Bitácora Linux Mint Ulyana 20

Sergio Alvariño salvari@gmail.com

junio-2020

Resumen

Bitácora de mi portatil
Solo para referencia rápida y personal.

Índice general

1 Introducción

Mi portátil es un ordenador Acer 5755G con las siguientes características:

- Core i5 2430M 2.4GHz
- NVIDIA Geforce GT 540M
- 8Gb RAM
- 750Gb HD

Mi 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 sobrecalientamiento, pero en las últimas versiones de Linux instalando el driver de Nvidia parece funcionar correctamente.

2 Programas básicos

2.1 Linux Mint

Linux Mint incluye sudo [1] y las aplicaciones que uso habitualmente para gestión de paquetes por defecto (*aptitude* y *synaptic*).

Tampoco voy a enredar nada con los orígenes del software (de momento)

[1] ya no incluye gksu pero tampoco es imprescindible

2.2 Firmware

Ya no es necesario intalar los paquetes de *microcode* la instalación de Tricia 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 toda nuestra instalación quede reflejada en los repos.

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 añado a mano los ficheros de configuración al repo.

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 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*

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

2.5 Fuentes 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`

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.

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.

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

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

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
```

2.8 Algunos programas de control

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
```

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

2.10 Dropbox

Lo instalamos desde el software manager.

2.11 Chrome

Instalado 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`.

2.12.1 Freeplane

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

1. descargamos desde la web.
2. Descomprimimos en `~/apps/freeplane`
3. Creamos enlace simbólico
4. Añadimos a los menús

2.12.2 Telegram Desktop

Cliente de Telegram, descargado desde la página web.

2.12.3 Tor browser

Descargamos desde la página oficial del proyecto Descomprimimos en `~/apps/` y ejecutamos desde terminal:

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

2.12.4 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" | s
browser-release.list

sudo apt update

sudo apt install brave-browser
```

2.12.5 TiddlyDesktop

Descargamos desde la página web, descomprimimos y generamos la entrada en el menú.

2.12.6 Joplin

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

La instalamos en el directorio `~/apps/joplin` descargando el AppImage desde la página web

Como siempre nos creamos una entrada en el menú.

2.13 Terminal y Shell

Por defecto tenemos instalado `bash`.

2.13.1 bash-git-promt

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 python-pip python-virtualenv virtualenv python3-pip
apt install virtualenvwrapper
```

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 seria `~/.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 "$HOME/.local/share/virtualenvs" ] ; then
    WORKON_HOME="$HOME/.local/share/virtualenvs"
fi
```

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* y el fichero del tema *gnzh* en `~/.antigen/bundles/robbyrussell/oh-my-zsh/themes/gnzh.zsh-theme`

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

2.13.3 tmux

Esto no tiene mucho que ver con los shell, lo he instalado para aprender a usarlo.

```
sudo apt install tmux
```

El tao de tmux

Nota: Instalar *rxvt* junto con tmux como terminal alternativo

---

2.14 Utilidades

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

Instalamos *gpick* con `sudo apt install gpick`

2.15 Codecs

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

2.16 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 htop

```
sudo apt install htop
```

3.2 gparted

Instalamos *gparted* para poder formatear memorias usb

```
sudo apt install gparted
```

3.3 wkhtmltopdf

```
sudo apt install wkhtmltopdf
```

4 Internet

5 Rclone

Instalamos desde la página web, siempre que te fies obviamente.

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

5.1 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
```

5.2 Referencias

- Como usar rclone (blogdelazaro)
- y con cifrado (blogdelazaro)
- Documentación

6 Documentación

6.1 Vanilla LaTeX

Para instalar la versión más reciente de LaTeX hay que aplicar este truco.

```
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 practicamente 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 lineas:

```
# 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 linea:

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

6.1.1 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.

6.1.2 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 linea:

```
<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
```

6.2 Tipos de letra

Creamos el directorio de usuario para tipos de letra:

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

6.3 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* y enlazado algunas fuentes (Hack y Menlo)

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

6.4 Pandoc

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

Instalado el *Pandoc* descargando paquete deb desde la página web del proyecto.

Además descargamos plantillas adicionales desde este repo ejecutando los siguientes comandos:

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

6.5 Calibre

La mejor utilidad para gestionar tu colección de libros electrónicos.

Ejecutamos lo que manda la página web:

```
sudo -v && wget -nv -O- https://raw.githubusercontent.com/kovidgoyal/calibre/master/set
installer.py \
| sudo python -c "import sys; main=lambda:sys.stderr.write('Download failed\n'); exec(sy
```

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 empy 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)

6.6 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.

Para tener una versión más actualizada instalamos:

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

6.6.1 Cambiados algunos valores por defecto

He cambiado los siguientes valores en las dos versiones, non están exactamente en el mismo menú pero no son díficiles 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

6.6.2 Solucionados problemas de *hyphenation*

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

- hyphen-es
- hyphen-gl

Y ahora funciona correctamente.

6.7 Foliate: lector de libros electrónicos

Instalado el paquete deb desde su propio github

7 Desarrollo software

7.1 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
```

7.2 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
```

7.3 Emacs

Instalado emacs desde los repos:

```
sudo aptitude install emacs
```

7.4 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.

7.4.1 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
```

7.4.2 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
```

7.4.3 gdc

Instalado con:

```
sudo aptitude install gdc
```

7.4.4 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
```

7.4.5 Emacs para editar D

Instalados los siguientes paquetes desde Melpa

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

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

7.5 C, C++

7.5.1 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.

Descomprimimos 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`

7.6 Processing

Bajamos los paquetes de las respectivas páginas web, descomprimimimos 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.

7.7 openFrameworks

Nos bajamos los fuentes para linux 64bits desde la página web del proyecto, y las descomprimimos 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*.

7.8 Python

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

```
python -V
Python 2.7.12

python3 -V
Python 3.5.2
```

7.8.1 Paquetes de desarrollo

Para que no haya problemas a la hora de instalar paquetes en el futuro conviene que instalemos los paquetes de desarrollo:

```
sudo apt install python-dev
sudo apt install python3-dev
```

7.8.2 pip, virtualenv, virtualenvwrapper, virtualfish

Los he instalado a nivel de sistema.

pip es un gestor de paquetes para **Python** que facilita la instalación de librerías y utilidades.

Para poder usar los entornos virtuales instalaremos también *virtualenv*.

Instalamos los dos desde aptitude:

```
sudo apt install python-pip python-virtualenv virtualenv python3-pip
```

virtualenv es una herramienta imprescindible en Python, pero da un poco de trabajo, así que se han desarrollado algunos frontends para simplificar su uso, para *bash* y *zsh* usaremos *virtualenvwrapper*, y para *fish* el *virtualfish*. Como veremos son todos muy parecidos.

Instalamos el virtualwrapper:

```
sudo apt install virtualenvwrapper -y
```

Para usar *virtualenvwrapper* tenemos que hacer:

```
source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
```

O añadir esa linea a nuestros ficheros *.bashrc* y/o *.zshrc*

Definimos la variable de entorno *WORKON_HOME* para que apunte al directorio por defecto, `~/.local/share/virtualenvs`. En ese directorio es donde se guardarán nuestros entornos virtuales.

En el fichero `.profile` añadimos:

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

Aquí tenemos la referencia de comandos de *virtualenvwrapper*

Por último, si queremos tener utilidades parecidas en nuestro *fish shell* instalamos *virtualfish*:

```
sudo pip install virtualfish
```

Aquí tenemos la documentación de *virtualfish* y la descripción de todos los comandos y plugins disponibles.

7.8.3 pipenv

No lo he instalado, pero en caso de instalación mejor lo instalamos a nivel de usuario con:

```
pip install --user pipenv
```

7.8.4 Instalación del Python 3.8 (última disponible)

Ejecutamos:

```
sudo apt install python3.8 python3.8-dev python3.8-venv
```

7.8.5 Instalación de bpython y ptpython

bpython instalado desde repos `sudo apt install bpython bpython3`

ptpython instalado en un virtualenv para probarlo

7.8.6 Emacs para programar python

7.8.6.1 elpy: Emacs Python Development Enviroment Para instalar `elpy` necesitamos intalar previamente *venv* el nuevo gestor de *virtualenvs* en Python 3.:

```
sudo apt install python3-venv
```

En el fichero ~/.emacs necesitamos activar el módulo *elpy*:

```
;;-------------------------------------------------------------------
---
;; elpy
(elpy-enable)
```

En cuanto activemos *elpy* tendremos autocompletado del código y errores sintácticos. Merece la pena leerse toda la documentación

7.8.6.2 Flycheck Para tener análisis sintáctico en tiempo real mientras estamos programando:

Añadimos a nuestro fichero ~/.emacs:

```
;; Enable Flycheck

(when (require 'flycheck nil t)

  (setq elpy-modules (delq 'elpy-module-flymake elpy-modules))

  (add-hook 'elpy-mode-hook 'flycheck-mode))
```

7.8.6.3 Formateado Usando *autopep8* o *black* tendremos autoformateado del código como paso previo a salvar el mismo en disco. (Yo aún no he probado *black*)

```
# and autopep8 for automatic PEP8 formatting
sudo apt install python-autopep8
# and yapf for code formatting (innecesario)
# sudo apt install yapf yapf3
```

Y añadimos la sección siguiente a nuestro fichero ~/.emacs

```
;; Enable autopep8

(require 'py-autopep8)

(add-hook 'elpy-mode-hook 'py-autopep8-enable-on-save)
```

7.8.6.4 jedi Jedi le da ciertos superpoderes al autocompletado visualizando la documentación de cada propuesta de autocompletado.

Instalamos previamente:

```
sudo apt install python-jedi python3-jedi
# flake8 for code checks
sudo apt install flake8 python-flake8 python3-flake8
```

Y añadimos la sección en el fichero ~/.emacs:

```
;;-------------------------------------------------------------------
---
;; elpy
(elpy-enable)
(setq elpy-rpc-backend "jedi")

(add-hook 'python-mode-hook 'jedi:setup)
(setq jedi:complete-on-dot t)
```

Desde *Emacs* ejecutamos: `alt-x jedi:install-server`

7.8.7 Jupyter

Una instalación para pruebas.

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

7.9 neovim

Vamos a probar *neovim*:

```
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
```

7.9.0.1 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
```

7.10 Firefox developer edition

El rollo de siempre, descargar desde la página web descomprimir en `~/apps` y crear un lanzador.

7.11 Navegadores cli

Herramientas útiles para depuración web

```
sudo apt install httpie links
```

7.12 MariaDB

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

Primero añadimos los reports

Añadimos la clave de firma:

```
sudo apt-get install software-properties-common
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://ftp.icm.edu.pl/pub/unix
```

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

```
# MariaDB 10.4 repository list - created 2020-01-26 10:37 UTC
# http://downloads.mariadb.org/mariadb/repositories/
deb [arch=amd64,arm64,ppc64el] http://ftp.ubuntu-tw.org/mirror/mariadb/repo/10.4/ubunt
# deb-src http://ftp.ubuntu-tw.org/mirror/mariadb/repo/10.4/ubuntu bionic 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
```

Y ahora aseguramos la instación 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
```

7.13 Squirrel SQL Client

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

Como de costumbre descomprimimos 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 `maria-java-client-2.6.0.jar`

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

7.14 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 bionic-
cran35/'
sudo apt install r-base
```

7.14.1 R-studio

Descargamos la última versión disponible de *R-studio* desde la página web

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

7.15 Octave

Instalado desde flatpak

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

8 Desarrollo hardware

8.1 Arduino IDE

Bajamos los paquetes de la página web, descomprimimimos 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 <usrname> tty
sudo gpasswd --add <usrname> dialout
sudo gpasswd --add <usrname> uucp
sudo gpasswd --add <usrname> plugdev
```

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

Fichero `90-extraacl.rules` mete mi usario 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"
```

8.1.1 Añadir soporte para *Feather M0*

Arrancamos el IDE Arduino y en la opción de *Preferences::Aditional Boar 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

8.1.2 Añadir soporte para *Circuit Playground Express*

Bastaría con instalar *Arduino SAMD Boards*

8.1.3 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, y tiene mejor pinta)

8.1.4 Añadir soporte para ESP32

8.1.5 Añadir biblioteca de soporte para Makeblock

Nota: Pendiente de instalar

Clonamos el repo oficial en github.

Una vez que descarguemos las librerias 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.

8.2 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.

8.3 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`

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

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

8.4 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:js-reynaud/kicad-5
sudo apt-get update
sudo apt-get install kicad
sudo apt install kicad-footprints kicad-libraries kicad-packages3d kicad-symbols kicad-templates
```

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

Algunas librerías alternativas:

- Freetronics una libreria 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.

8.5 Analizador lógico

Mi analizador es un OpenBench de Seedstudio, aquí hay mas info

8.5.1 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*.

8.5.2 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 descomprimimos 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.

8.5.3 OLS

Nota: Pendiente de instalar

Página oficial

8.6 IceStudio

Instalamos dependencias con `sudo apt install xclip`

Bajamos el *AppImage* desde el github de IceStudio y lo dejamos en `~/apps/icestudio`

8.7 PlatformIO

8.7.1 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 puec
```

Ahora

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

Seguimos las instrucciones de aqui

8.7.2 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 linea de comandos pero no es conveniente hacerlo.

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

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

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

8.7.3 vscodium

```
wget -qO - https://gitlab.com/paulcarroty/vscodium-deb-rpm-repo/raw/master/pub.gpg | su
key add -
```

```
echo 'deb https://gitlab.com/paulcarroty/vscodium-deb-rpm-repo/raw/repos/debs/ vscodiu
-append /etc/apt/sources.list.d/vscodium.list
sudo apt update && sudo apt install codium
```

8.7.4 Editor Atom

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

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

8.8 RepRap

8.8.1 OpenScad

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

8.8.2 Slic3r

Descargamos la estable desde la página web y como de costumbre descomprimimos en `~/apps` y creamos un lanzador con *MenuLibre*

8.8.3 Slic3r Prusa Edition

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

8.8.4 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.

8.8.5 Ultimaker Cura

Descargamos el *AppImage* desde la página web

8.8.6 Pronterface

Seguimos las instrucciones para Ubuntu Bionic:

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.

8.9 Cortadora de vinilos

8.9.1 Inkcut

Instalado en un entorno virtual:

```
mkvirtualenv -p `which python3` inkcut

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

pip install PyQt5

pip install inkcut
```

8.9.2 Plugin para inkscape

Instalamos dependencias:

```
pip install python-usb
```

Instalamos el fichero `.deb` desde la web https://github.com/fablabnbg/inkscape-silhouette/releases