

## Nodo Bitcoin

Para participar de la red **Bitcoin** como se desarrolló en **Infraestructura para Bitcoin** es necesario disponer de un ordenador con un cliente Bitcoin-Core.

Se diferencian dos clases de participantes en la red de Bitcoin:

1. Usuarios regulares que mediante un nodo (propio o de un tercero) interactúan realizando transacciones.
2. Mineros que adicional al nodo tienen un poder de hash para resolver la prueba de trabajo POW. Esto les genera un beneficio en Bitcoin (un premio por producción de bloque más las comisiones).

Mantener un nodo no es minar. No tiene ningún beneficio monetario otorgado por la Red. Encontramos su necesidad por otros tipo de razones:

- ▷ Contribuir en la red Bitcoin dando soporte a su descentralización.
- ▷ Verificar propietariamente la información de todo el historial. No solo se descarga la base de datos sino que se verifica que sea correcta.
- ▷ Privacidad. Si se desea realizar una transacción o consultar un historial se recomienda hacerlo sin intermediarios. Esto para no ceder datos personales.
- ▷ Realizar análisis de data sin pagar suscripciones o confiar en bases de datos de terceros. Obtener información directamente de la Red Bitcoin.

Es por esta razón que en este manual mostraremos como montar un nodo completo de Bitcoin que tenga todas las medidas recomendadas de seguridad y privacidad.

## Consideraciones Hardware

Se detalla el equipo que se usa:

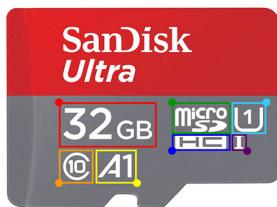
1. Raspberrypi4 4 Gb Ram.
2. Disco duro externo HDD 1 Tera.



Adicionalmente tomamos en consideración los siguientes puntos:

- ▶ El RaspberryPi arranca el sistema operativo desde una tarjeta SD. Se recomienda una memoria mayor a 16 Gb que soporte mas de 10 MB/s de lectura/escritura.

### LECTURA DE UNA TARJETA SD



- 1: Capacidad de almacenamiento
- 2: Clase 10 de velocidad mínima (10MB/s)
- 3: Certificado de velocidad mínima constante tipo A1
- 4: Formato de tarjeta Micro SD
- 5: Velocidad Ultra High Speed tipo U1 (10MB/s)
- 6: versión de tarjeta SD High Capacity
- 7: Interfaz de Ultra High Speed I (entre 50MB/s y 104MB/s)

- ▶ El Raspberry puede arrancar con una fuente de 700 mA en nuestro caso el necesitar conectar un disco duro externo por USB hace

necesaria una mayor alimentación para no tener cortes y posibles fallas. Lo recomendable es disponer una fuente de alimentación que entregue 5 V y más de 2 A.

- ▶ **Disco Duro Externo.** El disco duro puede ser HDD (con USB 3.0) con una velocidad de lectura/escritura mayor a 50 MB/sec para tener algo aceptable (se puede verificar como veremos más adelante). Para optimizar el performance es mejor usar un disco de estado sólido SSD por su mayor velocidad de lectura/escritura.
- ▶ **Conexión a Internet.** Se usa una conexión mediante cable Ethernet para tener mayor estabilidad.

Para optimizar los recursos limitados del Raspberry instalaremos un sistema operativo LITE sin modo gráfico. Instalaremos uno a uno los paquetes a usar evitando software innecesario (la versión Desktop viene con varias herramientas que mejoran la experiencia del usuario pero consumen muchos recursos) y procuramos no usar clientes Bitcoin-core como Umbrel, myNode u otros. Se verifica e instala el cliente Bitcoin-Core desde una fuente pública.

## Pre Instalación

En la página del proyecto Oficial de [Raspberrypi4](#) se puede descargar el software necesario para instalar y configurarlo desde otra PC.

Descargamos dos archivos:

1. Raspberry Pi Imager. <https://www.raspberrypi.com/software/>  
Este programa copia el sistema operativo a la tarjeta SD para usarlo luego en la tarjeta.

**Install Raspberry Pi OS using Raspberry Pi Imager**

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi. [Watch our 45-second video](#) to learn how to install an operating system using Raspberry Pi Imager.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

[Download for Windows](#)

2. El sistema operativo Raspbian.

<https://www.raspberrypi.com/software/operating-systems/>  
Buscamos el sistema de 64 bits Lite.

## Raspberry Pi OS Lite

Release date: September 22nd 2022

System: 64-bit

Kernel version: 5.15

Debian version: 11 (bullseye)

Size: 289MB

[Show SHA256 file integrity hash:](#)

[Release notes](#)

[Download](#)

[Download torrent](#)

[Archive](#)

Con el sistema operativo que descargamos debemos verificar que coincide con la firma SHA256 que el desarrollador entrega. En el caso del ejemplo es:



72c773781a0a57160eb3fa8bb2a927642fe60c3af62bc980827057bcecb7  
b98b

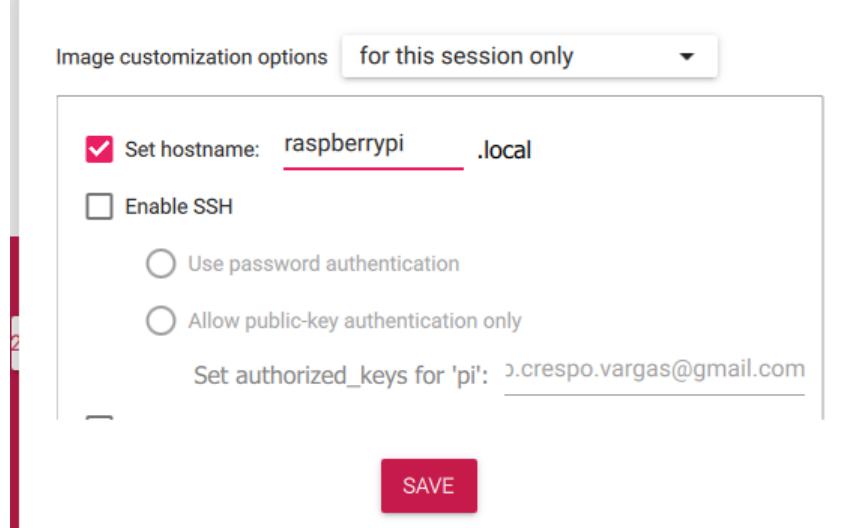
Este procedimiento se hace para asegurarnos que el sistema operativo no ha sido manipulado por un tercero malicioso. En la terminal aplicamos el siguiente comando.

```
~/Desktop  
$ sha256sum 2022-09-22-rpios-bullseye-arm64-lite.img.xz  
72c773781a0a57160eb3fa8bb2a927642fe60c3af62bc980827057bcecb7b98b *2022-09-22-ras  
pios-bullseye-arm64-lite.img.xz
```

Podemos verificar que la firma obtenida coincide con la firma que nos da el fabricante .

## Raspberry Pi Imager

Una vez instalado se configura el boot en el SD cuidando los siguientes detalles:



1. Nombre. Para acceder remotamente facilita el acceso.
2. Enable SSH. Para acceder a una terminal por ssh (evita el uso de un teclado y pantalla) y controlar remotamente el dispositivo.
3. Conexión a Internet. En el caso del nodo es cableado pero de usarse Wifi se puede configurar antes de iniciar el sistema.

Una vez culminado el proceso de escritura del sd, se conecta al Raspberry y esta listo para arrancar.

## Configuración Post Instalación

Una vez conectado el Raspberry se accede remotamente usando SSH. En una terminal [Linux](#) accedemos con el siguiente comando. (En windows se puede usar el bash de git o PuTTY).

```
ssh user@raspberry.local
```

SHELL Copiar

Una vez se introduce la contraseña tenemos la siguiente pantalla.

```
Warning: Permanently added the ECDSA host key for IP address '192.168.1.6' to the list of known hosts.  
ghost@onepi.local's password:  
Linux onepi 5.15.61-v8+ #1579 SMP PREEMPT Fri Aug 26 11:16:44 BST 2022  
aarch64  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/*copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Wed Oct 26 22:32:28 2022 from fe80::4555:11ce:f2fe:e2ac%wl  
an0  
ghost@onepi:~ $
```

El primer paso es actualizar el sistema con los comandos.

```
sudo apt update && sudo apt upgrade
```

SHELL Copiar

Instalamos algunas dependencias que usaremos para desplegar el repositorio principal.

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

SHELL Copiar

Por seguridad generamos claves ssh útil para conectarnos con repositorios como Github.

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

SHELL Copiar

Generando el archivo id\_rsa.pub en la dirección `~/ssh` .

Hasta este punto tenemos una configuración básica para un Raspberry Pi4. Tanto para desplegar un Nodo con Bitcoin-Core como para usar en análisis de datos o automatismos con Bots.

## Preparando Nodo Bitcoin

Una vez que el Raspberry Pi esta listo para funcionar por primera vez conectamos el disco duro por un puerto USB 3.0 (azul) disponible.

Al darle energía toma un par de minutos en arrancar completamente y estar disponible.

## Ping al Nodo

Para constatar que el nodo esta funcionando correctamente y esta conectado a Internet se puede hacer un ping desde otro punto de la red. Para lo cual se necesita saber que IP tiene asignado el Nodo o convenientemente usar el alias 'raspberry.local' que se configuró en la Pre Instalación.

Usando el comando ping se puede saber si el Nodo esta conectado y funcionando.

```
ping -c raspberry.local
```

SHELL Copiar

Se muestra la salida al usar ping cuando el Nodo esta funcionando correctamente y cuando deja de estar operando.

```
gh pi:~ $ ping -c 5 192.168.1.33
PING 192.168.1.33 (192.168.1.33) 56(84) bytes of data.
64 bytes from 192.168.1.33: icmp_seq=1 ttl=64 time=35.4 ms
64 bytes from 192.168.1.33: icmp_seq=2 ttl=64 time=6.29 ms
64 bytes from 192.168.1.33: icmp_seq=3 ttl=64 time=17.6 ms
64 bytes from 192.168.1.33: icmp_seq=4 ttl=64 time=7.81 ms
64 bytes from 192.168.1.33: icmp_seq=5 ttl=64 time=6.97 ms

--- 192.168.1.33 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 6.289/14.812/35.417/11.093 ms
gh pi:~ $ ping -c 5 192.168.1.33
PING 192.168.1.33 (192.168.1.33) 56(84) bytes of data.

--- 192.168.1.33 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4075ms
```

Una vez se accede por SSH se instala los siguientes paquetes.

```
sudo apt install wget curl gpg git --install-recommends
```

SHELL Copiar

## Medimos el Performance

Para asegurarnos que tendremos un buen funcionamiento analizamos el rendimiento del disco duro externo. Instalamos el paquete 'hdparm'.

```
sudo apt install hdparm
```

SHELL Copiar

Para ver si el disco externo a sido montado correctamente en el sistema usamos el comando 'lsblk'

```
lsblk -pli
```

SHELL Copiar

Retornando la dirección del punto donde el disco externo fue montado.

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINT
/dev/sda	8:0	0	931.5G	0	disk	
/dev/sda1	8:1	0	931.5G	0	part	
/dev/mmcblk0	179:0	0	29G	0	disk	
/dev/mmcblk0p1	179:1	0	256M	0	part	/boot
/dev/mmcblk0p2	179:2	0	28.7G	0	part	/

Con la dirección del disco podemos medir la velocidad de lectura/escritura

```
sudo hdparm -t --direct /dev/sda
```

SHELL Copiar

```
/dev/sda1:  
Timing O_DIRECT disk reads: 306 MB in 3.00 seconds = 101.86 MB/sec
```

Si la velocidad medida es mayor a > 50 MB/sec es suficiente.

Si la velocidad en USB 3.0 no llega a ser aceptable se recomienda revisar drivers, conectores y puertos.

## Directorio de Almacenamiento

El almacenamiento de la base de datos del Blockchain se guarda en un directorio dedicado `/data/` que debe estar en un lugar distinto de `/home/`, precisamente en el punto donde se monta el disco externo.

Por lo general en un sistema Linux el montaje del disco externo es automático. Para hacerlo manualmente primero vemos que se ha reconocido el periférico.

```
sudo fdisk -l
```

SHELL Copiar

Una vez encontrado el disco duro y su dirección (en el caso del ejemplo `/dev/sda1`) se crea una carpeta para que el disco sea montado.

```
sudo mkdir /media/blockchain  
sudo mount /dev/sda1 /media/blockchain
```

SHELL Copiar

Para que este montaje sea automático con cada arranque del sistema (cómo cuando se reinicia inesperadamente por un corte de luz) usamos `fstab`.

Necesitamos el ID que tiene el disco externo, lo obtenemos con el siguiente comando.

```
sudo blkid
```

SHELL Copiar

Se copia la variable `PARTUUID` del disco y editamos el archivo de configuración `sudo nano /etc/fstab` y agregamos la siguiente línea

```
PARTUUID=b47b5397-0670-41a3-b38f-2bc64dd2cc18 /media/blockchain ext4  
user,errors=remount-ro,auto,exec,rw
```

Copiar

Al reiniciar el Nodo el montaje es automático y se puede constatar con el comando `sudo lsblk -pli`

En el punto de montaje `/media/blockchain` es que se crea la carpeta `/data/` donde se almacenará todo el Blockchain. Se le otorgan permisos de propiedad adecuados.

```
sudo mkdir data  
sudo chown admin:admin data/
```

SHELL Copiar

## Configuración SWAP

En los sistemas Linux además de la memoria RAM es usual que se asigne un espacio extra del disco duro como memoria `swap`. Este tipo de memoria se asigna como una especie de memoria RAM extra (no tan rápida). En el RaspberryPi esta configurada por defecto. Para incrementar la velocidad y esencialmente la estabilidad del sistema es que dejamos sin efecto esta asignación.

Editamos el archivo.

```
sudo nano /etc/dphys-swapfile
```

Copiar

Comentando la siguiente línea únicamente.

```
# comment or delete the CONF_SWAPSIZE line. It will then be created
dynamically
#CONF_SWAPSIZE=100
```

Copiar

Finalmente reiniciamos la configuración y el servicio para que tenga efecto.

```
sudo dphys-swapfile install
sudo systemctl restart dphys-swapfile.service
```

Copiar

## Seguridad

### Acceso con llaves SSH

En la configuración por defecto el acceso a terminal por SSH solicita un password. Cualquier persona que conozca esta contraseña puede acceder a una terminal siendo una brecha de seguridad.

Una mejor opción es deshabilitar el acceso por password y que únicamente se pueda acceder si se tiene un certificado SSH. Para esto se debe copiar la llave pública generada al Raspberry Pi.

Desde la dirección `/ssh/` copiamos la llave pública al Raspberry.

```
ssh-copy-id admin@raspberry.local
```

SHELL Copiar

Finalmente deshabilitamos que se pueda acceder por SSH mediante contraseña.

```
sudo nano /etc/ssh/sshd_config
```

SHELL Copiar

Cambiamos las siguientes líneas.

```
PasswordAuthentication no
ChallengeResponseAuthentication no
```

Copiar

Finalmente reiniciamos el servicio ssh.

```
sudo systemctl restart sshd  
exit
```

SHELL Copiar

## Firewall UFW

Un firewall controla que tipo de tráfico desde afuera acepta el ordenador y que aplicaciones tienen la salida de datos hacia afuera.

Muchas redes tienen los puertos disponibles por default para distintas entradas en nuevas conexiones. Inhabilitar puertos innecesarios cierra brechas de seguridad considerando que Bitcoin-Core se conecta mediante la red de TOR no necesita ningún puerto de entrada extra.

Usamos un firewall amigable en su configuración.

```
sudo apt install ufw  
sudo ufw default deny incoming  
sudo ufw default allow outgoing  
sudo ufw allow ssh  
sudo ufw logging off  
sudo ufw enable
```

SHELL Copiar

Se habilita a UFW para que inicie automáticamente con cada inicio de sistema.

```
sudo systemctl enable ufw
```

SHELL Copiar

Se puede consultar el estado con el siguiente comando.

```
sudo ufw status
```

SHELL Copiar

En el caso del nodo tenemos esta respuesta.

```
c:~ $ sudo ufw status  
Status: active  
  
To                         Action      From  
--                         ----      ---  
22/tcp                      ALLOW      Anywhere  
22/tcp (v6)                 ALLOW      Anywhere (v6)
```

## Fail2ban

El ingreso por SSH mediante contraseña se ha deshabilitado, pero suponiendo que el atacante logra tener acceso a la llave privada necesita la contraseña para tener acceso. Esta se puede obtener por fuerza bruta y para evitar este ataque, este software banea los IPs donde tenga una contraseña inválida una cantidad 'x' de veces por una cantidad 'x' de tiempo.

Basta con instalar para que fail2ban se active.

```
sudo apt install fail2ban
```

SHELL Copiar

#### Open File Limit

El nodo puede ser inundado de request (honestos o maliciosos) y dar un error del tipo `can't accept connection: too many open file`. Esto se soluciona aumentando el límite dado por default. Se crea un archivo en la dirección `/etc/security/limits.d/90-limits.conf` con las siguientes líneas.

```
*      soft  nofile 128000
*      hard  nofile 128000
root  soft  nofile 128000
root  hard  nofile 128000
```

Copiar

Y aumentamos la siguiente línea antes del comentario final en los archivos siguientes.

```
sudo nano /etc/pam.d/common-session
```

Copiar

```
session required          pam_limits.so
```

```
sudo nano /etc/pam.d/common-session-noninteractive
```

Copiar

```
session required          pam_limits.so
```

#### NGINX Reverse Proxy

NGINX es un servidor web open source. Mucha data requerida al nodo estará expuesta por el puerto de comunicación como por ejemplo el

explorador de bloques. Si bien estos servicios estarán en la red local es buena práctica que estén encriptados. NGINX es un servidor completo, pero solo se lo aplicará para encriptar la comunicación con SSL/TLS. Esta configuración se llama 'reverse proxy'. Instalamos NGINX.

```
sudo apt install nginx
```

SHELL Copiar

Creamos un certificado SSL/TLS (válido por 10 años)

```
sudo openssl req -x509 -nodes -newkey rsa:4096 -keyout /etc/ssl/private/nginx-selfsigned.key -out /etc/ssl/certs/nginx-selfsigned.crt -subj "/CN=localhost" -days 3650
```

Copiar

Editamos la configuración en `nginx.conf`

```
$ sudo mv /etc/nginx/nginx.conf /etc/nginx/nginx.conf.bak  
$ sudo nano /etc/nginx/nginx.conf
```

Copiar

```
user www-data;  
worker_processes 1;  
pid /run/nginx.pid;  
include /etc/nginx/modules-enabled/*.conf;  
  
events {  
    worker_connections 768;  
}  
  
stream {  
    ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;  
    ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;  
    ssl_session_cache shared:SSL:1m;  
    ssl_session_timeout 4h;  
    ssl_protocols TLSv1.2 TLSv1.3;  
    ssl_prefer_server_ciphers on;  
  
    include /etc/nginx/streams-enabled/*.conf;  
}
```

Copiar

Creamos un nuevo directorio para futuras configuraciones.

```
sudo mkdir /etc/nginx/streams-enabled
```

Copiar

Testeamos la configuración.

```
sudo nginx -t
> nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
> nginx: configuration file /etc/nginx/nginx.conf test is successfu
```

Copiar

#### Desactivar conexiones inalámbricas

El Raspberry Pi 4 tiene soporte para conexiones Wifi y Bluetooth. Deshabilitamos estas pues el nodo tendrá conexión a Internet por cable ethernet.

```
sudo nano /boot/config.txt
```

SHELL Copiar

Añadimos las líneas. Se guarda y al siguiente reinicio están desactivadas.

```
dtoverlay=disable-bt
dtoverlay=disable-wifi
```

Copiar

#### Privacidad usando TOR

Al correr un Nodo Bitcoin se expone una dirección IP (como un servidor). El tráfico asociado a este IP es visible para el proveedor del servicio de Internet ISP. Para tener privacidad se recomienda usar un VPN cifrado y evitar exponer el IP directamente. Otra manera es usar TOR project para anonimizar el tráfico y ocultar la dirección IP.

Se instala con el comando.

```
sudo apt install apt-transport-https
```

Copiar

Creamos un archivo llamado `tor.list` y agregamos las siguientes líneas.

```
sudo nano /etc/apt/sources.list.d/tor.list
```

SHELL Copiar

```
deb [arch=arm64 signed-by=/usr/share/keyrings/tor-archive-
keyring.gpg] https://deb.torproject.org/torproject.org bullseye main
```

Copiar

```
deb-src [arch=arm64 signed-by=/usr/share/keyrings/tor-archive-keyring.gpg] https://deb.torproject.org/torproject.org bullseye main
```

Se accede a `root` para añadir las llaves gpg.

```
sudo su
wget -qO-
https://deb.torproject.org/torproject.org/A3C4F0F979CAA22CDBA8F512EE8CBC
9E886DDD89.asc | gpg --dearmor | tee /usr/share/keyrings/tor-archive-
keyring.gpg >/dev/null
exit
```

Instalamos el keyring

```
sudo apt update
sudo apt install tor deb.torproject.org-keyring
```

Podemos verificar que Tor se instaló correctamente con el comando.

```
tor --version
> Tor version 0.4.7.10.
```

El servicio de Tor debe escuchar por default por los puertos `9050` y `9051`.

Para habilitar el `9051` configuramos la linea en el archivo `/etc/tor/torrc`

```
ControlPort 9051
```

Reiniciamos el servicio

```
sudo service tor restart
```

Y constatamos que esta correctamente configurado.

```
sudo ss -tulpn | grep tor | grep LISTEN
```

Con una salida esperada.

```
c:~ $ sudo ss -tulpn | grep tor | grep LISTEN
tcp  LISTEN 0      4096          127.0.0.1:9050        0.0.0.0:*
pid=12743,fd=6)
tcp  LISTEN 0      4096          127.0.0.1:9051        0.0.0.0:*
pid=12743,fd=7)
```

## Configurando TOR

Bitcoin-Core se comunica directamente con el demonio Tor para enrutar todo el tráfico mediante esa red. Necesitamos habilitar a Tor para que acepte instrucciones mediante el puerto de control con su propia autenticación.

Modificamos el archivo `/etc/tor/torrc` removiendo el # de las siguientes líneas.

```
# uncomment:
ControlPort 9051
CookieAuthentication 1

# add:
CookieAuthFileGroupReadable 1
```

Copiar

Recargamos el servicio Tor para activar las modificaciones.

```
sudo systemctl reload tor
```

SHELL Copiar

Revisamos `systemd journal` para ver la salida de logs en tiempo real.

```
sudo journalctl -f -u tor@defualt
```

SHELL Copiar

## SSH con TOR

Una forma de controlar el nodo mediante una terminal estando en cualquier lugar del mundo 🌎 es usando un servicio oculto SSH mediante TOR.

## Servidor SSH usando TOR

Se habilita el servicio en el archivo `torrc`

```
sudo nano /etc/tor/torrc
```

SHELL Copiar

Modificando las siguientes líneas en la sección "location-hidden services",

```
##### This section is just for location-hidden services #### Copiar
# Hidden Service SSH server
HiddenServiceDir /var/lib/tor/hidden_service_sshd/
HiddenServiceVersion 3
HiddenServicePort 22 127.0.0.1:22
```

Se recarga el servicio Tor y se toma nota de la dirección que se genera.

```
$ sudo systemctl reload tor
$ sudo cat /var/lib/tor/hidden_service_sshd/hostname
> abcdefg.....xyz.onion
```

Copiar

## Cliente SSH

Desde otro ordenador instalamos netcat

```
sudo apt install netcat tor
```

SHELL Copiar

y accedemos sabiendo el host (link .onion) previamente dado.

```
ssh -o "ProxyCommand /usr/bin/nc -X 5 -x 127.0.0.1:9050 %h %p"
admin@abcd.....xyz.onion
```

Copiar

Si el puerto 9050 esta ocupado, Tor usa el puerto 9150 para el servicio ssh.

## Instalando Bitcoin-Core



El cliente Bitcoin-core descarga el blockchain completo, valida cada transacción desde el primero en 2009 hasta el último (cada 10 min sale un nuevo bloque). La palabra 'validar' hace referencia a que revisa y constata que cada bloque es correcto mediante su firma hash y la dificultad de la prueba de trabajo POW. Si alguien intenta hacer pasar un bloque malicioso como verdadero el sistema revela que existe manipulación (tamper evident) y el nodo al verificar que es falso lo descarta.

El nodo siempre toma como verdadera la cadena con bloques válidos más larga. Por lo que es virtualmente imposible hacer modificaciones. Esta característica se llama **tamper-proof** haciendo de las transacciones seguras e inmutables.

## Instalando el Core

Se descarga la última versión del binario del Bitcoin-Core y se verifica que no ha sido manipulado.

En una carpeta `/tmp/` descargamos la última versión disponible en: <https://bitcoincore.org/en/download/> (ARM Linux 64 bits), como también las firmas criptográficas.

```
# Bitcoin Core binary
wget https://bitcoincore.org/bin/bitcoin-core-24.0.1/bitcoin-24.0.1-
aarch64-linux-gnu.tar.gz

# cryptographic checksum
wget https://bitcoincore.org/bin/bitcoin-core-24.0.1/SHA256SUMS

# signatures checksums
wget https://bitcoincore.org/bin/bitcoin-core-4.0.1/SHA256SUMS.asc

# signature timestamp ots
wget https://bitcoincore.org/bin/bitcoin-core-24.0.1/SHA256SUMS.ots
```

## Verificación Hash

Con el `SHA256SUMS` constatamos que el Core tenga la misma firma.

```
sha256sum --ignore-missing --check SHA256SUMS
> bitcoin-24.0.1-aarch64-linux-gnu.tar.gz: OK
```

## Verificación de Firmas

Bitcoin-Core se lanza en cada versión con la firma PGP de un número de desarrolladores donde cada uno usa su firma. Para verificar y validar cada una de estas firmas, primero descargamos las llaves públicas.

```
wget https://raw.githubusercontent.com/bitcoin/bitcoin/master/contrib/builder-keys/keys.txt
while read fingerprint keyholder_name; do gpg --keyserver https://keyserver.ubuntu.com --recv-keys ${fingerprint}; done < ./keys.txt
```

Copiar

Importamos una a una las firmas GPG.

```
gpg: key 7588242FBE38D3A8: public key "Gavin Andresen <gavinandresen@gmail.com>" imported
gpg: Total number processed: 1
gpg:          imported: 1
gpg: key 8F617F1200A6D25C: public key "Gloria Zhao <gloriazhao@berkeley.edu>" imported
gpg: Total number processed: 1
gpg:          imported: 1
gpg: key 410108112E7EA81F: public key "Hennadii Stepanov (GitHub key) <3296351-hebasto@users.noreply.github.com>" imported
gpg: Total number processed: 1
gpg:          imported: 1
gpg: keyserver receive failed: No data
gpg: key 25F27A38A47AD566: public key "James O'Beirne <james.obeirne@pm.me>" imported
gpg: Total number processed: 1
gpg:          imported: 1
gpg: key 2DA9C5A7FA81EA35: public key "Jarol Rodriguez <jarolrod@tutanota.com>" imported
gpg: Total number processed: 1
gpg:          imported: 1
gpg: key 535B12980BB8A4D3: public key "Jeffri H Frontz <jeff.frontz@gmail.com>" imported
gpg: Total number processed: 1
gpg:          imported: 1
gpg: key C5242A1AB3936517: public key "jl2012 <jl2012@xbt.hk>" imported
gpg: Total number processed: 1
gpg:          imported: 1
gpg: key 796C4109063D4EAF: public key "Jon Atack <jon@atack.com>" imported
gpg: Total number processed: 1
gpg:          imported: 1
gpg: key 29D4BCB6416F53EC: public key "Jonas Schnelli <dev@jonasschnelli.ch>" imported
gpg: Total number processed: 1
```

Se verifican las firmas con el archivo **SHA256SUMS.asc** poniendo énfasis en el siguiente texto.

```
> gpg: Good signature from ...
> Primary key fingerprint: ...
```

Copiar

Para mayor referencia de como funcionan las firmas PGP tenemos el apunte [GNU GPG](#).

### Verificación Timestamp

El binario `checksum` es un archivo que tiene un timestamp en el Blockchain de Bitcoin. Usando el protocolo [OpenTimestamps](#) se verifica que el archivo ya existía previamente a un punto en el tiempo (a un bloque en Bitcoin).



On your local computer, download the checksums file and its timestamp proof:

- <https://bitcoincore.org/bin/bitcoin-core-24.0.1/SHA256SUMS>
- <https://bitcoincore.org/bin/bitcoin-core-24.0.1/SHA256SUMS.ots>

- ▷ In your browser, open the [OpenTimestamps](#) website
- ▷ In the “Stamp and verify” section, drop or upload the downloaded SHA256SUMS.ots proof file in the dotted box
- ▷ In the next box, drop or upload the SHA256SUMS file
- ▷ If the timestamps is verified, you should see the following message. The timestamp proves that the checksums file existed on the [release date](#) of Bitcoin Core v24.0.1

Una vez verificado de que el core es legítimo y seguro mediante checksum, firmas digitales y el timestamp procedemos a Instalarlo.

### Instando el Core

Se descomprime el Core y se procede a instalarlo.

```
tar -xvf bitcoin-24.0.1-aarch64-linux-gnu.tar.gz
sudo install -m 0755 -o root -g root -t /usr/local/bin bitcoin-
24.0.1/bin/*
```

SHELL Copiar

El comando `tar` descomprime los archivos, los flags `-xvf` son de: x (descomprimir), v (verbose, imprima en pantalla las acciones) y f (file).

`install` copia los archivos de `/bitcoin-24.0.1/bin/*` a una ubicación `/usr/local/bin`. Los flags usados -m (permisos 0755 owner rwx / group rx / other rx), -o (owner) -g (grupo) -t (target).

Se puede ver que está correctamente instalado al consultar la versión.

```
bitcoind --version  
> Bitcoin Core version v23.0.0
```

SHELL Copiar

## bitcoin user

La aplicación Core puede correr en segundo plano como un demonio y por razones de seguridad desde un nuevo usuario 'bitcoin' sin permisos de admin ni la posibilidad de cambiar archivos de configuración del sistema.

Creamos un nuevo usuario bitcoin

```
sudo adduser --gecos "" --disable-password bitcoin
```

SHELL Copiar

Añadimos el usuario admin al grupo bitcoin

```
sudo adduser admin bitcoin
```

SHELL Copiar

Permitimos al usuario bitcoin configurar Tor directamente al añadirlo al grupo debian-tor

```
sudo adduser bitcoin debian-tor
```

Copiar

## Data folder

El Bitcoin-Core descarga la data usando por defecto el directorio `.bitcoin` en `/home/`. En el Raspberry tenemos un disco externo específicamente para almacenar la data.

Para lo cual creamos un directorio en el disco duro.

```
mkdir /media/blockchain/data/bitcoin
sudo chown bitcoin: bitcoin /media/blockchain/data/bitcoin
```

SHELL Copiar

Y desde el usuario `bitcoin` creamos un link simbólico que apunte al directorio que hemos creado.

```
sudo su - bitcoin
whoami
ln -s /data/bitcoin /home/bitcoin/.bitcoin
```

SHELL Copiar

Para verificar que todo esta correctamente configurado usamos el comando `ls -la` que no debe mostrar error en su respuesta ('`.bitcoin`' en rojo indicaría que hay error).

```
g:~ $ ls -la
total 68
drwxr-xr-x 8 ghost ghost 4096 Nov  8 10:21 .
drwxr-xr-x 4 root  root 4096 Oct 28 10:58 ..
-rw----- 1 ghost ghost 8690 Nov  8 11:05 .bash_history
-rw-r--r-- 1 ghost ghost  220 Sep 21 22:51 .bash_logout
-rw-r--r-- 1 ghost ghost 3523 Sep 21 22:51 .bashrc
lrwxrwxrwx 1 ghost ghost   31 Oct 28 11:05 .bitcoin -> /media/blockchain/data/bi
tcoin/
drwxr-xr-x 3 ghost ghost 4096 Nov  8 06:24 .cache
```

#### Generando Credenciales de Acceso

Para otras instancias o programas que necesiten realizar consultas al Bitcoin Core, es que se crean credenciales de acceso adecuadas.

Bitcoin Core provee un simple programa en Python para generar un archivo de configuración.

En el directorio `bitcoin` descargamos RPCAuth (RPC remote procedure call).

```
cd .bitcoin
wget
https://raw.githubusercontent.com/bitcoin/bitcoin/master/share/rpcauth/r
pcauth.py
```

SHELL Copiar

Al correr el script, se solicitan un username `raspibolt` y un password como argumentos.

Este password es para **Bitcoin RPC** únicamente.

```
python3 rpcauth.py raspibolt YourPasswordB
> String to be appended to bitcoin.conf:
>
rpcauth=raspibolt:00d8682ce66c9ef3dd9d0c0a6516b10e$c31da4929b3d0e092ba1b
2755834889f888445923ac8fd69d8eb73efe0699afa
```

Copiar

Se copia la última línea de `rpcauth`.

#### Configuración bitcoind

Si abrimos desde el usuario `bitcoin` el archivo `bitcoin.conf` y llenamos el espacio `rpcauth` con la última línea que copiamos del `rpcauth.py` ejecutado anteriormente.

```
nano /home/bitcoin/.bitcoin/bitcoin.conf
```

SHELL Copiar

```
# /home/bitcoin/.bitcoin/bitcoin.conf

# Bitcoin daemon
server=1
txindex=1

# Network
listen=1
listenonion=1
proxy=127.0.0.1:9050
bind=127.0.0.1

# Connections
rpcauth=<replace with your own auth line generated by rpcauth.py>
zmqpubrawblock=tcp://127.0.0.1:28332
zmqpubrawtx=tcp://127.0.0.1:28333
whitelist=download@127.0.0.1           # for Electrs

# Raspberry Pi optimizations
maxconnections=40
maxuploadtarget=5000

# Initial block download optimizations
```

Copiar

```
dbcache=2000  
blocksonly=1
```

Se le otorgan permisos 640, lectura-escritura para bitcoin, el grupo solo puede leer el archivo y los otros no pueden hacer nada.

```
chmod 640 /home/bitcoin/.bitcoin/bitcoin.conf
```

Copiar

## Corriendo bitcoind

Aún con el usuario `bitcoin` iniciamos manualmente a bitcoind

```
bitcoind
```

SHELL Copiar

Empieza a correr un log en pantalla que muestra que funciona. Se puede parar con `ctrl+c`. Se otorgan permisos de lectura dentro del grupo para el archivo del log.

```
chmod g+r ~/data/bitcoin/debug.log  
exit
```

SHELL Copiar

Como usuarios admin realizamos un link del archivo `.bitcoin` para usarlo directamente.

```
$ ln -s /data/bitcoin /home/admin/.bitcoin
```

Copiar

## Autoinicio en boot

El sistema necesita correr el demonio de bitcoin (demon o demonio es un término en Linux para referir a un proceso que corre en segundo plano) automáticamente, incluso si nadie se logea en el sistema.

Se usa a `systemd`, otro demonio que controla que el proceso se haya iniciado usando los archivos de configuración.

Creamos un archivo para configurar el demonio bitcoin.

```
$ sudo nano /etc/systemd/system/bitcoind.service
```

Copiar

Con el siguiente contenido.

[Copiar](#)

```
# RaspiBolt: systemd unit for bitcoind
# /etc/systemd/system/bitcoind.service

[Unit]
Description=Bitcoin daemon
After=network.target

[Service]

# Service execution
#####
ExecStart=/usr/local/bin/bitcoind -daemon \
           -pid=/run/bitcoind/bitcoind.pid \
           -conf=/home/bitcoin/.bitcoin/bitcoin.conf \
           -datadir=/home/bitcoin/.bitcoin \
           -startupnotify="chmod g+r
/home/bitcoin/.bitcoin/.cookie"

# Process management
#####
Type=forking
PIDFile=/run/bitcoind/bitcoind.pid
Restart=on-failure
TimeoutSec=300
RestartSec=30

# Directory creation and permissions
#####
User=bitcoin
UMask=0027

# /run/bitcoind
RuntimeDirectory=bitcoind
RuntimeDirectoryMode=0710

# Hardening measures
#####
```

```
# Provide a private /tmp and /var/tmp.  
PrivateTmp=true  
  
# Mount /usr, /boot/ and /etc read-only for the process.  
ProtectSystem=full  
  
# Disallow the process and all of its children to gain  
# new privileges through execve().  
NoNewPrivileges=true  
  
# Use a new /dev namespace only populated with API pseudo devices  
# such as /dev/null, /dev/zero and /dev/random.  
PrivateDevices=true  
  
# Deny the creation of writable and executable memory mappings.  
MemoryDenyWriteExecute=true  
  
[Install]  
WantedBy=multi-user.target
```

Habilitamos el servicio y reiniciamos el sistema para empezar a correr.

```
sudo systemctl enable bitcoind.service  
sudo reboot
```

Copiar

#### Verificando bitcoind

Luego del reinicio ya debe empezar a sincronizar el blockchain para validarla.

Para ver si el servicio esta funcionando usamos el comando siguiente.

```
sudo systemctl status bitcoind.service
```

SHELL Copiar

```
> * bitcoind.service - Bitcoin daemon  
>     Loaded: loaded (/etc/systemd/system/bitcoind.service; enabled;  
vendor preset: enabled)  
>     Active: active (running) since Thu 2021-11-25 22:50:59 GMT; 7s ago  
>       Process: 2316 ExecStart=/usr/local/bin/bitcoind -daemon -  
pid=/run/bitcoind/bitcoind.pid -conf=/home/bitcoin/.bitcoin/bitcoin.>
```

```
conf -datadir=/home/bitcoin/.bitcoin (code=exited, status=0/SUCCESS)
>     Main PID: 2317 (bitcoind)
>         Tasks: 12 (limit: 4164)
>        CPU: 7.613s
>       CGroup: /system.slice/bitcoind.service
>                 `--2317 /usr/local/bin/bitcoind -daemon -
pid=/run/bitcoind/bitcoind.pid -conf=/home/bitcoin/.bitcoin/bitcoin.conf
> -datadir=/home/bitcoin/.bitcoin
>
```

Se debe tener cuidado especial en checar si el grupo `bitcoin` tiene los permisos correctos. Si se autentifica con `rpcauth` se crea un archivo `.cookie` que tiene las credenciales para tener acceso. Este archivo debe tener permisos adecuados si se quiere correr desde otros usuarios distintos a `bitcoin`. La salida debe contener permiso de lectura para el grupo y usuario `bitcoin`, si usamos `ls -la` revisamos que tenga `-rw-r----`.

```
ls -la /home/bitcoin/.bitcoin/.cookie
> -rw-r----- 1 bitcoin bitcoin 75 Dec 17 13:48
/home/bitcoin/.bitcoin/.cookie
```

Copiar

Ya se puede empezar a interactuar con el Core usando el `bitcoin-cli`.

```
$ bitcoin-cli getblockchaininfo
```

Copiar

En la siguiente imagen se muestra la salida un tiempo después de sincronizar el blockchain.

```
g- [ ] ~ $ bitcoin-cli getblockchaininfo
{
  "chain": "main",
  "blocks": 552645,
  "headers": 762370,
  "bestblockhash": "00000000000000000000000000000000272fc8214ee3e8a93d357a4b3b76c5ef00f13c72de2fea",
  "difficulty": 5646403851534.721,
  "time": 1544014175,
  "mediantime": 1544012549,
  "verificationprogress": 0.4658745420851458,
  "initialblockdownload": true,
  "chainwork": "00000000000000000000000000000000000000000000000000000000000000004464962d0c48369f2a8bad6",
  "size_on_disk": 221129427896,
  "pruned": false,
  "warnings": ""
}
```

La sincronización será completa una vez que  
“verificationprogress” llegue a 1 (0.99999...)

## Sync Finalizado

Después de 4 semanas el nodo completó su sincronización.

```
b bitcoin-cli getblockchaininfo
{
  "chain": "main",
  "blocks": 765846,
  "headers": 765846,
  "bestblockhash": "000000000000000000000000000000002cbeb2e91e5ddf
d5fb",
  "difficulty": 36950494067222.41,
  "time": 1670151348,
  "mediantime": 1670147938,
  "verificationprogress": 0.9999996613558999,
  "initialblockdownload": false,
  "chainwork": "000000000000000000000000000000000000000000000000000000000000000
",
  "size_on_disk": 501681390874,
  "pruned": false,
  "warnings": ""
}
```

Cabe notar que la sincronización se realizó con el nodo conectado inalámbricamente por Wifi. Para dar un punto de comparación cuando instalamos el mismo hardware con [Umbrel](#) demoró como 5 meses con conexión cableada.

## Optimizando Memoria

Una vez que el sync esta completado se puede disponer de la memoria RAM asignada solo para verificar bloques. Inicialmente esto aceleraba la descarga de bloques pero ahora es mejor usar esta memoria en otros programas (LND, Block Explorer y Mempool).

Comentamos las siguientes líneas en el archivo `sudo nano /home/bitcoin/.bitcoin/bitcoin.conf` para usar el cache default de 300 MB en lugar de los 2 GB que se asignaban.

```
#dbcacho=2000  
#blocksonly=1
```

Copiar

y se reinicia el servicio del Core para que cobre efecto.

```
sudo systemctl restart bitcoind
```

Copiar

## Cliente OpenTimestamps

Para verificar propiamente firmas de tiempo de confianza ([Timestamp Trustly](#)) de manera directa usamos un software open source que usa la información del blockchain que el mismo nodo ha verificado.

Instalamos el cliente de [OpenTimestamp client](#)

```
sudo pip install opentimestamps-client  
ots --vesion  
>v0.7.1
```

Copiar

## Verificando timestamp bitcoin-core

Como un ejercicio de aplicación mostramos que con el nodo podemos verificar la existencia de ciertos archivos antes de un cierto tiempo (en el caso Bitcoin una altura de bloque).

En el apartado [Verificación Timestamp](#) descargamos el SHA256SUMS de Bitcoin-Core. El SHA256SUMS es un archivo de texto que contiene el hash sha256 del archivo binario del software que los desarrolladores entregan con cada versión nueva del Core. Si estos binarios se manipulan maliciosamente el SHA256SUMS sería distinto y mostraría que ha sido modificado. Para certificar que este archivo SHA256SUMS existía en cierto momento y es único, su firma se estampa en el Blockchain de Bitcoin.

Bajamos los dos archivos SHA256SUMS y firma de tiempo 'SHA256SUMS.ots'.

```
wget https://bitcoincore.org/bin/bitcoin-core-24.0.1/SHA256SUMS  
wget https://bitcoincore.org/bin/bitcoin-core-24.0.1/SHA256SUMS.ots
```

Copiar

Con estos dos archivos se puede verificar con el nodo.

```
ots verify hello.txt.ots -f hello.txt
```

Copiar

```
$ ots verify SHA256SUMS.ots -f SHA256SUMS
Got 1 attestation(s) from https://alice.btc.calendar.opentimestamps.org
Got 1 attestation(s) from https://btc.calendar.catalaxy.com
Got 1 attestation(s) from https://finney.calendar.eternitywall.com
Got 1 attestation(s) from https://bob.btc.calendar.opentimestamps.org
Success! Bitcoin block 766964 attests existence as of 2022-12-11 -04
```

Mostrando que el archivo SHA256SUMS existía en el bloque 766964 (11 de diciembre de 2022).

## Electrum Server

Electrum es una implementación de código abierto para que una wallet externa se conecte al nodo Bitcoin Core y lo utilice como interfaz para realizar transacciones de manera simple y segura. Electrum indexa la información necesaria para la verificación rápida de transacciones.

Este software al ser de código libre tiene otras implementaciones como Electrs o Fulcrum. En nuestro caso elegimos [Electrs](#) por ser un desarrollado en Rust, un cliente ligero con buena Performance para el hardware limitado que tenemos.

## Pre instalación

Instalamos dependencias

```
sudo apt install cargo clang cmake
```

Copiar

## Activando Proxy y Firewall

En el apartado de [Seguridad](#) configuramos un servidor NGINX con reverse proxy. Se añade la configuración para Electrs.

Copiamos la siguiente configuración.

```
sudo nano /etc/nginx/stream-enabled/electrs-reverse-proxy.conf
```

Copiar

```
upstream electrs {  
    server 127.0.0.1:50001;  
}  
  
server {  
    listen 50002 ssl;  
    proxy_pass electrs;  
}
```

Copiar

Se testea y recarga NGINX.

```
sudo nginx -t  
sudo systemctl reload nginx
```

Copiar

Configuramos el Firewall

```
sudo ufw allow 50002/tcp comment 'electrum ssl'
```

Copiar

## Instalando Electrs

Descargamos el proyecto

```
mkdir /home/admin/rust  
cd /home/admin/rust  
git clone --branch v0.9.11 https://github.com/romanz/electrs.git  
cd electrs
```

Copiar

Siempre que se usa software de terceros, oficial o no, se recomienda comprobar y verificar si esta apropiadamente firmador por su desarrollador Roman Zeyde.

```
curl https://romanze.de/pgp.txt | gpg --import  
git verify-tag v0.9.11  
> gpg: Good signature from "Roman Zeyde <me@romanze.de>" [unknown]  
> gpg: WARNING: This key is not certified with a trusted signature!  
> gpg: There is no indication that the signature belongs to the  
owner.  
> Primary key fingerprint: 15C8 C357 4AE4 F1E2 5F3F 35C5 87CA E5FA 4691  
7CBB
```

Copiar

Ahora procedemos a compilar e instalar los binarios. Este proceso puede durar un tiempo, en mi caso alrededor de los 55 min.

```
cargo build --locked --release  
sudo install -m 0755 -o root -g root -t /usr/local/bin  
. ./target/release/electrs
```

Copiar

Creamos un nuevo usuario Electrs y lo añadimos al grupo bitcoin

```
sudo adduser --disabled-password --gecos "" electrs  
sudo adduser electrs bitcoin  
sudo su - electrs
```

Copiar

Creamos un archivo de configuración.

```
nano /data/electrs/electrs.conf
```

Copiar

Antes de editarlo notamos unas consideraciones:

1. Electrs crea una base de datos indexando y ordenando la información de transacciones de todos los bloques.
2. Hasta la fecha de elaborar este documento el Blockchain de Bitcoin llega a un poco mas de 500 GB de tamaño. La base de datos que indexa Electrs llega casi a los 33 GB.

Se decidió crear una carpeta donde almacenar la base de datos de Electrs en el mismo disco SDD externo.

```
sudo mkdir /media/blockchain/data/electrs  
sudo chown -R electrs:electrs /media/blockchain/data/electrs
```

Copiar

```
# /data/electrs/electrs.conf  
  
# Bitcoin Core settings  
network = "bitcoin"  
daemon_dir= "/home/bitcoin/.bitcoin"  
daemon_rpc_addr = "127.0.0.1:8332"  
daemon_p2p_addr = "127.0.0.1:8333"  
  
# Electrs settings  
electrum_rpc_addr = "127.0.0.1:50001"  
db_dir = "/media/blockchain/data/electrs/db"  
  
# Logging  
log_filters = "INFO"  
timestamp = true
```

Copiar

Con esta configuración ya se puede arrancar el servicio de Electrs

```
electrs --conf /home/blockchain/electrs/electrs.conf Copiar  
  
>Starting electrs 0.9.11 on aarch64 linux with Config { network: Bitcoin,  
db_path: "/home/blockchain/data/electrs/db/", daemon_dir:  
"/home/bitcoin/.bitcoin", daemon_auth:  
CookieFile("/home/bitcoin/.bitcoin/.cookie"), daemon_rpc_addr:  
127.0.0.1:8332, daemon_p2p_addr: 127.0.0.1:8333, electrum_rpc_addr:  
127.0.0.1:50001, monitoring_addr: 127.0.0.1:4224, wait_duration: 10s,  
jsonrpc_timeout: 15s, index_batch_size: 10, index_lookup_limit:  
Some(1000), reindex_last_blocks: 0, auto_reindex: true, ignore_mempool:  
false, sync_once: false, disable_electrum_rpc: false, server_banner:  
"Welcome to electrs 0.9.11 (Electrum Rust Server)!", args: [] }  
[2021-11-09T07:09:42.744Z INFO electrs::metrics::metrics_impl] serving  
Prometheus metrics on 127.0.0.1:4224  
[2021-11-09T07:09:42.744Z INFO electrs::server] serving Electrum RPC on  
127.0.0.1:50001  
[2021-11-09T07:09:42.812Z INFO electrs::db]  
"/home/blockchain/data/electrs/db/bitcoin": 0 SST files, 0 GB, 0 Grows  
[2021-11-09T07:09:43.174Z INFO electrs::index] indexing 2000 blocks:  
[1..2000]  
[2021-11-09T07:09:44.665Z INFO electrs::chain] chain updated:  
tip=00000000dfd5d65c9d8561b4b8f60a63018fe3933ecb131fb37f905f87da951a,  
height=2000  
[2021-11-09T07:09:44.986Z INFO electrs::index] indexing 2000 blocks:  
[2001..4000]  
[2021-11-09T07:09:46.191Z INFO electrs::chain] chain updated:  
tip=00000000922e2aa9e84a474350a3555f49f06061fd49df50a9352f156692a842,  
height=4000  
[2021-11-09T07:09:46.481Z INFO electrs::index] indexing 2000 blocks:  
[4001..6000]  
[2021-11-09T07:09:47.581Z INFO electrs::chain] chain updated:  
tip=00000000dbbb79792303bdd1c6c4d7ab9c21bba0667213c2eca955e11230c5a5,  
height=6000
```

Si funciona todo correctamente se sale del user `electrs`

```
exit Copiar
```

Para que el servidor Electrs se inicie automáticamente después de un reinicio configuramos el servicio.

```
sudo nano /etc/systemd/system/electrs.service
```

[Copiar](#)

```
# /etc/systemd/system/electrs.service
```

[Copiar](#)

```
[Unit]
Description=Electrs daemon
Wants=bitcoind.service
After=bitcoind.service

[Service]
# Service execution
#####
ExecStart=/usr/local/bin/electrs --conf /data/electrs/electrs.conf

# Process management
#####
Type=simple
Restart=always
TimeoutSec=120
RestartSec=30
KillMode=process

# Directory creation and permissions
#####
User=electrs

# /run/electrs
RuntimeDirectory=electrs
RuntimeDirectoryMode=0710

# Hardening measures
#####
# Provide a private /tmp and /var/tmp.
PrivateTmp=true

# Use a new /dev namespace only populated with API pseudo devices
# such as /dev/null, /dev/zero and /dev/random.
PrivateDevices=true

# Deny the creation of writable and executable memory mappings.
```

```
MemoryDenyWriteExecute=true
```

```
[Install]  
WantedBy=multi-user.target
```

Se habilita el servicio.

```
sudo systemctl enable electrs  
sudo systemctl start electrs
```

Copiar

Se puede observar el funcionamiento del servicio y las acciones que toma en el log.

```
sudo journalctl -f -u electrs
```

Copiar

## Acceso a Electrs usando TOR

Hasta este punto el servidor Electrum es accesible desde otra terminal en la misma red LAN de Internet. Para ser accesible en cualquier lugar del mundo y ser resistente a cualquier bloqueo habilitamos un servicio en TOR.

Se añade en el archivo `torrc` el servicio electrs.

```
sudo nano /etc/tor/torrc
```

Copiar

```
HiddenServiceDir /var/lib/tor/hidden_service_electrs/  
HiddenServiceVersion 3  
HiddenServicePort 50002 127.0.0.1:50002
```

Copiar

Recargamos la configuración de TOR y mostramos la dirección para conectarse.

```
sudo systemctl reload tor  
sudo cat /var/lib/tor/hidden_service_electrs/hostname  
> abcdefg.....xyz.onion
```

Copiar

## Actualizar el Core.

Simplemente se siguen los pasos del punto [Instalando el Core](#) para hacer la verificación de Checksum256, ots, firmas gpg y se procede a instalar de igual forma que el Core del ejemplo.

