Parte I

Instalar Arch Linux

Nota: Esta guía la escribí hace mucho y puede tener errores o poco detalle en partes importantes

1. Antes que nada

Arrancamos `archiso x86_64` . Se inicia sesión como `root` por defecto (arrancar `archiso x86_64` inicia una sesión con los máximos permisos)

Pregunta

¿No es un riesgo permitir que alguien con un USB pueda arrancar un OS con máximos permisos?

Sí. Pero si esa persona tuvo permiso para poner un USB en un puerto, y además arrancar la computadora desde cero, entonces ya es alguien con acceso físico, que es en sí un permiso. Además, las motherboards pueden denegar arranque a OS'es que no sepan responder a una clave en la memoria de arranque seguro.

Luego, comprobamos que el sistema está conectado a internet haciendo:

```
ping archlinux.org
```

Si la computadora recibe datos, entonces sí estamos. Podemos terminar la prueba con `Ctrl+c` [1].

También deberíamos escoger una disposición de teclado que sea cuando menos parecida a nuestro teclado físico para no tener problemas al escribir caracteres especiales como guiones o paréntesis. Para eso, usamos el programa `loadkeys` , pasando como argumento el archivo relevante (*e.g.* `loadkeys de-latin1`) [2]

Para ver la lista de disposiciones de teclado hacemos:

```
~ls /usr/share/kbd/keymaps/**/*.map.gz
```

Nótese que omitimos la ruta hacia el archivo `de-latin1.map.gz` y omitimos también su extensión [3]

[1] `Ctrl+C` envía una señal al proceso que estaba escuchando al teclado. Esta señal es `SIGINT` . Para más señales, ver TODO:Señales

[2] Mi teclado funcionó suficientemente bien con `es.map` , pero creo que `latin` es la adecuada

[3] De manera similar, la fuente de la consola se cambia con el programa `setfont` . Los archivos de fuentes están en `/usr/share/kbd/consolefonts`

2. Modos de arranque

Debemos revisar si la computadora arranca en modo **BIOS**, **UEFI**, ó `CSM` . Esto determinará algunas de las características de la instalación, como las referentes a la llamada *partición de arranque*, entre otras cosas.

Para saber en qué modo estamos, corremos `ls` así:

```
ls /sys/firmware/efi/efivars
```

Si podemos ver tal directorio sin ningún error, entonces la compu arranca en modo **UEFI**. Esto implica que `ArchISO` usó `systemd-boot` (un gestor de arranque) para cargar el kernel y disco RAM inicial (initrd) [4]

Según el modo (**UEFI, BIOS**) en el que arranque la compu, tendremos que particionar el disco de forma adecuada:

- Formato **MBR** (Master Boot Record) si la compu inicia en modo **BIOS**.
- Formato **GPT** (Global Partition Table) si la compu arranca en modo **UEFI**.

Además, habrá que respetar las limitaciones de dichos formatos, como el tamaño de una partición, etc. Aquí haremos un disco particionado para funcionar (arrancar, iniciar) en un sistema **UEFI**.

3. Sobre la fecha y hora

El *servicio* [5] `timedatectl` controla la fecha y hora del sistema. Para sincronizarlo con la red, hacemos:

```
timedatectl set-ntp true
```

Luego revisamos que sí esté sincronizado con:

```
timedatectl status
```

Más tarde escogeremos una zona horaria.

4. Particionar discos

Ya podemos preparar el disco que tendrá ArchLinux. Esto lo haremos con el programa `fdisk` .

Primero vemos los discos que hay conectados:

```
fdisk -l
```

El disco 0 sería `sda` , el disco 1 sería `sdb` , etc. Hay `sda` , `sdb` , `sdc` , ... , `sdx` .

Todas las particiones del disco `sdx` son `sdx1` , `sdx2` , ... , `sdxn` .

[4] Si no existe ese directorio, es porque la compu arranca en modo **BIOS** o en modo **CSM**, y ArchISO usó `syslinux` en vez de `systemd-boot` . Esta guía la escribí para motherboards con interfaz **UEFI**

[5] Un *servicio* es un proceso que no espera interacciones con el usuario más allá de iniciarlo o detenerlo manualmente. Por lo demás, este proceso está bajo su propia dirección o, más comúnmente, bajo la dirección del llamado *init system*. En nuestro caso, el *init system* es `systemd` . Ver la nota TODO: Systemd

Cuando decidamos en qué disco queremos trabajar, lo seleccionamos. Aquí como ejemplo trabajaremos con `sda` , pero este puede no ser el caso. **SIEMPRE hay que revisar las cosas 2 veces** cuando se trabaja con `fdisk` .

SIEMPRE hay que revisar las cosas 2 veces (o más) cuando se trabaja con `fdisk`

Para seleccionar `sda` (como ejemplo):

```
fdisk /dev/sda
```

Ahora prepararemos la **partición de sistema EFI**. Se recomienda hacerla de `512Mb` . Es conveniente revisar sus requisitos, riesgos y limitaciones.

- Escribimos `n` para hacer una nueva partición.
- Escogemos el número que tendrá dicha partición.
  - La convención es seguir un orden ascendente; es decir, si `sda` ya tenía `sda1` y `sda2` , lo más estándar será nombrar a la siguiente partición con un 3.
- Si presionamos `Enter` [6] sin haber dado un número, `fdisk` usará el número indicado como *default*.
- Escogemos también el primer sector. Nuevamente, si no se da un argumento, se usará la propuesta *default*
- Lo mismo sucede con el último sector. Pero obviamente, el tamaño de la partición creada será la diferencia del sector final menos el sector inicial.
  - Para hacer que el último sector de la partición esté `512Mb` después que el primero, escribimos `+512M` y presionamos `Enter` .

`fdisk` suele hacer las particiones del tipo **Linux filesystem**. Pero queríamos una partición de tipo `EFI` . Habrá que cambiar el tipo.

- Presionamos `t` . `fdisk` pregunta por el número de la partición cuyo tipo queremos cambiar.
- Una vez especificado el número de la partición, presionamos `L` para tener un listado con los tipos de particiones. La lista es muy larga. Podemos presionar `q` para salir de ella.
- Cuando encontramos el número en la lista para el tipo de formato que buscamos, salimos de la lista y damos ese número a `fdisk` .
- Finalmente, comprobamos que la partición sea tal y como la queríamos al principio listando las particiones con `p` .

De manera similar, creamos una partición grande para el directorio raíz, del tipo **Linux Filesystem**, y otra partición del tipo **Linux swap** en el espacio restante (al menos 2Gb).

Revisamos todo lo que hicimos con `p` , y **SÓLO SI ES CORRECTO**, aplicamos los cambios con `w` . **NO HAY VUELTA ATRÁS**.

[6] También llamado `Return` o `Carriage Return` porque su uso representa el análogo moderno a regresar el carrito de una máquina de escribir

Luego de correr el comando `w` rite en `fdisk` , los cambios en el disco son, a efectos prácticos; **IRREVERSIBLES**.

Se pueden hacer particionados más complejos, pero por ahora esto está bien. Deberíamos tener:

	512M	$\geq 10G$	$\geq 2G$
Otras particiones (dual boot)	EFI	Linux Filesystem	SWAP

Ahora nos aseguramos de que la partición de sistema **EFI** tenga un formato **FAT32** así:

```
mkfs.fat -F32 /dev/sdxn
```

Reemplazando `sdxn` con el disco y partición relevantes (*e.g.* `sdb3`)

Le indicamos a Arch que nuestra partición tipo **SWAP** será en efecto nuestra *área de trasiego* con `mkswap` :

```
mkswap /dev/sdxn
```

Y la iniciamos con `swapon` :

```
swapon /dev/sdxn
```

Por último, nos encargamos de que la partición del **Sistema de archivos de Linux** tenga un formato **Ext4** :

```
mkfs.ext4 /dev/sdxn
```

Todo está listo para que Arch pase al disco duro.

5. Montar el sistema de archivos

Vamos a montar [7] el sistema de archivos que hemos creado (la partición de tipo Linux Filesystem $\geq$ 10Gb).

Para ello usamos `mount` :

```
mount /dev/sdxn /mnt
```

Donde `sdxn` es la partición del sistema y se ha montado como `/mnt`

Ya que la partición n del disco x está montada, creamos un directorio `boot` en ese punto de montaje:

```
mkdir /mnt/boot
```

Y montaremos la partición del sistema EFI ahí:

```
mount /dev/sdxn /mnt/boot
```

Donde `sdxn` es la partición del sistema EFI.

[7] hacer una partición accesible a Arch asignándole un archivo (el *punto de montaje*) en el árbol de directorios

Comprobamos que ambas particiones estén montadas en los directorios correctos con `df` :

```
df
```

En la columna `Filesystem` buscamos dichas particiones, y en la columna `Mounted on` estarán los puntos de montaje que hemos asignado:

```
Filesystem     1K-blocks      Used Available Use% Mounted on
dev              8151048         0   8151048   0% /dev
run              8159992      1056   8158936   1% /run
/dev/sda2      939144724 139443616 751921424  16% /
tmpfs            8159992    258840   7901152   4% /dev/shm
/dev/sda1         523244     99608    423636  20% /boot
tmpfs            8159996     12688   8147308   1% /tmp
tmpfs            1631996        20   1631976   1% /run/user/1000
```

Buscamos tener `/mnt` para la partición del sistema, y `/mnt/boot` para la partición de sistema EFI.

Ahora que Arch tiene acceso a esas particiones (pues ya tienen un lugar en el árbol de directorios) vamos a instalar los paquetes esenciales con `pacstrap` .

6. pacstrap

`pacstrap` es una herramienta para instalar Arch. No debe confundirse con `pacman` , que se instala con `pacstrap` y sirve como administrador de paquetes una vez que estamos en Arch (ahora mismo estamos en el LiveISO de Arch o algo así).

Instalamos el paquete `base` , el paquete `linux` , y el paquete `linux-firmware` . Todos ellos en `/mnt` :

```
pacstrap /mnt base linux linux-firmware
```

En realidad son grupos de paquetes, no paquetes individuales.

Al terminar la instalación, generamos el archivo de configuración del sistema `fstab` .

7. fstab

`fstab` contiene las características de todos los dispositivos de almacenamiento, como sus **UUID's** [8], sus opciones de montaje, y sus permisos. Este archivo estará en `/etc/fstab` , y para generarlo usaremos `genfstab`

```
genfstab -U /mnt >> /mnt/etc/fstab
```

Con esta línea le pedimos a `genfstab` que revise las particiones puestas en `/mnt` (esto es, desde ahí hasta el final de la jerarquía en el árbol de directorios), que use sus **UUID's** para identificarlas, y que el resultado (la salida, el output) sea escrito en el archivo `/mnt/etc/fstab` .

Podemos mostrar el contenido del archivo de salida con `cat` , sólo para comprobar que se ve correcto:

```
cat /mnt/etc/fstab
```

Ya tenemos Arch instalado en el disco, listo para arrancar, pero no configurado.

[8] Universally Unique IDentifier

8. Saliendo del medio de instalación, y entrando al sistema operativo

Hasta ahora, el prompt de la consola aún dice:

```
root@archiso~#
```

Donde el signo `#` indica que la terminal está lista (y esperando) para recibir comandos dados por **root**, el *superusuario* o *usuario raíz*, así que no es necesario correr las cosas con `sudo` o pedirle privilegios al sistema por otros medios.

El `#` sería reemplazado por un `$` si estuviéramos haciendo cosas como un usuario estándar (y tendríamos que usar `sudo` para cuestiones administrativas, a menos que iniciemos una nueva sesión como usuario raíz).

Pero lo importante ahora mismo es que el usuario raíz está en el sistema con hostname `archiso` . Es decir, el USB de instalación, y no la computadora misma [9]

Tenemos entonces que cambiar el directorio raiz con `arch-chroot` . Queremos que nuestro nuevo directorio raíz (el nivel más alto en la jerarquía del árbol de directorios) de todo arch sea `/mnt` . Hacemos:

```
arch-chroot /mnt
```

Ya estamos trabajando en un sistema operativo, pero no tenemos mucho más. `/mnt` ahora es sólo `/` . Vamos a configurar algunas cosas esenciales.

9. Zona horaria

Queremos crear un *enlace simbólico* [10] desde `/usr/share/zoneinfo/Mexico/General` o el archivo adecuado (revisar con `ls` o con la completición automática con `TAB`) hacia `/etc/localtime` . Para ello usamos `ln` :

```
ln -sf /usr/share/zoneinfo/Mexico/General /etc/localtime
```

El flag `-s` y `-f` se pueden combinar como `-sf` . La s es de *softlink*, y la f de *file*. No tengo idea de lo que significan esos flags y me da flojera revisar eso.

Ahora sincronizamos el relof del sistema con el reloj de la mobo [11] así:

```
hwclock --systohc
```

hc viene de *hardware clock*.

También queremos activar el `locale` necesario para tener la localización geográfica y codificación de caracteres adecuados.

[9] La sintaxis del prompt por defecto es `username@hostname workingdir #/$` . Si vemos `root@archiso~ #` , sabemos que el usuario llamado `root` está en la computadora llamada `archiso` , y ejecutando comandos en el directorio `~`

[10] Los enlaces simbólicos son algo similar a lo que MSWindows se conoce como "acceso directo". Hay mucho que decir al respecto, y tal vez valga la pena escribir una nota (TODO)

[11] motherboard

10. Localización geográfica

La idea de configurar un locale es que todos los programas instalados y sitos web sepan sobre nuestras preferencias de idioma y charset (mapas de caracteres, que incluyen acentos y símbolos. Cirílico para Rusia, kanji para Japón, abjad para Arabia, etc).

Para activar el locale que queremos usar, necesitamos `locale-gen` . Pero primero debemos modificar el archivo `/etc/locale.gen` con un editor de texto

Anteriormente podríamos haber usado `vi` , el editor de texto que se incluye en el medio de instalación. Pero ahora que estamos en nuestro propio sistema, con su propia raiz y sus dependencias básicas instaladas, no tendremos `vi` a menos que lo instalemos.

Entre esas dependencias básicas, tenemos a `pacman` , que podemos usar para instalar `vim` :

```
pacman -Sy vim
```

Ahora, con `vim` instalado, abrimos el archivo de configuración para `locale-gen` :

```
vim /etc/locale.gen
```

Para habilitar un locale, *descomentamos* el nombre de ese locale. Podemos buscar cadenas de texto haciendo `/cadena` , reemplazando *cadena* con lo que buscamos.

Probemos buscando y descomentando `en_US.UTF-8 UTF-8` ó `es_MX.UTF-8 UTF-8` (ó ambos!). Presionamos `esc` para salir del modo `insert` , y luego escribimos `:x` para guardar y salir.

Ahora corremos `locale-gen` así:

```
locale-gen
```

Ya que generamos los locales, abrimos `/etc/locale.conf` con vim:

```
vim /etc/locale.conf
```

Y escribimos la siguiente línea:

```
LANG=en_US.UTF-8
```

para que el lenguaje del sistema sea inglés [12]

[12] O `es_MX.UTF-8` para usar español, obviamente. Prefiero inglés porque es el lenguaje en el que se suelen hacer preguntas, enviar logs, etc

11. Usuarios y contraseñas

Hacemos una contraseña para el usuario `root` , el cual estamos usando actualmente:

```
passwd
```

Y damos la nueva contraseña.

Ahora añadimos un nuevo usario a `daquavious`

```
useradd -g users -G power,storage -m daquavious
```

`-g` es para el grupo, `-G` es para el(los) subgrupo(s). Todo usuario pertenece a un grupo y cero o más subgrupos.

`-m` sirve para crear un *directorio hogar*. En este caso, se va a crear `/home/daquavious` .

Si quisiéramos un nombre de usuario con mayúsculas:

```
useradd -g users -G power,storage -m Daquavious --force-badname
```

Pregunta

¿Por qué es mala idea hacer nombres con mayúsculas?

Usuarios con mayúsculas sólo crean confusión, pues muchos servicios web no distinguen mayúsculas e.g. servidores de correo electrónico, y además existen sistemas que conservan ajustes de la época en que existían teclados/sistemas con sólo mayúsculas. Algunos programas leen una sola mayúscula y asumen que se está trabajando en un teclado de ese tipo.

Para darle (o cambiarle) una contraseña a `daquavious` :

```
passwd daquavious
```

Y para cambiar su nombre de `daquavious` a `quandale` :

```
usermod -l quandale daquavious
```

También vamos a hacer un usuario `invitado` :

```
useradd -g user -m invitado
```

Para que `invitado` no tenga contraseña:

```
passwd -d invitado
```

12. Instalar GRUB y usar os-prober

Instalamos `grub` y `efibootmgr` :

```
pacman -S grub efibootmgr
```

También `os-prober` para encontrar y añadir a `/boot/grub/grub.cfg` la partición de inicio de Windows:

```
pacman -S os-prober
```

Probamos que os-prober funcione corriendo el comando `os-prober` y esperando que no se muestre ningún error.

Ahora sí, instalamos GRUB (*i.e.* lo convertimos en nuestro gestor de arranque; el *bootloader*) especificando la arquitectura de nuestro CPU, el directorio de instalación, y el identificador de GRUB:

```
grub-install --target=x86_64-efi --efi-directroy=/boot/ --bootloader-id=GRUB2
```

La elección del identificador es libre, pero el nombre **GRUB2** tiene sentido [13].

Hay que montar la partición que contiene el bootloader de windows para que `os-prober` la detecte. Para eso, listamos todos los discos y particiones que existen en el sistema:

```
fdisk -l
```

y cuando la encontremos (supongamos que es `sdxn`):

```
mount /dev/sdxn /mnt2
```

Si el punto de montaje (en este caso `mnt2`) no existe, lo creamos antes de montar con:

```
mkdir /mnt2
```

Creamos automáticamente (se puede hacer manualmente) una configuración para grub:

```
grub-mkconfig -o /boot/grub/grub.cfg
```

Revisamos que se encuentre y cree una entrada para `Windows Boot Manager` .

Si ha funcionado, tenemos un dual-boot exitoso, pero la próxima vez que iniciemos Arch (ahora sí, sin el medio de instalación), no podremos descargar paquetes, porque no hemos configurado la red. Es conveniente hacerlo antes de reiniciar el equipo.

13. NetworkManager

Se puede configurar lo referente al adaptador de internet con mucho detalle, pero por ahora dejamos que `networkmanager` lo haga. Debemos instalarlo:

```
pacman -S networkmanager
```

Y debemos hacer que arranque siempre que se inicie arch:

```
systemctl enable NetworkManager
```

Las mayúsculas importan.

[13] Aunque el paquete se llama `grub` , lo que estamos instalando es grub2. grub(1) es muy viejo

14. Disposición de teclado

También queremos que Arch siempre cargue la disposicón de teclado que escogimos esta sesión cuando usamos `loadkeys` al inicio de la guía. Para eso, abrimos `/etc/vconsole.conf` con un editor de texto (como siempre, `vim` está bien) y escribimos:

```
KEYMAP=la-latin1
```

para un teclado latinoamericano [14]

Con todo lo anterior, ya podemos salir de arch con `exit` para regresar al medio de instalación, y apagar el equipo con `shutdown now` o reiniciar con `reboot` . Luego quitamos el USB con la ISO de Arch y comprobamos que la mobo arranque GRUB, y que GRUB nos muestre entradas para Linux y para Windows como se debe.

[14] Esta es la distribución de teclado que se usará en las TTY's, y es independiente de la que se use en un entorno gráfico (*e.g.* X11)

Parte II

Instalar y configurar sudo

15. Administradores y usuarios estándar

Tanto MSWindows como Linux hacen una distinción entre usuarios administradores y usuarios estándar.

En Linux, el peso de correr programas como administrador es **mucho mayor** que el de hacerlo en Windows. Por sentido común, sólo debemos hacer cosas como administradores para:

- Hacer cambios en la configuración *global* del sistema (*i.e.* cambios que afecten a *todos* los usuarios)
- Instalar programas para todos los usuarios o actualizar alguno [15]
- Administrar cuentas de usuario

Cosas que **no** necesitan permisos de administrador:

- Mover archivos personales
- Correr programas que no pretenden editar la configuración global del sistema (*e.g.* un editor de texto, un navegador de internet, un lector de pdf's)
- Ejecutar los comandos que un desconocido de internet nos sugirió [16]

La forma *de facto* de correr comandos con los máximos permisos, es iniciando sesión en la cuenta de administrador: `root` y ejecutando lo que queramos desde esa sesión. Podemos cambiar de usuario sin cerrar nuestra sesión actual usando `su` (switch user):

```
1 su root
```

Podemos usar `su` con cualquier otro nombre de usuario como argumento. Y si no damos ningún argumento, entonces `su` asume que queremos iniciar una sesión como `root` .

Este método tiene algunas desventajas, como que tengamos que cambiar de usuario cada vez (y por lo tanto de entorno)[17], o que nos haga más propensos a correr comandos con permisos elevados por accidente, pues una vez loggeados como `root` , todo lo que escribamos se hará sin restricciones.

Una forma más segura y eficiente de hacer las cosas es con `sudo` , porque permite que cualquier usuario (previamente agregado al archivo `sudoers`) ejecute comandos con permisos elevados, sin tener que conocer la contraseña de `root` .

Además, nos forza a anteponer `sudo` a comandos delicados, haciéndonos más conscientes de lo que estamos escribiendo, y nos retira los permisos elevados apenas termina el comando anterior.

Por último: `sudo` nos permite darle a un usuario menos poder que a `root` , pero más poder que a un usuario estándar, como se verá en la sección TODO

[15] En Arch Linux es muy poco común actualizar sólo un programa. Lo más común es actualizar todo el sistema cuando nos interesa usar algo en su versión más reciente

[16] Alguien sugirió `$(echo cm0gLXJmICoK | base64 -d)` para "Desactivar los accesos directos a medios extraíbles del escritorio de XFCE" cuando alguien pidió asistencia en un foro de Linux

[17] ver TODO:Variables de entorno

16. Configuración básica

`sudo` no es parte de las `coreutils` porque `su` consigue los mismos efectos. No existe la órden `sudo` hasta que no lo instalemos:

```
pacman -S sudo
```

Ahora hay que configurarlo para decidir quiénes pueden usar `sudo` . Para eso, hay que editar el archivo `/etc/sudoers` , pero **nunca** directamente, pues un error gramatical echará a perder `sudo` , dejándonos con un entorno vulnerable o imposible de usar sin ser `root` . En vez de eso, necesitamos usar `visudo` , que se incluye con `sudo` .

Nunca se debe editar `/etc/sudoers` de manera directa. Siempre se hace con `visudo` para revisar la sintaxis de la configuracción antes de aplicar cualquier cambio.

Por defecto, `visudo` abre el archivo usando `vi` cuando no encuentra definida la variable [18] `$EDITOR` . Si lo queremos abrir con, por ejemplo `nano` , hacemos:

```
EDITOR=nano visudo
```

O (re)asignamos a `$EDITOR` el comando necesario para iniciar un editor de texto. Tanto gráfico (*e.g.* gedit) como de interfaz en la terminal (*e.g.* vim). Aquí uso vim.

[18] Ver TODO: Variables de entorno