Instalar desde las fuentes

1. Obtener las fuentes

A veces no es posible obtener un programa o una librería desde un paquete para su distribución. En este caso, queda la solución de compilar e instalar uno mismo el programa desde las fuentes.

Eso es posible para la mayoría de los programas en Linux gracias a las ventajas de los programas libres y de la licencia GPL tal como quedó definida en el primer capítulo, o con otras licencias libres. Cualquier programa libre va acompañado con sus fuentes. Por lo tanto, usted mismo puede volver a reconstruir el programa al compilarlo.

Es posible obtener un archivo fuente en diversos sitios web, como por ejemplo SourceForge. Suele ser un archivo comprimido en formato tgz (archivo tar comprimido con gzip) o tar.bz2 (archivo tar comprimido en formato bzip2). Contiene:

- el código fuente en forma de archivos `.c`, `.h`, `.cpp`, etc., según el lenguaje;
- a veces un archivo `Makefile` que permite automatizar la compilación del producto;
- a menudo un archivo `.configure` que permite generar el archivo Makefile en función de su instalación y de varias opciones.

2. Requisitos y dependencias

Para compilar su producto, tiene que respetar unos requisitos:

- presencia de la herramienta make;
- presencia del compilador o de los compiladores necesarios, en particular gcc;
- presencia de las dependencias: librerías, intérpretes, etc.

Este último punto es muy importante. Si falta una dependencia, se arriesga a encontrarse con varios problemas:

- no logrará preparar las fuentes para la compilación;
- la compilación generará errores;
- se compilará el programa, pero con menos opciones;
- no se iniciará el binario resultante.

El comando **./configure** le proporcionará las dependencias que faltan y su versión si es posible. En este caso, puede instalarlas desde los paquetes de su distribución, o bien instalarlas desde las fuentes.

Cuando se compila desde las fuentes sin pasar por un gestor de paquetes, se pierde una parte de la gestión de las dependencias. Cuando instala paquetes que dependen de una versión de la herramienta instalada desde las fuentes, es posible, si las dependencias se basan en la existencia de un paquete y no de un archivo, que el gestor le impida instalar el paquete. Busque correctamente entre los repositorios, oficiales o no, si el programa existe en forma de paquete antes de volver a compilarlo o si existe un paquete fuente.

En cualquier caso, no hace falta que sea root para compilar el programa. Sin embargo, según el destino de la aplicación, tendrá que pasar a root para finalizar la instalación.

3. Ejemplo de instalación

Va a compilar e instalar el servidor web nginx. Este servidor web es más ligero y más rápido que un servidor apache pero también más simple.

Recupere las últimas fuentes en https://nginx.org/download, versión 1.17.7 de enero de 2020. Para ello, use wget.

```
$ wget https://nginx.org/download/nginx-1.17.7.tar.gz
--2021-03-20 07:49:44-- https://nginx.org/download/nginx-1.17.7.tar.gz
Resolviendo nginx.org (nginx.org)... 3.125.197.172, 52.58.199.22, 2a05:d014:edb:5704::6, ...
Conectando con nginx.org (nginx.org)[3.125.197.172]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 1037747 (1013K) [application/octet-stream]
Guardando como: “nginx-1.17.7.tar.gz”

nginx-1.17.7.tar.gz      100%[====================>]    1013K  5,73MB/s    en 0,2s

2021-03-20 07:49:44 (5,73 MB/s) - “nginx-1.17.7.tar.gz” guardado [1037747/1037747]
$ ls -l
-rw-r--r-- 1 alejandro alejandro 1037747 dic 24 2019 nginx-1.17.7.tar.gz
```

➔ Descomprima el archivo:

```
$ tar zxvf nginx-1.17.7.tar.gz
nginx-1.17.7/
nginx-1.17.7/auto/
nginx-1.17.7/conf/
nginx-1.17.7/contrib/
nginx-1.17.7/src/
nginx-1.17.7/configure
nginx-1.17.7/LICENSE
nginx-1.17.7/README
nginx-1.17.7/html/...
```

➔ Vaya a la carpeta **`nginx-1.17.7`** creada por la descompresión:

```
$ cd nginx-1.17.7
$
 ls -l
total 792
drwxr-xr-x 6 alejandro alejandro   4096 mar 20 07:51 auto
-rw-r--r-- 1 alejandro alejandro 301572 dic 24 2019 CHANGES
-rw-r--r-- 1 alejandro alejandro 460207 dic 24 2019 CHANGES.ru
```

```
drwxr-xr-x 2 alejandro alejandro  4096 mar 20 07:51 conf
-rwxr-xr-x 1 alejandro alejandro  2502 dic 24 2019 configure
drwxr-xr-x 4 alejandro alejandro  4096 mar 20 07:51 contrib
drwxr-xr-x 2 alejandro alejandro  4096 mar 20 07:51 html
-rw-r--r-- 1 alejandro alejandro  1397 dic 24 2019 LICENSE
-rw-r--r-- 1 alejandro alejandro   352 mar 20 08:04 Makefile
drwxr-xr-x 2 alejandro alejandro  4096 mar 20 07:51 man
drwxr-xr-x 3 alejandro alejandro  4096 mar 20 08:05 objs
-rw-r--r-- 1 alejandro alejandro    49 dic 24 2019 README
drwxr-xr-x 9 alejandro alejandro  4096 mar 20 07:51 src
```

➔ Observe la presencia del archivo `configure` que es ejecutable.

```
$ ./configure --help

  --help                         print this message

  --prefix=PATH                  set installation prefix
  --sbin-path=PATH               set nginx binary pathname
  --modules-path=PATH            set modules path
  --conf-path=PATH               set nginx.conf pathname
...
```

Una opción importante de `configure` es **`--prefix`** . Define la ubicación de la instalación una vez compilado el producto. Por defecto, el programa se instala en **`/usr/local/`** .

➔ Ejecute **`./configure prefix=/opt/nginx`** . Le informará de las dependencias que faltan en caso necesario.

```
$ ./configure --prefix=/opt/nginx
checking for OS
 + Linux 5.4.0-67-generic x86_64
checking for C compiler ... found
 + using GNU C compiler
 + gcc version: 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04)
checking for gcc -pipe switch ... found
```

```
checking for -Wl,-E switch ... found
checking for gcc builtin atomic operations ... found
checking for C99 variadic macros ... found
checking for gcc variadic macros ... found
...
checking for PCRE library in /opt/local/ ... not found

./configure: error: the HTTP rewrite module requires the PCRE library.
You can either disable the module by using --without-http_rewrite_module
option, or install the PCRE library into the system, or build the PCRE library
statically from the source with nginx by using --with-pcre=<path> option.
```

- Fíjese en el bloque en negrita: falta una librería. Esto se trata de un error, es pues muy importante instalar la librería en cuestión. Si `configure` no es más que una alerta (warning), tiene dos opciones:
  - instalar la librería que falta (y su paquete de desarrollo) bien desde las fuentes, bien desde el gestor de paquetes de su distribución. Por ejemplo en Ubuntu: `apt-get install libpcre3-dev zliblg-dev` .
  - no instalarlo: si ésta no es vital. Sin embargo, faltarán algunas funcionalidades.

- Después de haber resuelto, si lo desea, las dependencias, debe volver a ejecutar el comando **./configure**. El comando **configure** crea el archivo `Makefile` correspondiente. Este archivo contiene el conjunto de las reglas, rutas y opciones para compilar el programa. El comando **make** depende de él.

- Inicie la compilación con el comando **make**. Puede que aparezcan alertas (líneas warning): no significa obligatoriamente que el programa no compilará o no funcionará posteriormente. De todas maneras, si la compilación produce errores, se parará por sí misma con un mensaje de error del compilador que puede a veces (pero no siempre) ponerle sobre la pista de una solución.

La compilación puede ser más o menos larga según el producto.

```
$ make
make -f objs/Makefile
make[1]: se entra en el directorio '/home/alejandro/nginx-1.17.7'
cc -c -pipe -O -W -Wall -Wpointer-arith -Wno-unused-parameter -Werror -g
-I src/core -I src/event -I src/event/modules -I src/os/unix -I objs \
        -o objs/src/core/nginx.o \
        src/core/nginx.c
cc -c -pipe -O -W -Wall -Wpointer-arith -Wno-unused-parameter -Werror -g
-I src/core -I src/event -I src/event/modules -I src/os/unix -I objs \
        -o objs/src/core/ngx_log.o \
        src/core/ngx_log.c
cc -c -pipe -O -W -Wall -Wpointer-arith -Wno-unused-parameter -Werror -g
-I src/core -I src/event -I src/event/modules -I src/os/unix -I objs \
        -o objs/src/core/ngx_palloc.o \
        src/core/ngx_palloc.c
cc -c -pipe -O -W -Wall -Wpointer-arith -Wno-unused-parameter -Werror -g
-I src/core -I src/event -I src/event/modules -I src/os/unix -I objs \
        -o objs/src/core/ngx_array.o \
        src/core/ngx_array.c
cc -c -pipe -O -W -Wall -Wpointer-arith -Wno-unused-parameter -Werror -g
-I src/core -I src/event -I src/event/modules -I src/os/unix -I objs \
        -o objs/src/core/ngx_list.o \
        src/core/ngx_list.c
cc -c -pipe -O -W -Wall -Wpointer-arith -Wno-unused-parameter -Werror -g
-I src/core -I src/event -I src/event/modules -I src/os/unix -I objs \
        -o objs/src/core/ngx_hash.o \
        src/core/ngx_hash.c
cc -c -pipe -O -W -Wall -Wpointer-arith -Wno-unused-parameter -Werror -g
-I src/core -I src/event -I src/event/modules -I src/os/unix -I objs \
        -o objs/src/core/ngx_buf.o \
        src/core/ngx_buf.c
...

objs/ngx_modules.o \
-ldl -lpthread -lcrypt -lpcre -lz \
-Wl,-E
sed -e "s|%%PREFIX%%|/opt/ngnix|" \
        -e "s|%%PID_PATH%%|/opt/ngnix/logs/nginx.pid|" \
        -e "s|%%CONF_PATH%%|/opt/ngnix/conf/nginx.conf|" \
        -e "s|%%ERROR_LOG_PATH%%|/opt/ngnix/logs/error.log|" \
```

```
    < man/nginx.8 > objs/nginx.8
make[1]: se sale del directorio '/home/alejandro/nginx-1.17.7''
```

Al finalizar sin error la compilación, termine la instalación del producto con **make install**. Cuidado: el producto se va a instalar en /usr/local/, lo que requiere los derechos del usuario root. Según su distribución, deberá usar o bien **su**, o bien el comando **sudo**.

```
sudo make install
make -f objs/Makefile install
make[1]: se entra en el directorio '/home/alejandro/nginx-1.17.7'
test -d '/opt/ngnix' || mkdir -p '/opt/ngnix'
test -d '/opt/ngnix/sbin' \
    || mkdir -p '/opt/ngnix/sbin'
test ! -f '/opt/ngnix/sbin/nginx' \
    || mv '/opt/ngnix/sbin/nginx' \
        '/opt/ngnix/sbin/nginx.old'
cp objs/nginx '/opt/ngnix/sbin/nginx'
test -d '/opt/ngnix/conf' \
    || mkdir -p '/opt/ngnix/conf'
cp conf/koi-win '/opt/ngnix/conf'
cp conf/koi-utf '/opt/ngnix/conf'
cp conf/win-utf '/opt/ngnix/conf'
test -f '/opt/ngnix/conf/mime.types' \
    || cp conf/mime.types '/opt/ngnix/conf'
cp conf/mime.types '/opt/ngnix/conf/mime.types.default'
test -f '/opt/ngnix/conf/fastcgi_params' \
    || cp conf/fastcgi_params '/opt/ngnix/conf'
cp conf/fastcgi_params \
    '/opt/ngnix/conf/fastcgi_params.default'
test -f '/opt/ngnix/conf/fastcgi.conf' \
    || cp conf/fastcgi.conf '/opt/ngnix/conf'
cp conf/fastcgi.conf '/opt/ngnix/conf/fastcgi.conf.default'
test -f '/opt/ngnix/conf/uwsgi_params' \
    || cp conf/uwsgi_params '/opt/ngnix/conf'
cp conf/uwsgi_params \
    '/opt/ngnix/conf/uwsgi_params.default'
test -f '/opt/ngnix/conf/scgi_params' \
    || cp conf/scgi_params '/opt/ngnix/conf'
```

```
cp conf/scgi_params \
      '/opt/ngnix/conf/scgi_params.default'
test -f '/opt/ngnix/conf/nginx.conf' \
      || cp conf/nginx.conf '/opt/ngnix/conf/nginx.conf'
cp conf/nginx.conf '/opt/ngnix/conf/nginx.conf.default'
test -d '/opt/ngnix/logs' \
      || mkdir -p '/opt/ngnix/logs'
test -d '/opt/ngnix/logs' \
      || mkdir -p '/opt/ngnix/logs'
test -d '/opt/ngnix/html' \
      || cp -R html '/opt/ngnix'
test -d '/opt/ngnix/logs' \
      || mkdir -p '/opt/ngnix/logs'
make[1]: se sale del directorio '/home/alejandro/nginx-1.17.7'
...
```

➔ Diríjase a `/opt/nginx/sbin` y pruebe el producto:

```
$

alejandro@ubuntu:/opt/ngnix/sbin$ sudo ./nginx -v
nginx version: nginx/1.17.7

alejandro@ubuntu:/opt/ngnix/sbin$ sudo ./nginx
alejandro@ubuntu:/opt/ngnix/sbin$ ps auxww | grep nginx
root      2328  0.0  0.0   4588   384 ?      Ss   12:45   0:00 nginx: master process ./nginx
nobody    2329  0.0  0.0   5264  2892 ?      S    12:45   0:00 nginx: worker process
alejand+  2331  0.0  0.0   6528   672 pts/1  S+   12:45   0:00 grep --color=auto nginx
```

Si lo requiere, instale curl (`apt-get install curl`) y pruebe un acceso a su servidor web:

```
$ curl -I http://127.0.0.1
HTTP/1.1 200 OK
Server: nginx/1.17.7
Date: Sun, 21 Mar 2021 12:47:06 GMT
```

```
Content-Type: text/html
Content-Length: 612
Last-Modified: Sat, 20 Mar 2021 08:07:25 GMT
Connection: keep-alive
ETag: "6055ad3d-264"
Accept-Ranges: bytes
```

Felicidades, la compilación y la instalación han funcionado perfectamente.

4. Desinstalación

La mayoría de los `Makefile` permiten la desinstalación. Se efectúa con el comando **make uninstall**. Nuestro ejemplo no lo permite, pero, como hemos instalado nginx en su propio directorio, basta con eliminar este.

5. Las bases del Makefile

a. Bases

El programa **make** utiliza un archivo Makefile para ejecutar un conjunto de acciones como la compilación de un proyecto, pero este archivo no se limita a eso: se trata de una especie de script por niveles.

Supongamos el proyecto siguiente, encargado de que aparezca «Hola». Muchas cosas para un ejemplo tan limitado, pero se trata de un ejemplo.

```
$ cat hola.h
#ifndef H_HOLA
#define H_HOLA
void Hola(void);
#endif
```

```
$ cat hola.c
#include <stdio.h>
#include <stdlib.h>

void Hola(void)
{
    printf("Hola\n");
}

$ cat main.c
#include <stdio.h>
#include <stdlib.h>
#include "hola.h"

int main(void)
{
    Hola();
    return 0;
}
```

Para compilar este proyecto a mano, debe ejecutar las etapas siguientes:

```
$ gcc -o hola.o -c hola.c
$ gcc -o main.o -c main.c
Gcc -o hola hola.o main.o
$ ./hola
Hola
```

El Makefile se compone de reglas con la estructura siguiente:

```
destino: dependencia
comandos
```

Por lo tanto, un primer Makefile podría ser:

```
$ cat Makefile
```

```
hola: hola.o main.o
   gcc -o hola hola.o main.o

hola.o: hola.c
   gcc -o hola.o -c hola.c

main.o: main.c hola.h
   gcc -o main.o -c main.c
```

Primera regla: para ejecutar la regla hola, hace falta disponer de los archivos **`hola.o`** y **`main.o`**. Si se dispone de ellos, hay que ejecutar el comando **gcc -o hola hola.o main.o**.

Segunda regla: para ejecutar la regla `hola.o` hace falta disponer del archivo **`hola.c`**. Si está presente, entonces se ejecuta el comando **gcc -o hola.o -c hola.c**.

Tercera regla: para ejecutar la regla `main.o` hace falta disponer de los archivos **`main.c`** y **`hola.h`**. Si están presentes, entonces se ejecuta el comando **gcc -o main.o -c main.c**.

Las dos últimas reglas permiten resolver la primera. Si usted ejecuta el comando **make**, éste va a determinar cuáles son las reglas aplicables, en qué orden y luego va a aplicarlas en el orden de las dependencias. Si los archivos están actualizados, **make** no vuelve a construirlos a no ser que hayan sido modificados.

```
$ rm -f *.o
$ make
gcc -o hola.o -c hola.c
gcc -o main.o -c main.c
gcc -o hola hola.o main.o
$ ./hola
Hola
```

b. Makefile intermedio

El anterior Makefile funciona, pero no es óptimo:

- No permite compilar varios binarios.

- No permite limpiar los archivos temporales (.o) después de la compilación.
- No permite forzar la compilación del proyecto.

El añadido de nuevas reglas permite paliar estos problemas:

- all: genera n reglas;
- clean: limpia los .o;
- mrproper: llama a clean y suprime los binarios.

```
$ cat Makefile
all: hola

hola: hola.o main.o
	gcc -o hola hola.o main.o

hola.o: hola.c
	gcc -o hola.o -c hola.c

main.o: main.c hola.h
	gcc -o main.o -c main.c

clean:
	rm -rf *.o

mrproper: clean
	rm -rf hola

$ make clean
rm -rf *.o
$ make mrproper
rm -rf *.o
rm -rf hola
$ make all
gcc -o hola.o -c hola.c
gcc -o main.o -c main.c
gcc -o hola hola.o main.o
```

c. Un poco más complejo

Variables de usuario

Para terminar esta pequeña presentación, puede definir variables en su archivo y utilizar variables internas predefinidas.

El Makefile se convierte en:

```
$ cat Makefile
CC=gcc
CFLAGS=-W -Wall -ansi -pedantic
LDFLAGS=
EXEC=hola

all: $(EXEC)

hola: hola.o main.o
    gcc -o hola hola.o main.o $(LDFLAGS)

hola.o: hola.c
    gcc -o hola.o -c hola.c $(CFLAGS)

main.o: main.c hola.h
    gcc -o main.o -c main.c $(CFLAGS)

clean:
    rm -rf *.o

mrproper: clean
    rm -rf $(EXEC)
```

Variables internas

Entre las variables internas:

- $@: nombre del destino.
- $<: nombre de la primera dependencia.
- $^: lista de las dependencias.
- $?: dependencias más recientes que la actual.
- $*: nombre del archivo sin el sufijo.

El Makefile se convierte en:

```
$ cat Makefile
CC=gcc
CFLAGS=-W -Wall -ansi -pedantic
LDFLAGS=
EXEC=hola

all: $(EXEC)

hola: hola.o main.o
	gcc -o $@ $^ $(LDFLAGS)

hola.o: hola.c
	gcc -o $@ -c $ $(CFLAGS)

main.o: main.c hola.h
	gcc -o $@ -c $ $(CFLAGS)

clean:
	rm -rf *.o

mrproper: clean
	 rm -rf $(EXEC)
```

Reglas de inferencia

Existen reglas predefinidas, basadas en accesos directos, que permiten generar metas en función del nombre del archivo C y objeto: **%.o: %.c**. La tentación de crear una regla única para **`main.o`** y **`hola.o`** es grande:

```
%.o: %.c
	gcc -o $@ -c $< $(CFLAGS)
```

Esta regla es correcta, pero falta la dependencia del header **`hola.h`** : si se modifica éste, ya no se compila el proyecto. Hay que añadir una regla específica:

```
main.o: hola.h
```

El Makefile se convierte en:

```
$ cat Makefile
CC=gcc
CFLAGS=-W -Wall -ansi -pedantic
LDFLAGS=
EXEC=hola

all: $(EXEC)

hola: hola.o main.o
	gcc -o $@ $^ $(LDFLAGS)

%.o: %.c
	gcc -o $@ -c $< $(CFLAGS)

main.o: hola.h

clean:
	rm -rf *.o

mrproper: clean
	rm -rf $(EXEC)
```