Slackware ARM GCC aarch64-linux compilación cruzada para la Raspberry Pi

Prefacio

Estaba pensando en la CPU de 64 bits Cortex-A53 de mi Raspberry Pi 3 y por qué estoy usando principalmente el sistema operativo Slackware ARM de 32 bits en ella. Entonces empecé a preguntarme si sería posible construir un kernel arm64 y módulos para correr con Slackware ARM. Después de leer acerca de cómo se podría lograr esto, parecía claro que se requeriría cierta compilación cruzada. Aunque tengo algo de experiencia en la construcción de núcleos (kernel) Linux, especialmente para la plataforma Raspberry Pi, nunca había hecho ninguna compilación cruzada hasta hace una semana (2016-12-15). Así que, todo este concepto era totalmente nuevo para mí.

Para mi primer intento (y en gran parte basado en la lectura de cómo lo hacían otros usuarios) usé un sistema Ubuntu 16.04.1 LTS (64 bits) para compilar un núcleo arm64 para el Raspberry Pi 3. Sin embargo, los resultados de hacer las cosas por este método fueron un tanto escasos y crearon muchos errores imprevistos. Entonces recordé algo Mozes había publicado el his Slackware ARM FAQ page sobre paquetes que se estaban construyendo de forma nativa. Una investigación más profunda me llevó a darme cuenta de que el éxito asegurado probablemente se encontraría al compilar de forma cruzada en un dispositivo ARM usando Slackware ARM. ¡Eso es exactamente lo que hice! Muy exitosamente, debo añadir. Gracias de nuevo, Mozes. <3

Como han resultado las cosas, no fue tan difícil. Se requirió invertir algún tiempo en la lectura sobre las cadenas de herramientas (toolchains) y cómo construir empleando compiladores cruzados, así como en probar los resultados de la compilación cruzada, pero en general ha sido un proceso relativamente simple. Usar Slackware ARM current para compilar la arquitectura aarch64 fue la clave del éxito aquí. Ahora soy muy consciente de que, en comparación, intentar compilar aarch64 en un sistema Ubuntu x86_64 era menos que productivo.

Notas

Slackware ARM current se usó en una Raspberry Pi 3 para construir e instalar el compilador cruzado GCC aarch64-linux, y para construir el kernel, los módulos y los dispositivos de árbol blobs de arm64 Linux. Eso no quiere decir que Slackware ARM 14.2 no funcionará también, pero simplemente no hice ninguna compilación cruzada usando punto flotante soft. Lo mismo se aplica a la Raspberry Pi 1 y 2. Aunque *se podría* compilar de forma cruzada para estos dispositivos, no hice ninguna prueba con ellos. También tenga en cuenta que las opciones de configuración y los ajustes deben ser considerados primero.

Requerimientos

Como requisito previo, deberías haberlo logrado;

- Una Raspberry Pi 3 corriendo Slackware ARM con aproximadamente 8GB de espacio libre en tu sistema.

- gawk, git, bison y flex, ya instalado en tu sistema.
- Un lector de tarjetas microSD USB para conectar con su Raspberry Pi 3.
- Una tarjeta microSD (de respuesto) con Slackware current instalada en ella *no es esencial, pero se aconseja*.

De que se trata

Este tutorial le permitirá;

- Descargar los paquetes de fuentes requeridos con el objetivo de construir un compilador cruzado de GCC en Slackware ARM.
- Descargar el kernel de Linux para la Raspberry Pi GitHub y cambie a la rama de desarrollo de 64 bits.
- configure, e instale, un compilador cruzado GCC aarch64-linux (arm64) sobre tu Raspberry Pi 3.
- Construir un kernel aarch64 (arm64), módulos, y árbol de dispositivos e instalarlos en su tarjeta microSD actual Slackware ARM (de repuesto).
- build an aarch64 (arm64) Linux kernel, modules, and device tree blob(s), and install them on your (spare) Slackware ARM current microSD card.
- arrancar con éxito Slackware ARM actual en su Raspberry Pi 3 ejecutando un kernel aarch64 (arm64).

Para no arriesgarse a estropear el sistema ARM de Slackware que utiliza para la compilación cruzada, se debe utilizar una tarjeta microSD (de repuesto) que contenga un sistema ARM de Slackware que funcione para instalar el núcleo Linux arm64, los módulos y el árbol de dispositivos. Necesitará una versión reciente (es decir, posterior a septiembre de 2016) del Raspberry Pi bootloader/GPU firmware instalada en esta tarjeta microSD (de repuesto) para evitar problemas. Para estar seguro, arranque su RPi3 con él y actualice el firmware. La versión del núcleo de Linux en esta tarjeta microSD (de repuesto) no importa, ya que la sustituirá por el núcleo aarch64 (arm64).

Descarga del código fuente requerido y configuración

Primero de todo, como un usuario normal (es decir, no 'root') cree un directorio de trabajo. Por ejemplo, yo usualmente trabajo en el directorio /tmp:

```
cd /tmp
mkdir build-dir
cd build-dir
```

Este es el directorio donde se va a descargar todos los paquetes requeridos y las fuentes de Linux RPi.

Descargue el código fuente del kernel de Linux

Use el siguiente comando 'git' para descargar el kernel Linux para la Raspberry Pi en un directorio llamado 'linux'.

```
git clone https://github.com/raspberrypi/linux linux
```

Esto puede tardar un poco, dependiendo de la velocidad de su conexión a Internet y otros factores. Una vez que se haya completado necesita cambiar a la rama de desarrollo del kernel de 64 bits.

```
cd linux
git checkout rpi-4.8.y
```

Cuando eso esté hecho, deberías ver un mensaje de que 'origin/rpi-4.8.y' es la rama actual.

Descarga del código fuente

Antes de descargar el código fuente del paquete necesario para construir el compilador cruzado GCC, tenga en cuenta que pueden existir versiones de paquetes más recientes que las que se muestran aquí. Es posible que desee instalar versiones más recientes. Siempre es una buena idea revisar. Para mantener las cosas simples, puede considerar descargar una versión de GCC que coincida con la que tiene instalada actualmente. He leído muchos artículos sobre esto y la mayoría aconseja instalar la última versión de GCC disponible. Sin embargo, si está ejecutando Slackware ARM current tendrá instalado gcc-5.4.0 y esto es adecuado para lo que se necesita.

Así que, primero muévete de vuelta al directorio 'build-dir' y entonces descarga los siguientes paquetes.

```
cd ../
wget -nc https://ftp.gnu.org/gnu/binutils/binutils-2.27
wget -nc ftp://gcc.gnu.org/pub/gcc/infrastructure/cloog-0.18.1
wget -nc https://ftp.gnu.org/gnu/gcc/gcc-5.4.0
wget -nc https://ftp.gnu.org/gnu/glibc/glibc-2.24
wget -nc https://ftp.gnu.org/gnu/gmp/gmp-6.1.1
wget -nc ftp://gcc.gnu.org/pub/gcc/infrastructure/isl-0.16.1
wget -nc https://ftp.gnu.org/gnu/mpc/mpfr-3.1.5
wget -nc https://ftp.gnu.org/gnu/mpfr/mpc-1.0.3
```

Desempaquetado de los tarballs descargados

Ahora desempaqueta todos los tarballs descargados. Puede hacer esto fácilmente con el comando 'for'.

```
for t in *.tar*; do tar -xvf $t; done
```

Una vez que esto se haya completado, puede usar el comando '**ls**' para verificar que los directorios están presentes.

Creación de enlaces simbólicos de dependencias de GCC

Ahora es necesario crear algunos enlaces simbólicos en el directorio gcc-5.4.0. Estos apuntarán a

algunos de los directorios fuente que acaba de desempaquetar, que son dependencias de GCC, y cuando estos enlaces simbólicos estén presentes, GCC los construirá automáticamente.

```
cd gcc-5.4.0
ln -sf ../cloog-0.18.1 cloog
ln -sf ../gmp-6.1.1 gmp
ln -sf ../isl-0.16.1 isl
ln -sf ../mpfr-3.1.5 mpc
ln -sf ../mpfr-3.1.5 mpfr
```

Alternativamente, algunos artículos le aconsejarán que utilice el siguiente comando para lograr lo mismo.

```
cd gcc-5.4.0
./contrib/download_prerequisites
```

Personalmente, siempre prefiero el método manual porque entonces sé lo que se está descargando/instalando y qué esperar. Depende de usted el método que utilice.

Creación del directorio de instalación del compilador cruzado de GCC

Lo siguiente que hay que hacer es crear un directorio de instalación. Este es el directorio donde se instalará la cadena de herramientas. Una vez más, me gusta trabajar en /tmp para que el directorio de instalación se cree allí.

```
cd /tmp
mkdir gcc-cross
```

Exportación del directorio de instalación PATH

Es necesario exportar el directorio de instalación /bin a la variable $PATH de tu usuario.

```
export PATH=/tmp/gcc-cross/bin:$PATH
```

Para comprobar que esto ha funcionado, utilice el siguiente comando:

```
echo $PATH
```

Debería ver que la primera entrada $PATH es en la carpeta /bin de su directorio de instalación. Es importante que la carpeta /bin de su directorio de instalación aparezca antes de cualquier otra entrada en $PATH.

```
/tmp/gcc-
cross/bin:/usr/local/bin:/usr/bin:/bin:/usr/games:/usr/lib/kde4/libexec:/usr
/lib/qt/bin
```

Construcción del compilador cruzado aarch64 de GCC

Ahora con todo esto en su lugar, la concentración se centra en la construcción del ensamblador cruzado, el desensamblador cruzado, el cross-linker y otras herramientas útiles.

Construyendo binutils

Primero regrese al directorio 'build-dir' y luego cree un directorio de compilación para binutils. Notará las varias opciones de compilación pero como una explicación rápida; '–with-sysroot' básicamente le dice a binutils que habilite el soporte'sysroot' en el compilador cruzado apuntando a un directorio vacío por defecto, '–target=aarch64-linux' es el tipo de sistema de destino (arm64), y '–disable-multilib' significa que sólo queremos que los binutils trabajen con el conjunto de instrucciones aarch64 y nada más.

```
cd build-dir
mkdir build-binutils
cd build-binutils
../binutils-2.27/configure --prefix=/tmp/gcc-cross --with-sysroot --
target=aarch64-linux --disable-multilib
make -j4
make install
```

Instalación de las cabeceras del kernel de Linux

Aquí necesitas instalar las cabeceras del núcleo de Linux. Note la opción 'ARCH=arm64' para el proceso make. GCC usa 'aarch64' donde el kernel de Linux usa 'arm64'. Los dos proyectos de código abierto independientes identifican la misma arquitectura de CPU de forma diferente.

```
cd ../linux
make ARCH=arm64 INSTALL_HDR_PATH=/tmp/gcc-cross/aarch64-linux
headers_install
```

Construir GCC

Primero muévete al directorio 'buit-dir' y cree un directorio de compilación para GCC antes de construirlo. Note que solo han sido especificados como lenguajes de construcción C y C++. Eso es todo lo que necesitarás aquí. De hecho, las opciones de lenguaje de construcción disponibles permiten sólo una, o una selección, o todas, de las siguientes opciones '–enable-languages=all,ada,c,c++,fortran,go,jit,lto,objc,obj-c++'.

```
cd ../
mkdir build-gcc
cd build-gcc
../gcc-5.4.0/configure --prefix=/tmp/gcc-cross --target=aarch64-linux --
enable-languages=c,c++ --disable-multilib
make -j4 all-gcc
```

```
make install-gcc
```

Construir e instalar glibc

Primero vaya al directorio 'build-dir' y cree un directorio 'build-glibc'. Luego muévase al directorio 'build-glibc' antes de construirlo. –build=$MACHTYPE' es una variable de entorno predefinida que describe la Raspberry Pi 3 (en este caso) y es necesario compilar algunas herramientas adicionales que se utilizan durante el proceso de construcción. Observe que está instalando los archivos de inicio de la biblioteca C en el directorio de instalación (csu/crt1.o, csu/crti.o, y csu/crtn.o) por separado porque no parece haber una regla “make” que lo haga sin crear otros problemas.

```
cd ../
mkdir -p build-glibc
cd build-glibc
../glibc-2.24/configure --prefix=/tmp/gcc-cross/aarch64-linux --
build=$MACHTYPE --host=aarch64-linux --target=aarch64-linux --with-
headers=/tmp/gcc-cross/aarch64-linux/include --disable-multilib
libc_cv_forced_unwind=yes
make install-bootstrap-headers=yes install-headers
make -j4 csu/subdir_lib
install csu/crt1.o csu/crti.o csu/crtn.o /tmp/gcc-cross/aarch64-linux/lib
$ARCH_TARGET-gcc -nostdlib -nostartfiles -shared -x c /dev/null -o /tmp/gcc-
cross/aarch64-linux/lib/libc.so
touch /tmp/gcc-cross/aarch64-linux/include/gnu/stubs.h
```

Construcción de la biblioteca de soporte glibc

Muevase al directorio 'build-gcc' una vez más contruya las librerías de soporte para el compilador cruzado.

```
cd ../build-gcc
make -j4 all-target-libgcc
make install-target-libgcc
```

Concluir la construcción de la biblioteca glibc C

Muevase al directorio 'build-glibc' para finalizar de construir la librería glibc y entonces instálalo.

```
cd ../build-glibc
make -j4
make install
```

Finalización de la construcción de la librería GCC C++

Muevase al directorio 'build-glibc' para finalizar de construir la librería GCC C++ y luego instalela.

```
cd ../build-gcc
make -j4
make install
```

Probando el compilador cruzado

Para testear/chequear que tu compilador cruzado GCC aarch64-linux está funcionando correctamente ejecute los siguientes comandos.

```
aarch64-linux-gcc -v
```

Usted debería obtener una respuesta similar a la siguiente:

```
Using built-in specs.
COLLECT_GCC=aarch64-linux-gcc
COLLECT_LTO_WRAPPER=/tmp/gcc-cross/libexec/gcc/aarch64-linux/5.4.0/lto-
wrapper
Target: aarch64-linux
Configured with: ../gcc-5.4.0/configure --prefix=/tmp/gcc-cross --
target=aarch64-linux --enable-languages=c,c++ --disable-multilib
Thread model: posix
gcc version 5.4.0 (GCC)
```

Una vez completado este proceso, exporte la ruta del compilador cruzado de GCC a su usuario normal. Si desea realizar una compilación cruzada, hágalo una vez que haya reiniciado el sistema para que el compilador cruzado de GCC pueda localizarse a través de $PATH de su usuario. También tiene la opción de agregar este comando a tu **~/.profile** como configuración permanente. Si decide o no agregar permanentemente el PATH de GCC a su perfil ~/.profile depende totalmente de usted. Si está usando su sistema actual Slackware ARM para construir exclusivamente paquetes aarch64 (arm64), entonces tendría sentido hacerlo.

Ejemplo de comandos para exportar:

```
export PATH=/tmp/gcc-cross/bin:$PATH
```

El compilador cruzado GCC aarch64-linux sobre tu sistema Slackware ARM está listo para *rock-n-roll!*.

Construyendo el kernel arm64, modulos, y and árbol de dispositivos blob (DTB)

Para construir el kernel aarch64, módulos y el árbol de dispositivos blob(s) es exactamente el mismo método que el que se utilizaría en circunstancias normales. Comandos como 'make bzImage && make modules && make modules_install' pueden ser demasiado familiares para usted. En nuestro caso,

CFLAGS será usado para instruir al compilador cruzado de GCC a construir específicamente para la arquitectura aarch64 (arm64).

Creando el archivo .config del kernel arm64

Primero de todo, como siempre, es necesario estar en el directorio fuente del núcleo de Raspberry Pi Linux que está en la carpeta 'build-dir'. Luego necesita crear un archivo .config para el kernel, basado en los parámetros de Raspberry Pi 3. Para simplificar, puede generar un archivo por defecto .config (**defconfig**). Este archivo contiene la configuración del núcleo de Linux para el núcleo arm64 que va a construir. Para ello ejecute los siguientes comandos:

```
cd /tmp/build-dir/linux
make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-linux- bcmrpi3_defconfig
```

Tome nota aquí de los CFLAGS que han sido especificados. Ya deberían ser auto-explicativos. Ponga especial atención al final '-' de 'CROSS_COMPILE=aarch64-linux-' porque eso **NO** es un error tipográfico. ¡Esto tiene que ser así!

Construyendo el kernel arm64

Lo siguiente es construir el kernel, basado en el archivo .config que acabas de crear. Una vez más, usarás el mismo CFLAGS que antes. Puede incluso establecer una 'LOCALVERSION' aquí que añada lo que haya establecido al final de la versión del núcleo (por ejemplo, LOCALVERSION=“-arm64” le daría eventualmente 4.8.13-v8–arm64) una vez que el núcleo y los módulos hayan sido construidos. A modo de ejemplo lo usaremos aquí.

```
make -j4 Image ARCH=arm64 CROSS_COMPILE=aarch64-linux- LOCALVERSION="-arm64"
```

Así, aquí `make' construye el núcleo que se guardará con el nombre '**Image**'. El resto debe estar familiarizado. Este proceso tomará un tiempo. Tal vez una hora más o menos.

Construyendo el árbol de dispositivos blob(s)

El árbol de dispositivos es un medio para describir el hardware que el núcleo (kernel) lee en el momento del arranque para saber qué hardware existe en el sistema. En nuestro caso, se relaciona con la Raspberry Pi 3 y es el método por el cual el sistema sabe qué controladores cargar para el hardware. En los dispositivos basados en ARM, el uso de árboles de dispositivos se ha vuelto obligatorio para todos los nuevos SOCs, incluyendo la Raspberry Pi. Los arboles de dispositivos blobs que se construirán son '**bcm2710-rpi-3-b.dtb**' y '**bcm2837-rpi-3-b.dtb**'.

Para construir el árbol de dispositivos blob(s) para la Raspberry Pi corra los siguientes comandos:

```
make -j4 dtbs ARCH=arm64 CROSS_COMPILE=aarch64-linux- LOCALVERSION="-arm64"
```

Es básicamente lo mismo que hiciste para construir el núcleo, sólo que donde 'Image' es sustituido

por '**dtbs**'.

Construyendo los módulos arm64

Para construir los módulos del núcleo se hace de la misma manera que antes. Ejecute el siguiente comando:

```
make -j4 modules ARCH=arm64 CROSS_COMPILE=aarch64-linux- LOCALVERSION="-
arm64"
```

Observe cómo cada vez que el comando es el mismo, excepto cuando se especifica lo que está construyendo. Si ha establecido una 'LOCALVERSION' entonces debe ser la misma para construir el kernel y los módulos. Este proceso probablemente tardará un poco más que la construcción del núcleo arm64.

Instalando los módulos arm64

Una vez que los módulos han sido construidos, se debe hacer 'make modules_install'. El proceso debería instalar los módulos de tu núcleo (kernel) en el directorio '/tmp/lib/modules/4.8.13-v8'.

Usted podría construir módulos *out-of-tree*/ kernel pero, para mantener las cosas simples, los va a instalar en la ubicación usual.Una vez más, usted usará el mismo CFLAGS que antes pero sin ninguna configuración de 'LOCALVERSION'. Primero necesitas convertirte en usuario '**root**' e introducir una contraseña cuando se te pida. Para instalar los módulos de aarch64 ejecute los siguientes comandos:

```
su -
make -j4 modules_install ARCH=arm64 CROSS_COMPILE=aarch64-linux-
```

Es necesario ser usuario '**root**', o tener permisos de administrador, para instalar los módulos arm64. ¡Un usuario normal no tiene permisos para hacer esto!

Así que, como soy un gran creyente en ser minucioso, siempre verifico las cosas en cada oportunidad. Soló para estar seguros, si no hay nada más, porque siempre es una buena política. Asegúrese de que los archivos y directorios que acaba de pasar bastante tiempo compilando realmente existen en su sistema y que están en el lugar correcto. Si esta es la primera vez que instala el compilador cruzado de GCC en su sistema y/o construye el kernel, los módulos y el árbol de dispositivos (blob(s)), entonces no hace falta decirlo. Podrías hacer esto después de cada proceso de construcción, lo que yo también hago a menudo.

```
ls -lah arch/arm64/boot/Image
la -lah arch/arm64/boot/dts/broadcom/bcm*-rpi-3-b.dtb
ls -lah /lib/modules/4.8.13-v8*
```

Si puedes ver que todos ellos existen, entonces todo ha funcionado según lo planeado.

Copiando el kernel arm64, modulos, y el árbol de dispositivos blobs (DTB)

Conecte la tarjeta microSD (de repuesto) que contiene un sistema current Slackware ARM que funciona en su Raspberry Pi 3 usando un lector de tarjetas microSD USB. Primero es necesario montar la patición, luego copiar el kernel arm64, los módulos y el árbol de dispositivos blob(s) en ella. Usted debería estar logeado como usuario '**root**'. Si no, escriba los siguientes comando e ingrese la clave del usuario 'root' cuando se la pida:

```
su -
```

Es necesario ser usuario '**root**' para llevar adelante cualquier procedimiento de montaje. ¡Un usuario normal no tiene privilegios para hacer esto!

Como usuario 'root' escriba los siguientes comandos:

```
fdisk -l
```

Este comando debería mostrarle qué dispositivo está usando la tarjeta microSD (de repuesto) en su sistema. En nuestro caso es una tarjeta de 32 GB y es identificada como '**/dev/sda**', como se muestra a continuación. Esto nos dice que **/dev/sda1** es nuestra partición /boot y '**/dev/sda3**' es nuestra partición raíz. Los tuyos pueden ser asignados de manera diferente, así que tenlo en cuenta.

```
Device     Boot  Start      End  Sectors  Size Id Type
/dev/sda1  *        32   195327   195296 95.4M  c W95 FAT32 (LBA)
/dev/sda2       196608   720895   524288  256M 82 Linux swap
/dev/sda3       720896 61145087 60424192 28.8G 83 Linux
```

Con el objetivo de montar las particiones primero necesitas crear los puntos de montaje. Trabajando en el directorio /tmp puedes hacerlo así:

```
cd /tmp
mkdir rpi-boot
mkdir rpi-root
mount /dev/sda1 rpi-boot
mount /dev/sda3 rpi-root
```

Para comprobar que estas haciendo todo correctamente, usa el comando '**mount**'. La salida salida de esto debe ser similar a los siguiente:

```
/dev/mmcblk0p3 on / type ext4 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
tmpfs on /dev/shm type tmpfs (rw)
/dev/mmcblk0p1 on /boot type vfat (rw,fmask=177,dmask=077)
/dev/sda1 on /tmp/rpi-boot type vfat (rw)
/dev/sda3 on /tmp/rpi-root type ext4 (rw)
```

Lo siguiente que hay que hacer es copiar el núcleo, los módulos y el árbol de dispositivos blobs de arm64 a los directorios recién montados. Es importante hacer esto bien. Después de todo el trabajo duro que has hecho, sería una pena estropearlo en este momento.

Para copiar estos archivos, como usuario '**root**', ejecuta los siguientes comandos:

```
cp build-dir/linux/arch/arm64/boot/Image rpi-boot/boot/kernel8.img
cp build-dir/linux/arch/arm64/boot/dts/broadcom/bcm*-rpi-3-b.dtb rpi-
boot/boot
cp -rv /lib/modules/4.8.13-v8* rpi-root/lib/modules/
```

Una vez hecho esto, compruebe que los archivos que ha copiado están presentes y en el lugar correcto.

```
ls -lah rpi-boot/boot/kernel*
ls -lah rpi-boot/boot/bcm*-rpi-3-b.dtb
ls -lah rpi-root/lib/modules/4.8.13-v8*
```

Si todo se ve bien, entonces lo siguiente que necesita hacer es borrar el antiguo kernel armv7 en el directorio rpi-boot/boot. Este antiguo núcleo se llama '**kernel7.img**' y para evitar cualquier conflicto con el nuevo arm64 'kernel8.img' debería eliminarlo

```
rm -rf rpi-boot/kernel7.img
```

No debería ser necesario cambiar los archivos config.txt o cmdline.txt. Si estas utilizando un bootloader/firmware GPU (por ejemplo post-septiembre 2016) entonces nada más es necesario cambiar o borrar.

Ahora puedes desmontar los directorios previamente montados.

```
umount rpi-boot
umount rpi-root
```

Arrancando Slackware ARM aarch64

Apaga tu Raspberry Pi.

```
poweroff
```

Sacar la memoria microSD y tarjetas swaps. Encender la Raspberry Pi y arrancar con la memoria microSD sobre la cual fue copiado el kernel, los módulos y el árbol de directorio de dispositivos blobs.

El resultado final

Luego de arrancar el sistema con el kernel arm64, yo me logue de forma remota via SSH como el usuario 'root'. Entonces yo ejecute los siguientes comandos:

```
login as: root
root@192.168.10.33's password:
Last login: Sat Dec 17 20:32:50 2016 from 192.168.10.10
Linux 4.8.13-v8-arm64.
root@drie:~# cat /proc/version
```

```
Linux version 4.8.13-v8-arm64 (exaga@drie) (gcc version 5.4.0 (GCC) ) #2 SMP
Fri Dec 16 18:43:38 GMT 2016
root@drie:~# uname -a
Linux drie 4.8.13-v8-arm64 #2 SMP Fri Dec 16 18:43:38 GMT 2016 aarch64
GNU/Linux
root@drie:~# cat /etc/slackware-version
Slackware 14.2
root@drie:~# cat /proc/device-tree/model
Raspberry Pi 3 Model B Rev 1.2
root@drie:~# cat /proc/cmdline | awk -v RS=" " -F= '/serial/ { print $2 }'
0x4135b94e
root@drie:~#
```

Aunque ya me he encontrado con algunas cosas que necesitan trabajo y atención, es un comienzo. Espero encontrar más tiempo para dedicar a Slackware arm64 en navidad y año Nuevo 2017. Gracias por su interés. <3

Fuentes

ftp://ftp.arm.slackware.com/slackwarearm/slackwarearm-current/slackware/a/gawk-4.1.4-arm-1.txz # Slackware ARM current - paquete gawk.
ftp://ftp.arm.slackware.com/slackwarearm/slackwarearm-current/slackware/d/git-2.11.0-arm-1.txz # Slackware ARM current - paquete git.
ftp://ftp.arm.slackware.com/slackwarearm/slackwarearm-current/slackware/d/bison-3.0.4-arm-1.txz # Slackware ARM current - paquete bison.
ftp://ftp.arm.slackware.com/slackwarearm/slackwarearm-current/slackware/d/flex-2.6.0-arm-1.txz # Slackware ARM current - paquete flex.
http://arm.slackware.com/FAQs # Proyecto Slackware ARM Linux preguntas frecuentes.
http://wiki.osdev.org/GCC_Cross-Compiler # Documentación de la compilación cruzada de GCC.
https://www.raspberrypi.org/documentation/linux/kernel # Documentación del kernel Raspberry Pi.
https://www.github.com/raspberrypi/ # Raspberry Pi Foundation GitHub repository Linux kernel, bootloader/GPU firmware.
https://ftp.gnu.org/gnu/ # Fuentes de paquetes GCC, binutils, glibc, gmp, mpc, mpfr.
ftp://gcc.gnu.org/pub/gcc/infrastructure # cloog, paquete fuente pisl.

- Escrito originalmente por Exaga.
- Traducido por — *rramp 2019/07/16 22:18 (UTC)*.

howtos, hardware, aarch64, cross-compile, author exaga