasswd -a username vboxusers`, y tendremos que cerrar la sesión para

refrescar nuestros grupos.

15.2 qemu

Un par de comprobaciones previas:

- El comando `egrep -c '(vmx|svm)' /proc/cpuinfo` debe devolvernos un número mayor que cero si nuestro sistema soporta virtualización.
- El comando `kvm-ok` nos sirve para comprobar que la virtualización hardware no está deshabilitada en la BIOS (puede que tengas que ejecutar `apt install cpu-checker`)

Instalamos desde el repo oficial:

```
sudo apt install qemu-kvm libvirt-daemon-system libvirt-clients bridge-  
utils virtinst virt-manager  
sudo apt install virt-viewer
```

qemu-kvm nos da la emulación hardware para el hipervisor KVM

libvirt-daemon-system los ficheros de configuración para ejecutar el demonio `libvirt` como servicio

libvirt-clients software para gestionar plataformas de virtualización

bridge-utils utilidades de linea de comandos para configurar bridges ethernet

virtinst utilidades de linea de comandos para crear máquinas virtuales

virt-manager un interfaz gráfico junto con utilidades de linea de comandos para gestionar máquinas virtuales a través de *libvirt*

Solo queda añadir nuestro usuario a los grupos:

```
sudo gpasswd -a username libvirt  
sudo gpasswd -a username kvm
```

Podemos comprobar el estado del servicio con `scs libvirtd (systemctl status libvirtd)`.

15.2.1 Referencias

- [How to install KVM on Ubuntu 20.04 Graphical & headless server](#)
- [How to Install Kvm on Ubuntu 20.04](#)
- [How to Install KVM on Ubuntu 20.04](#)

15.3 Docker

Tenemos que añadir el repositorio correspondiente a nuestra distribución:

```
# Be safe
sudo apt remove docker docker-engine docker.io
sudo apt autoremove
sudo apt update

# Install pre-requisites
sudo apt install ca-certificates curl

# Import the GPG key

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-
key add -

# Next, point the package manager to the official Docker repository

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(
release; echo "$UBUNTU_CODENAME") stable"

# Update the package database

sudo apt update
#

apt-cache policy docker-ce

sudo apt install docker-ce

sudo gpasswd -a username docker
```

Esto dejará el servicio *docker* funcionando y habilitado (arrancará en cada reinicio del ordenador)

La forma de pararlo es:

```
sudo systemctl stop docker
sudo systemctl disable docker
systemctl status docker
```

Añadimos el *bundle docker* en nuestro fichero `~/.zshrc` para tener autocom-

pletado en comandos de docker.

Para usar *docker* tendremos que arrancarlo, con los alias de nuestro sistema para *systemd* ejecutamos:

```
scst docker # para arrancar el servicio
```

```
scsp docker # para parar el servicio
```

15.3.1 docker-compose

- Nos bajamos la última versión disponible de [las releases de github](#)
- Movemos el fichero que hemos descargado a `/usr/local/bin/docker-compose`
- Y le damos permisos de ejecución `sudo chmod +x /usr/local/bin/docker-compose`

15.3.2 Kitematic

Un interfaz gráfico para *Docker*. En su [página de releases](#) bajamos la última para Ubuntu e instalamos con el gestor de paquetes.

La verdad es que me gusta más el CLI.

16 Utilidades para mapas y cartografía

16.1 josm

Descargamos y añadimos la clave gpg:

```
wget -q https://josm.openstreetmap.de/josm-apt.key -O- | sudo apt-key add -
```

Añadimos el origen de software:

```
sudo add-apt-repository "deb [arch=amd64] https://josm.openstreetmap.de/apt $(. /etc/os-release; echo "$UBUNTU_CODENAME") universe"
```

Y ahora procedemos a la instalación:

```
sudo apt update
sudo apt install openjfx josm
```


Alternativamente también podemos instalar la versión “nightly” con el siguiente comando, pero tendréis actualizaciones diarias:

```
sudo apt josm-latest
```

Ya estamos listos para editar Open Street Map offline.

16.2 MOBAC

Bajamos el paquete desde [la página web](#) y descomprimos en ~/apps/mobac como de costumbre nos creamos una entrada de menú con *MenuLibre*.

Conviene bajarse wms adicionales para MOBAC y leerse [la wiki](#)

16.2.1 Referencias

*[Cartografía digital] (<https://digimapas.blogspot.com.es/2015/01/oruxmaps-vii-mapas-de-mobac.html>)

16.3 QGIS

Añadimos la clave gpg:

```
wget -q https://qgis.org/downloads/qgis-2019.gpg.key -O- | sudo apt-key add -
```

Ejecutamos:

```
sudo add-apt-repository "deb [arch=amd64] https://qgis.org/debian $(. /etc/os-release; echo "$UBUNTU_CODENAME") main"
```

E instalamos como siempre

```
sudo apt update
sudo apt install qgis
```

16.3.1 Referencias

- [Conectar WMS con QGIS](#)
- [Importar OSM en QGIS](#)
- [Learn OSM](#)
- [QGIS Tutorials](#)

17 Recetas variadas

17.1 Añadir las claves GPG de un repo

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys F7E06F06199EF2F2
```

17.2 Solucionar problemas de menús duplicados usando menulibre

Nota: Ya no uso *MenuLibre*

En el directorio `~/.config/menus/applications-merged` borramos todos los ficheros que haya.

17.3 Formatear memoria usb

“The driver descriptor says the physical block size is 2048 bytes, but Linux says it is 512 bytes.”

Este comando borró todas las particiones de la memoria:

```
sudo dd if=/dev/zero of=/dev/sdd bs=2048 count=32 && sync
```

I’m assuming your using gparted.

First delete whatever partitions you can...just keep pressing ignore.

There will be one with a black outline...you will have to unmount it...just right click on it and unmount.

Again you will have to click your way through ignore..if fix is an option choose it also.

Once all this is done... you can select the device menu and choose new partition table.

Select MSdos

Apply and choose ignore again.

Once it’s done it show it’s real size.

Next you can format the drive to whichever file system you like.

It's a pain in the behind this way, but it's the only way I get it done..I put live iso's on sticks all the time and have to remove them. I get stuck going through this process every time.

17.4 Copiar la clave pública ssh en un servidor remoto

```
cat /home/tim/.ssh/id_rsa.pub | ssh tim@just.some.other.server 'cat >>
.ssh/authorized_keys'
```

O también:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub username@remote.server
```

17.5 ssh access from termux

<https://linuxconfig.org/ssh-into-linux-your-computer-from-android-with-termux>

17.6 SDR instalaciones varias

Vamos a trastear con un dispositivo RTL-SDR.com.

Tenemos un montón de información en el blog de [SDR Galicia](#) y tienen incluso una guía de instalación muy completa, pero yo voy a seguir una guía un poco menos ambiciosa, por lo menos hasta que pueda hacer el curso que imparten ellos mismos (SDR Galicia)

La guía en cuestión la podemos encontrar [aquí](#)

Seguimos los pasos de instalación:

- La instalación de git, cmake y build-essential ya la tengo hecha.

```
sudo apt-get install libusb-1.0-0-dev
```

17.7 Posible problema con modemmanager y micros programables

Programando el *Circuit Playground Express* con el *Arduino IDE* tenía problemas continuos para hacer los *uploads*, al parecer el servicio *ModemManager* es el culpable, se pasa todo el tiempo capturando los nuevos puertos serie por que considera que todo es un modem.

Una prueba rápida para comprobarlo: `sudo systemctl stop ModemManager`

Con esto funciona todo bien, pero en el siguiente arranque volvera a cargarse.

Para dar una solución definitiva se puede programar una regla para impedir que el *ModemManager* capture el puerto con un dispositivo

Creamos un fichero con permisos de root en el directorio `/etc/udev/rules.d` que llamaremos: `99-arduino.rules`

Dentro de ese fichero especificamos los codigos VID/PID que se deben ignorar:

```
# for arduino brand, stop ModemManager grabbing port
ATTRS{idVendor}=="2a03", ENV{ID_MM_DEVICE_IGNORE}="1"
# for sparkfun brand, stop ModemManager grabbing port
ATTRS{idVendor}=="1b4f", ENV{ID_MM_DEVICE_IGNORE}="1"
```

Ojo que si tienes SystemV no va a funcionar.

<https://starter-kit.nettigo.eu/2015/serial-port-busy-for-avrdude-on-ubuntu-with-arduino-leonardo-eth/>

<https://www.codeproject.com/Tips/349002/Select-a-USB-Serial-Device-via-its-VID-PID>

17.8 Programar los nanos con chip ch340 o ch341

Linux mapea el chip correctamente en un puerto `/dev/ttyUSB0` y con eso basta, que no te lien con el cuento de "drivers para linux"

Todo lo que hace falta es configurar correctamente el *Arduino IDE*, hay que escoger:

```
Board: "Arduino Nano"
Processor: "ATmega168"
Port: "/dev/ttyUSB0"
```

Y ya funciona todo.

17.9 Linux Mint 20 es *python agnostic*

En principio no podemos invocar a python por que no se ha escogido una versión por defecto.

Tenemos dos opciones:

```
apt install python-is-python2  
apt install python-is-python3  
"test"
```