

# **Memoria Práctica 7**

## **Administración de Servidores**

**Instalación de programas desde código fuente, y realización de copias de seguridad**

Almudena García Jurado-Centurión

# Índice

## Introducción

### Parte 1: Instalando un programa desde su código fuente

- Descargando los fuentes
- Leyendo la documentación
- Instalando dependencias
- Ejecutando autojoe
- Ejecutando configure
- Compilando el programa
- Instalando el programa
- Probando el programa

### Parte 2: Realizando una copia de seguridad

1. Obteniendo lista de ficheros
2. Generando copia de seguridad
3. Transfiriendo ficheros a Gamba
4. Extrayendo la copia de seguridad
5. Avisando a los usuarios

### Parte 3: Programando copias de seguridad incrementales

1. Creando los directorios /backup en Choco y Gamba
2. Generando las copias de seguridad
3. Probando el procedimiento en Choco
4. Probando la copia de seguridad incremental
5. Extrayendo la copia de seguridad
6. Creando el script
  - Configurando rsync para transferir ficheros sin clave
  - Escribiendo el script
7. Programando el script con cron

## Conclusiones

## **Introducción**

En esta práctica aprenderemos a realizar diferentes tareas de mantenimiento del sistema.

En la primera parte, instalaremos un programa desde su código fuente: descargando los fuentes de los repositorios, instalando sus dependencias, y compilando e instalando el programa mediante la herramienta `make`.

En la segunda parte, practicaremos la generación y restauración de copias de seguridad mediante `tar` y `rsync`, y lo usaremos para transferir a Gamba el programa instalado en Choco

Finalmente, en la tercera parte prepararemos un sistema de copias de seguridad incrementales usando `tar`, `rsync` y `cron`.

## Parte 1: Instalando un programa desde su código fuente

En esta parte, realizaremos la instalación del programa Joe's Own Editor, también conocido como Joe, desde su código fuente. Esta instalación se realizará en la máquina virtual Choco, creada en la práctica anterior.

Habitualmente, esta instalación se compone de 4 pasos:

- Descarga de los fuentes
- Instalación de las dependencias
- Ejecución del fichero `configure`, que nos generará el `Makefile`
- Compilación mediante el comando `make`
- Instalación mediante `make install`

Aunque estos pasos pueden variar dependiendo del programa, para lo cual es importante leerse su documentación antes de comenzar el proceso de compilación.

### • Descargando los fuentes

Comenzaremos descargando los fuentes del programa. Para ello, nos vamos a su repositorio, ubicado en <https://github.com/jhallen/joe-editor> y descargamos sus fuentes usando `git`.

Como `git` no está instalado, lo instalamos mediante `yum install git`

```
Dependencies Resolved
=====
Package                Arch      Version              Repository           Size
=====
Installing:
git                    i686      1.7.1-9.el6_9        updates              4.5 M
Installing for dependencies:
perl                   i686      4:5.10.1-144.el6     base                 9.7 M
perl-Error             noarch    1:0.17015-4.el6      base                 29 k
perl-Git               noarch    1.7.1-9.el6_9        updates              29 k
perl-Module-Pluggable i686      1:3.90-144.el6       base                 41 k
perl-Pod-Escapes       i686      1:1.04-144.el6       base                 33 k
perl-Pod-Simple        i686      1:3.13-144.el6       base                 213 k
perl-libs              i686      4:5.10.1-144.el6     base                 594 k
perl-version           i686      3:0.77-144.el6       base                 52 k
rsync                  i686      3.0.6-12.el6         base                 329 k

Transaction Summary
=====
Install      10 Package(s)

Total download size: 16 M
Installed size: 46 M
Is this ok [y/N]: _
```

Una vez instalado `git`, descargamos los fuentes mediante `git clone [repositorio]`, indicando el repositorio mediante la URL dicha anteriormente.

El comando quedaría así:

```
git clone https://github.com/jhallen/joe-editor.git
```

```
[root@choco ~]# git clone https://github.com/jhallen/joe-editor.git
Initialized empty Git repository in /root/joe-editor/.git/
remote: Counting objects: 4527, done.
remote: Total 4527 (delta 0), reused 0 (delta 0), pack-reused 4527
Receiving objects: 100% (4527/4527), 7.53 MiB | 177 KiB/s, done.
Resolving deltas: 100% (3247/3247), done.
[root@choco ~]# _
```

- **Leyendo la documentación**

Vemos que los fuentes se han descargado correctamente. Estos estarán ubicados en un directorio con el mismo nombre que el repositorio, en el directorio desde donde se haya ejecutado el comando; en este caso, nuestro directorio personal.

Nos situamos en el directorio, y hacemos `ls` para ver sus ficheros

```
[root@choco ~]# cd joe-editor
[root@choco joe-editor]# ls
acinclude.m4  COPYING  INSTALL.AMIGA  po          setup.hint
autojoe       cygbuild  joe            rc          syntax
ChangeLog     docs      Makefile.am    README1.md  tests
charmaps      htdocs    man            README.md   xterm-patch
configure.ac  INSTALL  NEWS.md        runtests    xterm-readme
[root@choco joe-editor]# _
```

Vemos que no hay ningún fichero `configure` ni `Makefile`, aunque hay varios de nombre parecido. Revisamos la documentación para ver el proceso de instalación.

```
[root@choco joe-editor]# less INSTALL_
```

```
Run autotools to build configure scripts:

    ./autojoe

(You might find that you need to install automake and autoconf
first).

=====
Installation procedure
=====

[To create a Cygwin binary distribution, use the 'cygbuild' script
instead of these instructions.]

JOE uses the GNU Automake and Autoconf suites to build itself.

Usually you want JOE to use the terminfo database. JOE needs
a termcap emulation library to do this. In modern versions of
UNIX, this library is part of ncurses so you need the ncurses
library:

    apt-get install ncurses-dev

:_
```

Vemos que el fichero configure se genera mediante el script autojoe .

- **Instalando dependencias**

En la documentación también nos indican que autojoe requiere de las herramientas GNU Automake y Autoconf para funcionar, y que algunas funcionalidades de Joe pueden necesitar de la librería ncurses.

Así que, antes de ejecutar el script, deberemos instalar estas dependencias. Las instalamos mediante yum.

```
Setting up Install Process
Loading mirror speeds from cached hostfile
 * base: ftp.cica.es
 * extras: ftp.cica.es
 * updates: ftp.cica.es
Resolving Dependencies
--> Running transaction check
---> Package autoconf.noarch 0:2.63-5.1.el6 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                Arch          Version           Repository        Size
=====
Installing:
autoconf                noarch        2.63-5.1.el6      base              781 k
Transaction Summary
=====
Install      1 Package(s)

Total download size: 781 k
Installed size: 2.5 M
Is this ok [y/N]: y_
```

```
Setting up Install Process
Loading mirror speeds from cached hostfile
 * base: ftp.cica.es
 * extras: ftp.cica.es
 * updates: ftp.cica.es
Resolving Dependencies
--> Running transaction check
---> Package automake.noarch 0:1.11.1-4.el6 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                Arch          Version           Repository        Size
=====
Installing:
automake                noarch        1.11.1-4.el6      base              550 k
Transaction Summary
=====
Install      1 Package(s)

Total download size: 550 k
Installed size: 1.5 M
Is this ok [y/N]: _
```

```

[root@choco joe-editor]# yum search ncurses
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.cica.es
 * extras: ftp.cica.es
 * updates: ftp.cica.es
base                                     | 3.7 kB      00:00
extras                                 | 3.3 kB      00:00
updates                               | 3.4 kB      00:00
===== N/S Matched: ncurses =====
ncurses.i686 : Ncurses support utilities
ncurses-devel.i686 : Development files for the ncurses library
ncurses-libs.i686 : Ncurses libraries
ncurses-static.i686 : Static libraries for the ncurses library
ncurses-base.i686 : Descriptions of common terminals
ncurses-term.i686 : Terminal descriptions
ocaml-curses.i686 : OCaml bindings for ncurses

Name and summary matches only, use "search all" for everything.
[root@choco joe-editor]# _

```

```

Setting up Install Process
Loading mirror speeds from cached hostfile
 * base: ftp.cica.es
 * extras: ftp.cica.es
 * updates: ftp.cica.es
Resolving Dependencies
--> Running transaction check
--> Package ncurses-devel.i686 0:5.7-4.20090207.el6 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                Arch      Version                               Repository  Size
=====
Installing:
ncurses-devel          i686      5.7-4.20090207.el6                  base       641 k
Transaction Summary
=====
Install      1 Package(s)

Total download size: 641 k
Installed size: 1.7 M
Is this ok [y/N]: _

```

- Ejecutando autojoe

Instaladas las dependencias, pasamos a ejecutar el script autojoe, que nos generará el fichero `configure` necesario para la generación del Makefile.

```
[root@choco joe-editor]# ./autojoe
configure.ac:10: installing './config.guess'
configure.ac:10: installing './config.sub'
configure.ac:15: installing './install-sh'
configure.ac:15: installing './missing'
joe/Makefile.am: installing './depcomp'
[root@choco joe-editor]# _
```

Tras ejecutar el script, vemos que ya tenemos el fichero `configure`, necesario para generar nuestro Makefile.

```
[root@choco joe-editor]# ls
acinclude.m4  config.sub  htdocs      man          runtests
aclocal.m4    configure  INSTALL     missing      setup.hint
autojoe       configure.ac  INSTALL.AMIGA  NEWS.md     syntax
autom4te.cache  COPYING    install-sh   po          tests
ChangeLog      cygbuild   joe          rc           xterm-patch
charmaps       depcomp    Makefile.am  README1.md  xterm-readme
config.guess   docs       Makefile.in  README.md
[root@choco joe-editor]# _
```

- Ejecutando configure

Una vez generado el fichero `configure`, pasamos a ejecutarlo. Este nos generará una salida similar a esta:

```
config.status: creating syntax/ps.jsf
config.status: creating syntax/puppet.jsf
config.status: creating syntax/python.jsf
config.status: creating syntax/rexx.jsf
config.status: creating syntax/ruby.jsf
config.status: creating syntax/sed.jsf
config.status: creating syntax/sh.jsf
config.status: creating syntax/sieve.jsf
config.status: creating syntax/skill.jsf
config.status: creating syntax/sml.jsf
config.status: creating syntax/spec.jsf
config.status: creating syntax/sql.jsf
config.status: creating syntax/tcl.jsf
config.status: creating syntax/tex.jsf
config.status: creating syntax/troff.jsf
config.status: creating syntax/typescript.jsf
config.status: creating syntax/verilog.jsf
config.status: creating syntax/vhdl.jsf
config.status: creating syntax/whitespace.jsf
config.status: creating syntax/xml.jsf
config.status: creating syntax/yaml.jsf
config.status: creating syntax/filename.jsf
config.status: creating joe/autoconf.h
config.status: executing depfiles commands
[root@choco joe-editor]# _
```



Tras ejecutar el fichero *configure*, vemos que se ha generado el Makefile necesario para la compilación e instalación del programa.

```
[root@choco joe-editor]# ls
acinclude.m4      config.status    htdocs           man              setup.hint
aclocal.m4        config.sub       INSTALL          missing          syntax
autojoe           configure       INSTALL.AMIGA   NEWS.md         tests
autom4te.cache    configure.ac     install-sh       po              xterm-patch
ChangeLog         COPYING         joe             rc              xterm-readme
charmaps          cygbuild        Makefile         README1.md
config.guess      depcomp         Makefile.am      README.md
config.log        docs            Makefile.in      runtests
[root@choco joe-editor]# _
```

- **Compilando el programa**

Para compilar el programa, usaremos el comando `make`.

```
[root@choco joe-editor]# make _
```

```
l/share/joe/" -g -O2 -MT options.o -MD -MP -MF .deps/options.Tpo -c -o opti
ons.o options.c
mv -f .deps/options.Tpo .deps/options.Po
gcc -g -O2 -o joe b.o blocks.o bw.o cmd.o hash.o help.o kbd.o macro.o main.o
menu.o path.o poshist.o pw.o queue.o qw.o rc.o regex.o scrn.o tab.o termcap.o tt
y.o tw.o ublock.o uedit.o uerror.o ufile.o uformat.o uisrch.o umath.o undo.o use
arch.o ushell.o utag.o va.o vfile.o vs.o w.o utils.o syntax.o utf8.o selinux.o i
18n.o charmap.o mouse.o lattr.o gettext.o builtin.o builtins.o vt.o mmenu.o stat
e.o options.o -lm -lncurses -lutil
gcc -DHAVE_CONFIG_H -I. -DJOERC="/usr/local/etc/joe/" -DJOEDATA="/usr/loca
l/share/joe/" -g -O2 -MT termidx.o -MD -MP -MF .deps/termidx.Tpo -c -o term
idx.o termidx.c
mv -f .deps/termidx.Tpo .deps/termidx.Po
gcc -g -O2 -o termidx termidx.o -lm -lncurses -lutil
gcc -DHAVE_CONFIG_H -I. -DJOERC="/usr/local/etc/joe/" -DJOEDATA="/usr/loca
l/share/joe/" -g -O2 -MT stringify.o -MD -MP -MF .deps/stringify.Tpo -c -o
stringify.o stringify.c
mv -f .deps/stringify.Tpo .deps/stringify.Po
gcc -g -O2 -o stringify stringify.o -lm -lncurses -lutil
make[2]: Leaving directory `/root/joe-editor/joe'
make[1]: Leaving directory `/root/joe-editor/joe'
make[1]: Entering directory `/root/joe-editor'
make[1]: Nothing to be done for `all-am'.
make[1]: Leaving directory `/root/joe-editor'
[root@choco joe-editor]# _
```

Vemos que la compilación ha finalizado correctamente.

- Instalando el programa

Finalmente, instalamos el programa en el sistema usando `make install`, que copiará los binarios recién compilados a su directorio final, en `/usr/local/bin`

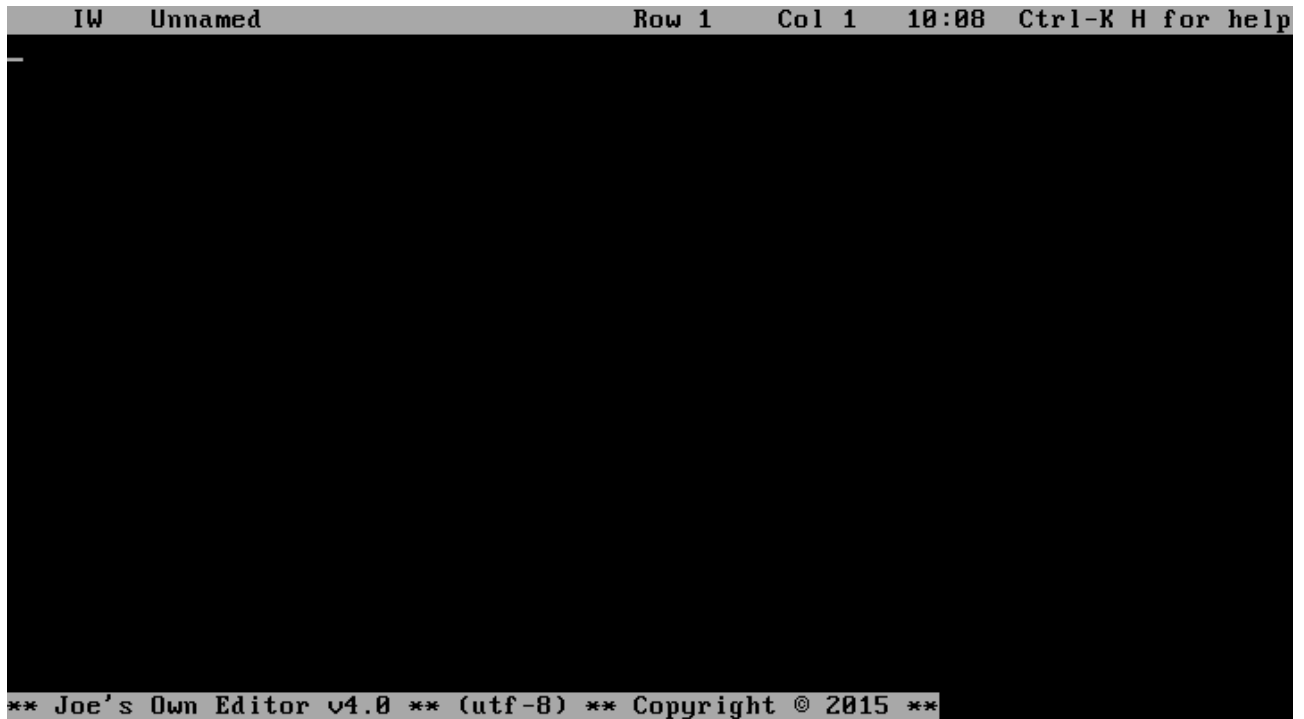
```
make[2]: Entering directory `/root/joe-editor/joe'
test -z "/usr/local/bin" || /bin/mkdir -p "/usr/local/bin"
/usr/bin/install -c joe termidx stringify '/usr/local/bin'
make[2]: Nothing to be done for `install-data-am'.
make[2]: Leaving directory `/root/joe-editor/joe'
make[1]: Leaving directory `/root/joe-editor/joe'
make[1]: Entering directory `/root/joe-editor'
make[2]: Entering directory `/root/joe-editor'
make install-exec-hook
make[3]: Entering directory `/root/joe-editor'
rm -f /usr/local/bin/jmacs /usr/local/bin/jstar /usr/local/bin/rjoe /usr/local/b
in/jpico
rm -f /usr/local/bin/jmacs.exe /usr/local/bin/jstar.exe /usr/local/bin/rjoe.exe
/usr/local/bin/jpico.exe
for i in jmacs jstar rjoe jpico ; do ln -s joe /usr/local/bin/$i ; done
make[3]: Leaving directory `/root/joe-editor'
test -z "/usr/local/share/joe/charmaps" || /bin/mkdir -p "/usr/local/share/joe/c
harmaps"
/usr/bin/install -c -m 644 charmaps/klingon '/usr/local/share/joe/charmaps'
test -z "/usr/local/share/doc/joe" || /bin/mkdir -p "/usr/local/share/doc/joe"
/usr/bin/install -c -m 644 README.md docs/README.old docs/man.md ChangeLog docs
/hacking.md NEWS.md '/usr/local/share/doc/joe'
make[2]: Leaving directory `/root/joe-editor'
make[1]: Leaving directory `/root/joe-editor'
[root@choco joe-editor]# _
```

Vemos que la instalación ha terminado correctamente

- **Probando el programa**

Para probar el programa, simplemente escribimos `joe` en el interprete de comandos.

```
[root@choco joe-editor]# joe_
```



```
IW Unnamed Row 1 Col 1 10:08 Ctrl-K H for help
```

```
** Joe's Own Editor v4.0 ** (utf-8) ** Copyright © 2015 **
```

Vemos que el programa se ha instalado y funciona correctamente.

Podemos comprobar su ruta de instalación con el comando `which`

```
[root@choco joe-editor]# which joe
/usr/local/bin/joe
```

Vemos que el binario está alojado en `/usr/local/bin`, tal como avisaba el `make install`

## Parte 2: Realizando una copia de seguridad

En esta parte, realizaremos una copia de seguridad de todos los ficheros que se han generado en Choco durante la instalación de joe, y transferiremos el fichero resultante a Gamba mediante rsync, donde lo extraeremos para realizar la instalación del programa.

Una vez la extracción haya sido correcta, avisaremos a todos los usuarios de Choco y Gamba para indicarles que esta disponible un nuevo editor.

### 1. Obteniendo lista de ficheros

Para saber los ficheros que se han generado durante la instalación, usaremos el comando `find`, buscando los ficheros que se hayan generado durante el intervalo de tiempo que duró la instalación.

Para definir ese intervalo, usaremos las opciones `-cnewer` y `-ctime`, que nos permiten buscar los ficheros cuya fecha de modificación sea posterior a la de un fichero dado, y que se hayan modificado antes o después de un intervalo de tiempo expresado en minutos.

En nuestro caso, le pasaremos a `cnewer` el fichero principal del repositorio git ubicado en el directorio de joe-editor; y a `ctime` la estimación de minutos pasada desde la instalación, que calculamos en unos 1125 minutos.

```
[root@choco ~]# find / -cnewer joe-editor/.git/ -cmin +1125 > file_list
find: `/proc/1324/task/1324/fd/5': No such file or directory
find: `/proc/1324/task/1324/fdinfo/5': No such file or directory
find: `/proc/1324/fd/5': No such file or directory
find: `/proc/1324/fdinfo/5': No such file or directory
[root@choco ~]# grep /usr/local/bin/joe file_list
/usr/local/bin/joe
[root@choco ~]# _
```

La salida del comando la guardamos en un fichero, con nombre *file\_list*

```
/usr/local/share/joe/syntax/pascal.jsf
/usr/local/share/joe/syntax/haml.jsf
/usr/local/share/joe/syntax/csh.jsf
/usr/local/share/joe/syntax/sml.jsf
/usr/local/share/joe/syntax/conf.jsf
/usr/local/share/joe/syntax/diff.jsf
/usr/local/share/joe/syntax/elixir.jsf
/usr/local/share/joe/syntax/jsf.jsf
/usr/local/share/joe/syntax/python.jsf
/usr/local/share/joe/syntax/git-commit.jsf
/usr/local/share/joe/syntax/fortran.jsf
/usr/local/share/joe/syntax/yaml.jsf
/usr/local/share/joe/syntax/powershell.jsf
/usr/local/share/joe/syntax/cobol.jsf
/usr/local/share/joe/syntax/java.jsf
/usr/local/share/joe/syntax/erlang.jsf
/usr/local/share/joe/syntax/md.jsf
/usr/local/share/joe/syntax/xml.jsf
/usr/local/share/joe/syntax/debian.jsf
/usr/local/share/joe/syntax/perl.jsf
/usr/local/share/joe/syntax/awk.jsf
/usr/local/share/joe/syntax/jsf_check.jsf
/usr/local/share/joe/syntax/erb.jsf
/usr/local/share/joe/syntax/php.jsf
--More--(87%)_
```

Vemos que los resultados obtenidos con `find` son bastante similares a los esperados,

## **2. Generando copia de seguridad**

Ya con la lista de ficheros a respaldar generada, pasaremos a crear la copia de seguridad. Para ello, usaremos el comando `tar`.

Para generar una copia de seguridad de ciertos ficheros y directorios, usamos la sintaxis:

```
tar cvpf [fichero_destino] --one-file-system [lista_directorios]
```

En nuestro caso, el fichero destino lo llamaremos *mybackup.tar* y la lista de ficheros y directorios será la almacenada en el fichero *file\_list*

Para poder generar la lista a partir del fichero, realizaremos una sustitución de comandos con el comando `cat`, pasando el contenido del fichero como parámetros de `tar`.

El comando resultante quedará así:

```
tar cvpf mybackup.tar --one-file-system $(cat file_list)
```

Lo ejecutamos:

```
[root@choco ~]# tar cvpf mybackup.tar --one-file-system $(cat file_list)
```

```
/var/lib/yum/history/history-2018-01-17.sqlite-journal
/var/lib/yum/rpmdb-indexes/
/var/lib/yum/rpmdb-indexes/file-requires
/var/lib/yum/rpmdb-indexes/version
/var/lib/yum/rpmdb-indexes/conflicts
/var/lib/yum/rpmdb-indexes/pkgtops-checksums
/var/lib/yum/rpmdb-indexes/file-requires
/var/lib/yum/rpmdb-indexes/version
/var/lib/yum/rpmdb-indexes/conflicts
/var/lib/yum/rpmdb-indexes/pkgtops-checksums
/var/lib/rpm/Requireversion
/var/lib/rpm/Packages
/var/lib/rpm/Providename
/var/lib/rpm/Shalheader
/var/lib/rpm/Basenames
/var/lib/rpm/Provideversion
/var/lib/rpm/Obsoletename
/var/lib/rpm/Name
/var/lib/rpm/Group
/var/lib/rpm/Filedigests
/var/lib/rpm/Dirnames
/var/lib/rpm/Sigmd5
/var/lib/rpm/Requirename
/var/lib/rpm/Installtid
[root@choco ~]# _
```

Vemos que se ha ejecutado correctamente  
Hacemos un `ls` para comprobarlo.

```
[root@choco ~]# ls -l
total 369972
-rw-----. 1 root root      1094 Jan 17 18:40 anaconda-ks.cfg
-rw-r--r--. 1 root root    45412 Jan 21 17:01 file_list
-rw-r--r--. 1 root root     8437 Jan 17 18:40 install.log
-rw-r--r--. 1 root root     3384 Jan 17 18:40 install.log.syslog
drwxr-xr-x. 13 root root     4096 Jan 20 21:43 joe-editor
drwxr-xr-x. 13 root root     4096 Jan 19 22:47 joe-editor~
-rw-r--r--. 1 root root 378767360 Jan 21 19:06 mybackup.tar
-rw-r--r--. 1 root root         1 Jan 20 01:55 n
[root@choco ~]#
```

Vemos que tenemos un nuevo fichero llamado *mybackup.tar*

Visualizamos su contenido con `tar -tvf [fichero]`

```
[root@choco ~]# tar -tvf mybackup.tar | more_
-rw----- root/root      241 2018-01-20 22:11 root/.joe_state
-rw----- root/root       44 2018-01-20 21:21 root/.lessht
drwxr-xr-x root/root       0 2018-01-20 21:43 root/joe-editor/
-rw-r--r-- root/root    2613 2018-01-20 21:17 root/joe-editor/acinclude.m4
drwxr-xr-x root/root       0 2018-01-20 21:17 root/joe-editor/docs/
-rw-r--r-- root/root    7274 2018-01-20 21:17 root/joe-editor/docs/README.old
-rw-r--r-- root/root   30991 2018-01-20 21:17 root/joe-editor/docs/hacking.md
-rw-r--r-- root/root    4913 2018-01-20 21:17 root/joe-editor/docs/history.md
-rw-r--r-- root/root    3001 2018-01-20 21:17 root/joe-editor/docs/help-system
.html
-rw-r--r-- root/root   116740 2018-01-20 21:17 root/joe-editor/docs/man.md
drwxr-xr-x root/root       0 2018-01-20 21:43 root/joe-editor/po/
-rw-r--r-- root/root   38643 2018-01-20 21:17 root/joe-editor/po/uk.po
-rw-r--r-- root/root   11226 2018-01-20 21:29 root/joe-editor/po/Makefile.in
-rw-r--r-- root/root    1740 2018-01-20 21:17 root/joe-editor/po/HOWTO
-rw-r--r-- root/root   36184 2018-01-20 21:17 root/joe-editor/po/de.po
-rw-r--r-- root/root   38752 2018-01-20 21:17 root/joe-editor/po/ru.po
-rw-r--r-- root/root   35419 2018-01-20 21:17 root/joe-editor/po/fr.po
-rw-r--r-- root/root     139 2018-01-20 21:17 root/joe-editor/po/Makefile.am
-rw-r--r-- root/root   11224 2018-01-20 21:43 root/joe-editor/po/Makefile
-rw-r--r-- root/root   25268 2018-01-20 21:17 root/joe-editor/po/joe.pot
-rwxr-xr-x root/root   44941 2018-01-20 21:29 root/joe-editor/config.guess
-rw-r--r-- root/root   18092 2018-01-20 21:17 root/joe-editor/COPYING
-rwxr-xr-x root/root     869 2018-01-20 21:17 root/joe-editor/cygbuild
--More--
```

El contenido del fichero parece estar correcto.

### 3. Transfiriendo ficheros a Gamba

Una vez generada la copia de seguridad, transferimos el fichero resultante a Gamba. Para ello, usaremos la herramienta `rsync`, que nos permite sincronizar ficheros y directorios entre directorios locales o remotos.

Para transferir ficheros con `rsync` usamos la sintaxis:

```
rsync {opciones} [ruta_origen] [ruta_destino]
```

En el caso de que la ruta destino sea un directorio remoto, podemos acceder a ella mediante `ssh`, indicando la ruta destino como `usuario@host:[ruta]`

Así, el comando quedaría como:

```
rsync {opciones} [ruta_origen] usuario@host:[ruta_destino]
```

Como opción usaremos `-a`, que nos permite generar una copia exacta de nuestro fichero, manteniendo la fecha de creación, los permisos, el usuario y el grupo del fichero en el directorio destino. En el caso de directorios, esta opción también recorrería recursivamente el directorio y mantendría los enlaces simbólicos que hubiera en este.

En nuestro caso, queremos transferir el fichero `mybackup.tar` desde Choco hasta Gamba. Para indicar el host, usaremos su dirección IP, que es `192.168.3.2`. Como usuario, dado que no hemos creado otro, usaremos `root`. La ruta destino también será el directorio `/root`

Así pues, con todo lo anterior, el comando resultante sería:

```
rsync -av mybackup.tar root@192.168.3.2:/root
```

Lo ejecutamos en Choco:

```
[root@choco ~]# rsync -av mybackup.tar root@192.168.3.2:/root/
root@192.168.3.2's password:
sending incremental file list
mybackup.tar

sent 378813677 bytes  received 31 bytes  26125083.31 bytes/sec
total size is 378767360  speedup is 1.00
[root@choco ~]# _
```

Nos pide la clave, así que la ponemos. Tras establecer la conexión, realiza la transmisión del fichero a la máquina destino, en este caso Gamba.

Tras terminar, nos indica el total de bytes transferidos y la velocidad de transmisión.

```
[root@choco ~]# ls -l
total 369976
-rw-----. 1 root root      1094 Jan 17 18:40 anaconda-ks.cfg
drwxr-xr-x. 2 root root      4096 Jan 21 20:35 backup
-rw-r--r--. 1 root root    45412 Jan 21 17:01 file_list
-rw-r--r--. 1 root root      8437 Jan 17 18:40 install.log
-rw-r--r--. 1 root root      3384 Jan 17 18:40 install.log.syslog
drwxr-xr-x. 13 root root      4096 Jan 20 21:43 joe-editor
drwxr-xr-x. 13 root root      4096 Jan 19 22:47 joe-editor~
-rw-r--r--. 1 root root 378767360 Jan 21 19:06 mybackup.tar
-rw-r--r--. 1 root root         1 Jan 20 01:55 n
[root@choco ~]# _
```

Comprobamos que el total de bytes transferidos coincide con el tamaño de nuestro fichero.

Nos vamos a Gamba, y comprobamos si el fichero se ha transferido correctamente

```
[root@gamba ~]# ls -l
total 369912
-rw-----. 1 root root      1094 Jan 17 18:47 anaconda-ks.cfg
-rw-r--r--. 1 root root      8437 Jan 17 18:47 install.log
-rw-r--r--. 1 root root      3384 Jan 17 18:46 install.log.syslog
-rw-r--r--. 1 root root 378767360 Jan 21 19:06 mybackup.tar
[root@gamba ~]# _
```

Vemos que el fichero existe, y que el tamaño coincide con el original. También comprobamos como tanto los permisos como el propietario se han mantenido respecto al fichero original.

Mostramos el contenido del fichero con `tar -tvf`



```

-rw----- root/root      241 2018-01-20 22:11 root/.joe_state
-rw----- root/root       44 2018-01-20 21:21 root/.lessht
drwxr-xr-x root/root       0 2018-01-20 21:43 root/joe-editor/
-rw-r--r-- root/root    2613 2018-01-20 21:17 root/joe-editor/acinclude.m4
drwxr-xr-x root/root       0 2018-01-20 21:17 root/joe-editor/docs/
-rw-r--r-- root/root    7274 2018-01-20 21:17 root/joe-editor/docs/README.old
-rw-r--r-- root/root   30991 2018-01-20 21:17 root/joe-editor/docs/hacking.md
-rw-r--r-- root/root    4913 2018-01-20 21:17 root/joe-editor/docs/history.md
-rw-r--r-- root/root    3001 2018-01-20 21:17 root/joe-editor/docs/help-system
.html
-rw-r--r-- root/root   116740 2018-01-20 21:17 root/joe-editor/docs/man.md
drwxr-xr-x root/root       0 2018-01-20 21:43 root/joe-editor/po/
-rw-r--r-- root/root   38643 2018-01-20 21:17 root/joe-editor/po/uk.po
-rw-r--r-- root/root   11226 2018-01-20 21:29 root/joe-editor/po/Makefile.in
-rw-r--r-- root/root    1740 2018-01-20 21:17 root/joe-editor/po/HOWTO
-rw-r--r-- root/root   36184 2018-01-20 21:17 root/joe-editor/po/de.po
-rw-r--r-- root/root   38752 2018-01-20 21:17 root/joe-editor/po/ru.po
-rw-r--r-- root/root   35419 2018-01-20 21:17 root/joe-editor/po/fr.po
-rw-r--r-- root/root     139 2018-01-20 21:17 root/joe-editor/po/Makefile.am
-rw-r--r-- root/root   11224 2018-01-20 21:43 root/joe-editor/po/Makefile
-rw-r--r-- root/root   25268 2018-01-20 21:17 root/joe-editor/po/joe.pot
-rwxr-xr-x root/root   44941 2018-01-20 21:29 root/joe-editor/config.guess
-rw-r--r-- root/root   18092 2018-01-20 21:17 root/joe-editor/COPYING
-rwxr-xr-x root/root     869 2018-01-20 21:17 root/joe-editor/cygbuild
--More--

```

El contenido parece ser idéntico al del fichero original.

#### **4. Extrayendo la copia de seguridad**

Ya con el fichero transferido correctamente a Gamba, pasamos a descomprimirlo para copiar los ficheros en su ubicación final.

Para extraerlo, usaremos `tar xvf`

Dado que el fichero comprimido solo almacena rutas relativas, debemos situarnos en el directorio raíz antes de extraerlo.

Ejecutamos el comando en Gamba:

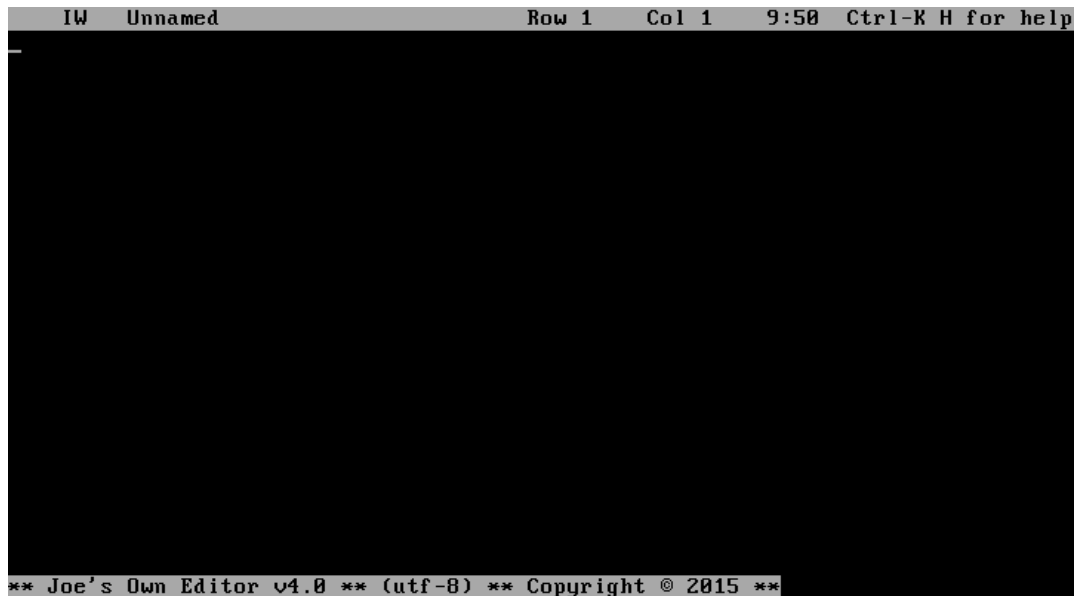
```
[root@gamba ~]# cd /
[root@gamba /]# tar xvf /root/mybackup.tar_

var/lib/yum/history/history-2018-01-17.sqlite-journal
var/lib/yum/rpmdb-indexes/
var/lib/yum/rpmdb-indexes/file-requires
var/lib/yum/rpmdb-indexes/version
var/lib/yum/rpmdb-indexes/conflicts
var/lib/yum/rpmdb-indexes/pkgtups-checksums
var/lib/yum/rpmdb-indexes/file-requires
var/lib/yum/rpmdb-indexes/version
var/lib/yum/rpmdb-indexes/conflicts
var/lib/yum/rpmdb-indexes/pkgtups-checksums
var/lib/rpm/Requireversion
var/lib/rpm/Packages
var/lib/rpm/Providename
var/lib/rpm/Shalheader
var/lib/rpm/Basenames
var/lib/rpm/Provideversion
var/lib/rpm/Obsoletename
var/lib/rpm/Name
var/lib/rpm/Group
var/lib/rpm/Filedigests
var/lib/rpm/Dirnames
var/lib/rpm/Sigmd5
var/lib/rpm/Requirename
var/lib/rpm/Installtid
[root@gamba /]# _
```

Vemos que la extracción termina correctamente.

Una vez han terminado de extraerse los contenidos del fichero, intentamos abrir el editor con el comando `joe`

```
[root@gamba ~]# joe_
```



```
IW Unnamed Row 1 Col 1 9:50 Ctrl-K H for help

** Joe's Own Editor v4.0 ** (utf-8) ** Copyright © 2015 **
```

Vemos que el editor se ha abierto correctamente.

```
[root@gamba ~]# ls
anaconda-ks.cfg install.log install.log.syslog joe-editor mybackup.tar
[root@gamba ~]# yum install ncurses-devel
Loaded plugins: fastestmirror
Setting up Install Process
Loading mirror speeds from cached hostfile
 * base: mirror.airenetworks.es
 * extras: mirror.airenetworks.es
 * updates: mirror.airenetworks.es
Package ncurses-devel-5.7-4.20090207.el6.i686 already installed and latest version
Nothing to do
[root@gamba ~]# yum install automake autoconf
Loaded plugins: fastestmirror
Setting up Install Process
Loading mirror speeds from cached hostfile
 * base: mirror.airenetworks.es
 * extras: mirror.airenetworks.es
 * updates: mirror.airenetworks.es
Package automake-1.11.1-4.el6.noarch already installed and latest version
Package autoconf-2.63-5.1.el6.noarch already installed and latest version
Nothing to do
[root@gamba ~]# _
```

También comprobamos a su vez, que el directorio del repositorio se ha copiado correctamente, y que los paquetes que instalamos durante la compilación también aparecen como instalados dentro de yum

De esta forma, concluimos que la copia de seguridad ha sido exitosa

## **5. Avisando a los usuarios**

Una vez completada la instalación en las máquinas Choco y Gamba, enviamos un mensaje a sus usuarios para informarles de que tienen disponible un nuevo programa en sus equipos.

Para ello, usaremos el comando `wall`, que envía un mensaje de difusión a todos los usuarios (activos o no) dentro de un sistema.

Este comando muestra un mensaje procedente de un fichero, a todos los usuarios (conectados o no) del sistema.

Para enviar el mensaje sin tener que crear un nuevo fichero, usaremos el comando `echo` con una tubería

Enviamos el mensaje en Choco

```
[root@choco ~]# echo "Nuevo programa instalado: Joe's Own Editor" | wall
[root@choco ~]#
Broadcast message from root@choco (Mon Jan 22 00:26:36 2018):

Nuevo programa instalado: Joe's Own Editor
_
```

Enviamos el mensaje en Gamba

```
[root@gamba ~]# echo "Nuevo programa instalado: Joe's Own Editor" | wall
[root@gamba ~]#
Broadcast message from root@gamba (Mon Jan 22 00:28:03 2018):

Nuevo programa instalado: Joe's Own Editor
_
```

## **Parte 3: Programando copias de seguridad incrementales**

En este paso, programaremos copias de seguridad incrementales, para los directorios */home* de Choco y Gamba, que se realizarán de forma diaria.

Las copias de seguridad de Choco se almacenarán en el directorio */backups* de Gamba, y viceversa.

Para programar las copias de seguridad usaremos el comando *cron*.

### **1. Creando los directorios */backup* en Choco y Gamba**

Creemos los directorios */backup* en Choco y Gamba, usando el comando *mkdir*.

```
[root@choco ~]# mkdir /backups
[root@choco ~]# ls /
backups boot etc lib media opt root selinux sys usr
bin dev home lost+found mnt proc sbin srv tmp var
[root@choco ~]# _
```

```
[root@gamba ~]# mkdir /backups
[root@gamba ~]# ls /
backups boot etc lib lost+found mnt proc sbin srv tmp var
bin dev home lib media opt root selinux sys usr
[root@gamba ~]# _
```

Comprobamos que el directorio se ha creado correctamente, haciendo un *ls* del directorio raíz.

### **2. Generando las copias de seguridad**

Para crear las copias de seguridad usaremos el comando *tar*. Para que las copias de seguridad sean incrementales, usaremos la sintaxis:

```
tar --create \
    --file=[archivo_copia_seguridad].tar \
    --listed-incremental=[ruta_archivo_metadatos] \
    [directorio_respalda]
```

Donde en *--file* se indica el nombre del fichero en que se almacenará la copia de seguridad, en *--listed-incremental* se indica la ruta donde se ubicará el fichero de metadatos asociado a la copia de seguridad, seguido de la ruta del directorio que queremos respaldar en nuestra copia de seguridad.

En nuestro caso, al fichero de la copia de seguridad lo nombraremos con la fecha en que se creó.

Para ello, usaremos el comando `date +%F`, que nos dará la fecha completa separada por guiones, en formato YYYY-MM-DD

El archivo de metadatos lo almacenaremos en `/var/log/choco.snar`, en el caso de Choco; y en `/var/log/gamba.snar`, en el caso de Gamba. Este fichero se usará para determinar que ficheros se han modificado desde la última copia de seguridad, y nos permitirá añadir a la copia de seguridad únicamente aquellos ficheros que se hayan modificado desde la anterior copia. En caso de no existir, se realizará una copia de seguridad completa,

El directorio a respaldar será el `/home` de cada máquina.

De esta forma, el comando quedará así:

- En Gamba:

```
tar --create \
    --file="$(date +%F)_gamba.tar" \
    --listed-incremental=/var/log/gamba.snar \
    /home
```

- En Choco:

```
tar --create \
    --file="$(date +%F)_choco.tar" \
    --listed-incremental=/var/log/choco.snar \
    /home
```

La copia de seguridad se creará inicialmente en el directorio `/home` de cada máquina, desde el cual se transferirá a su ubicación final mediante `rsync`.

### 3. Probando el procedimiento en Choco

Una vez establecido el proceso a seguir, lo probaremos en Choco, generando una copia de seguridad de su directorio /home y pasandola a Gamba mediante `rsync`.

Visualizamos el contenido de /home

```
[root@choco ~]# ls /home
[root@choco ~]# _
```

Inicialmente, el directorio /home de Choco esta vacío, así que creamos varios ficheros y directorios para mejorar la prueba.

```
[root@choco ~]# ls -R /home
/home:
A B

/home/A:
fichero

/home/B:
C

/home/B/C:
fichero2
[root@choco ~]# _
```

Probamos el comando en Choco:

```

[root@choco ~]# tar --create --file="$(date +%F)_choco.tar" --listed-incremental="/var/log/choco.snar" /home/
tar: Removing leading '/' from member names
[root@choco ~]# ls
2018-01-25_choco.tar  backup  install.log  joe-editor  mybackup.tar
anaconda-ks.cfg      file_list  install.log.syslog  joe-editor~  n
[root@choco ~]# tar -tvf 2018-01-25_choco.tar
drwxr-xr-x root/root      7 2018-01-25 23:50 home/
drwxr-xr-x root/root     10 2018-01-25 23:51 home/A/
drwxr-xr-x root/root      4 2018-01-25 23:50 home/B/
drwxr-xr-x root/root     11 2018-01-25 23:52 home/B/C/
-rw-r--r-- root/root      4 2018-01-25 23:51 home/A/fichero
-rw-r--r-- root/root      4 2018-01-25 23:52 home/B/C/fichero2
[root@choco ~]# more /var/log/choco.snar
GNU tar-1.23-2
151692084970464768
[root@choco ~]# _

```

Vemos que la copia de seguridad se ha creado correctamente, y que se ha generado un nuevo fichero log en `/var/log/choco.snar`.

Transferimos el fichero a Gamba mediante `rsync`





```

[root@choco ~]# rsync -av 2018-01-25_choco.tar root@192.168.3.2:/backups
root@192.168.3.2's password:
sending incremental file list
2018-01-25_choco.tar

sent 10328 bytes  received 31 bytes  1593.69 bytes/sec
total size is 10240  speedup is 0.99
[root@choco ~]# _

```

Comprobamos desde Gamba si el fichero ha llegado correctamente

```

[root@gamba ~]# ls /backups/
2018-01-25_choco.tar
[root@gamba ~]# tar -tvf /backups/2018-01-25_choco.tar
drwxr-xr-x root/root          7 2018-01-25 23:50 home/
drwxr-xr-x root/root        10 2018-01-25 23:51 home/A/
drwxr-xr-x root/root         4 2018-01-25 23:50 home/B/
drwxr-xr-x root/root        11 2018-01-25 23:52 home/B/C/
-rw-r--r-- root/root         4 2018-01-25 23:51 home/A/fichero
-rw-r--r-- root/root         4 2018-01-25 23:52 home/B/C/fichero2
[root@gamba ~]# _

```

Vemos que el fichero ha llegado correctamente, y su contenido es el esperado.

#### **4. Probando la copia de seguridad incremental**

En el caso anterior, no había ninguna copia de seguridad anterior, así que se generó una copia de seguridad completa.

Ahora crearemos y borraremos algunos ficheros, y generaremos una nueva copia de seguridad incremental, y comprobaremos como los cambios se reflejan en la misma

Borramos el fichero2, alojado en `/home/B/C`, y creamos un nuevo fichero4 alojado en `/home/B`

```

[root@choco ~]# rm /home/B/C/fichero2
rm: remove regular file `/home/B/C/fichero2'? y
[root@choco ~]# echo "abcd" > /home/B/fichero4
[root@choco ~]# ls -R /home
/home:
A  B

/home/A:
fichero

/home/B:
C  fichero4

/home/B/C:
[root@choco ~]# _

```

Generamos una nueva copia de seguridad, y comprobamos los resultados.

```
[root@choco ~]# tar --create --file="$(date +%F)_choco.tar" --listed-incremental="/var/log/choco.snar" /home/
tar: Removing leading '/' from member names
[root@choco ~]# ls
2018-01-25 choco.tar backup install.log.syslog mybackup.tar
2018-01-26 choco.tar file_list joe-editor n
anaconda-ks.cfg install.log joe-editor~
[root@choco ~]# tar -tvf 2018-01-26_choco.tar
drwxr-xr-x root/root          7 2018-01-25 23:50 home/
drwxr-xr-x root/root        10 2018-01-25 23:51 home/A/
drwxr-xr-x root/root        14 2018-01-26 00:12 home/B/
drwxr-xr-x root/root          1 2018-01-26 00:12 home/B/C/
-rw-r--r-- root/root          5 2018-01-26 00:12 home/B/fichero4
[root@choco ~]# more /var/log/choco.snar
GNU tar-1.23-2
1516922132093700010647
[root@choco ~]# _
```

Vemos que el nuevo fichero tar solo contiene el *fichero4*, pese a que *fichero* aún existe; y que ya no incluye a *fichero1*, que ha sido eliminado.

De esta forma, comprobamos que las copias de seguridad incrementales se están generando de forma correcta.

Transferimos el fichero resultante a Gamba

```
[root@choco ~]# rsync -av 2018-01-26_choco.tar root@192.168.3.2:/backups
root@192.168.3.2's password:
sending incremental file list
2018-01-26_choco.tar

sent 10328 bytes  received 31 bytes  2959.71 bytes/sec
total size is 10240  speedup is 0.99
[root@choco ~]# _
```

```
[root@gamba ~]# ls /backups/
2018-01-25_choco.tar  2018-01-26_choco.tar
[root@gamba ~]# tar -tvf /backups/2018-01-26_choco.tar
drwxr-xr-x root/root          7 2018-01-25 23:50 home/
drwxr-xr-x root/root        10 2018-01-25 23:51 home/A/
drwxr-xr-x root/root        14 2018-01-26 00:12 home/B/
drwxr-xr-x root/root         1 2018-01-26 00:12 home/B/C/
-rw-r--r-- root/root         5 2018-01-26 00:12 home/B/fichero4
[root@gamba ~]# _
```

Vemos que el fichero ha llegado correctamente.

## **5. Extrayendo la copia de seguridad**

Una vez generadas las copias de seguridad incrementales, vamos a simular una restauración completa a partir de las dos copias incrementales.

Para extraer los contenidos de una copia de seguridad, usaremos el comando `tar --extract`, con la siguiente sintaxis:

```
tar --extract \
    --listed-incremental=/dev/null \
    --file [archivo_copia].tar
tar --extract \
    --listed-incremental=/dev/null \
    --file [archivo_copia_2].tar
```

Ejecutamos los comandos en Gamba, con los dos ficheros almacenados en `/backups`, simulando una restauración completa del contenido de Choco

Empezamos restaurando el primer fichero

```
[root@gamba ~]# tar --extract --listed-incremental=/dev/null --file /backups/2018-01-25_choco.tar
[root@gamba ~]# ls -R home/
home/:
A B

home/A:
fichero

home/B:
C

home/B/C:
fichero2
```

Vemos que el árbol de ficheros es idéntico al original.

Ahora pasamos a extraer el segundo fichero:

```
[root@gamba ~]# tar --extract --listed-incremental=/dev/null --file /backups/2018-01-26_choco.tar
[root@gamba ~]# ls -R home
home:
A B

home/A:
fichero

home/B:
C fichero4

home/B/C:
[root@gamba ~]# _
```

Vemos que, tras extraer el segundo fichero, el contenido se ha fusionado con el contenido actual de *home*, y que el contenido de *home* es idéntico al árbol de directorios final, creado en el paso 4.

De esta forma corroboramos que la copia de seguridad incremental es correcta.

## **6. Creando el script**

Una vez hemos comprobado que el procedimiento a seguir es correcto, pasamos a generar el script necesario para ejecutarlo. Este script se ejecutará todos los días, generando una copia de seguridad de una de las máquinas y transfiriéndola a la otra.

- **Configurando rsync para transferir ficheros sin clave**

Para crear el script, necesitamos que todos los datos se puedan obtener directamente desde el intérprete, sin necesidad de pedirlos por teclado

Actualmente, `rsync` nos pide la clave de la máquina por teclado, así que tenemos que reconfigurarlo para que nos permita la sincronización sin pedirnos la clave.

Dado que `rsync` establecer la conexión a través de `ssh`, configuraremos este para que nos permita conectar sin clave.

Para ello, debemos generar un par de claves publica/privada, mediante el comando `ssh-keygen`. Este comando nos generará una clave pública, que transferiremos a la máquina destino mediante `ssh-copy-id`

## Generamos la clave pública en Gamba

```
[root@gamba ~]# ssh-keygen -t dsa -b 1024
Generating public/private dsa key pair.
Enter file in which to save the key (/root/.ssh/id_dsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_dsa.
Your public key has been saved in /root/.ssh/id_dsa.pub.
The key fingerprint is:
e3:85:ec:88:ea:75:92:0a:ad:a7:9b:c4:63:f6:7a:20 root@gamba
The key's randomart image is:
+--[ DSA 1024]-----+
|
|
|
|      . .
|     S .
|Eo    o + o
|o*o = o o
|+++ = o
|=**.
```

ssh-keygen almacena la clave generada en .ssh/id\_dsa.pub

Enviamos la clave resultante a Choco mediante `ssh-copy-id`. Para ello, indicamos la ruta de nuestro fichero clave mediante la opción `-i`

```
[root@gamba ~]# ssh-copy-id -i .ssh/id_dsa.pub 192.168.2.2
root@192.168.2.2's password:
Now try logging into the machine, with "ssh '192.168.2.2'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.

[root@gamba ~]# ssh 192.168.2.2
Last login: Thu Jan 25 21:46:47 2018
[root@choco ~]# exit
logout
Connection to 192.168.2.2 closed.
[root@gamba ~]# _
```

Comprobamos que la conexión es exitosa y no nos pide clave

Repetimos el proceso en Choco

```
[root@choco ~]# ssh-keygen -t dsa -b 1024
Generating public/private dsa key pair.
Enter file in which to save the key (/root/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_dsa.
Your public key has been saved in /root/.ssh/id_dsa.pub.
The key fingerprint is:
3e:37:0f:ec:f9:2e:f8:ce:b3:18:55:85:bd:5c:76:71 root@choco
The key's randomart image is:
+--[ DSA 1024]-----+
|           o..E|
|          ... =|
|         .. +.|
|        .  o  |
|       S .    |
|      . o     |
|     +. =     |
|    .Bo=      |
|   .oB*+     |
+-----+
[root@choco ~]# _
```

```
[root@choco ~]# ssh-copy-id -i .ssh/id_dsa.pub 192.168.3.2
root@192.168.3.2's password:
Now try logging into the machine, with "ssh '192.168.3.2'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.

[root@choco ~]# ssh root@192.168.3.2
Last login: Fri Jan 26 01:58:31 2018
[root@gamba ~]# _
```

Comprobamos que podemos conectarnos sin clave.

Comprobamos que podemos transferir ficheros sin clave:

```
[root@choco ~]# rsync -av file_list root@192.168.3.2
sending incremental file list
file_list

sent 45493 bytes  received 31 bytes  91048.00 bytes/sec
total size is 45412  speedup is 1.00
[root@choco ~]# _
```

```
[root@gamba ~]# rsync -av install.log root@192.168.2.2
sending incremental file list
install.log

sent 8516 bytes  received 31 bytes  17094.00 bytes/sec
total size is 8437  speedup is 0.99
[root@gamba ~]# _
```



- **Escribiendo el script**

Una vez con todos los problemas resueltos, copiamos los comandos anteriores en nuestro script

El script, en Choco, quedará así:

```
filename="$(date +"%F")_choco.tar"

tar --create \
    --file=$filename \
    --listed-incremental="/var/log/choco.snar" \
    /home

rsync -av $filename root@192.168.3.2:/backups
rm $filename

echo "Nueva copia de seguridad generada: $filename" | wall
```

Probamos el script:

```
[root@choco ~]# chmod +x backup.sh
[root@choco ~]# ./backup.sh
tar: Removing leading '/' from member names
sending incremental file list
2018-01-26_choco.tar

sent 844 bytes  received 121 bytes  1930.00 bytes/sec
total size is 10240  speedup is 10.61
[root@choco ~]#
Broadcast message from root@choco (Fri Jan 26 02:51:04 2018):

Nueva copia de seguridad generada: 2018-01-26_choco.tar
[root@choco ~]# _
```

Vemos que el script funciona correctamente.

Transferimos el script a Gamba

```
[root@choco ~]# rsync -av backup.sh root@192.168.3.2:/root
sending incremental file list
backup.sh

sent 344 bytes  received 31 bytes  750.00 bytes/sec
total size is 267  speedup is 0.71
[root@choco ~]# _
```

Adaptamos el script en Gamba

```
#!/bin/bash

filename="$(date +"%F")_gamba.tar"

tar --create \
    --file=$filename \
    --listed-incremental="/var/log/gamba.snar" \
    /home

rsync -av $filename root@192.168.2.2:/backups
rm $filename

echo "Nueva copia de seguridad generada: $filename" | wall
```

Probamos el script

```
[root@gamba ~]# chmod +x backup.sh
[root@gamba ~]# ./backup.sh
tar: Removing leading '/' from member names
sending incremental file list
2018-01-26_gamba.tar

sent 10328 bytes  received 31 bytes  20718.00 bytes/sec
total size is 10240  speedup is 0.99
[root@gamba ~]#
Broadcast message from root@gamba (Fri Jan 26 02:57:13 2018):

Nueva copia de seguridad generada: 2018-01-26_gamba.tar

[root@gamba ~]# _
```

```
[root@choco ~]# rsync -av backup.sh root@192.168.3.2:/root
sending incremental file list
backup.sh

sent 344 bytes  received 31 bytes  750.00 bytes/sec
total size is 267  speedup is 0.71
[root@choco ~]# ls /backups/
2018-01-26_gamba.tar
[root@choco ~]# _
```

Vemos que Choco ha guardado nuestra copia de seguridad de Gamba, y que el script es correcto.

## **7. Programando el script con cron**

Ya con el script terminado, vamos a programarlo para que se ejecute una vez al día. Para ello, usaremos la herramienta `cr on`, que nos permite programar tareas periódicas.

Para añadir tareas a cron, usaremos el comando `crontab -e`, que nos abrirá un editor en el que añadir la periodicidad de la tarea y el script asociado a la misma.

La sintaxis es:

```
#periodicidad [ruta_script]
```

Para indicar la periodicidad, tenemos los siguientes atajos:

`@reboot`: se ejecuta una única vez al inicio.

`@yearly/@annually`: ejecutar cada año.

`@monthly`: ejecutar una vez al mes.

`@weekly`: una vez a la semana.

`@daily/@midnight`: una vez al día.

`@hourly`: cada hora.

También podemos indicarla de forma mas precisa mediante el uso de los 5 asteriscos:

```
* * * * * script.sh
```

De izquierda a derecha, los asteriscos representan:

Minutos: de 0 a 59.

Horas: de 0 a 23.

Día del mes: de 1 a 31.

Mes: de 1 a 12.

Día de la semana: de 0 a 6, siendo 0 el domingo.

Si se deja un asterisco, quiere decir "cada" minuto, hora, día de mes, mes o día de la semana

Si queremos precisar una periodicidad determinada, solo tenemos que cambiar su correspondiente asterisco por el número correspondiente

En nuestro caso, como simplemente queremos ejecutarlo una vez al día, usaremos el atajo @daily

De esta forma, nuestra tarea quedará como:

@daily /root/backup.sh

Programamos la tarea con crontab -e

```
[root@gamba ~]# crontab -e_
@daily /root/backup.sh:
-
-
"/tmp/crontab.xx4aA1" 1L, 24C written
crontab: installing new crontab
[root@gamba ~]#
```

Tras guardar el fichero, cron nos avisa de que la tarea ya ha sido programada. Lo comprobamos con crontab -l

```
[root@gamba ~]# crontab -l
@daily /root/backup.sh:
[root@gamba ~]# _
```

Vemos que la tarea ha sido programada correctamente. Repetimos el proceso en Choco y comprobamos

```
[root@choco ~]# crontab -l
@daily /root/backup.sh
[root@choco ~]# _
```

Y, con esto, finalmente queda completado nuestro generador de copias de seguridad

## **Conclusiones**

La instalación de un programa desde el código fuente nos permite tener instalaciones con un grado de personalización mayor a la ofrecida por el sistema.

A cambio, aunque el proceso suele ser sencillo, puede originar múltiples complicaciones, tanto para obtener las librerías necesarias como para realizar el propio proceso de instalación; dado que los desarrolladores pueden haber definido diferentes herramientas para generar los ficheros de compilación e instalación (`Makefile`, `cmake`, `qmake`... etc). Por esta razón, es muy importante leer la documentación del programa antes de compilar, que nos indicará las dependencias necesarias y el procedimiento a seguir para la compilación e instalación.

En este caso, nuestro programa no disponía de fichero `configure` ni `Makefile`, sino que había que generarlos a partir de un script del mismo directorio, y debíamos instalar varios programas y dependencias para que este script pudiera funcionar.

Las copias de seguridad son una herramienta muy útil, que nos permite replicar el árbol de ficheros del sistema (o parte de él) en una fecha dada, para poder restaurarlo posteriormente.

Estas copias deben ser almacenadas en una máquina distinta a la original, para lo cual debemos transferir la información de forma fiable a otra máquina. Herramientas como `rsync` nos facilitan esta tarea, permitiéndonos transferir ficheros o directorios a otra máquina remota, e incluso permitiéndonos mantener el propietario y los permisos originales del mismo.

En este caso, hemos conseguido replicar la instalación de un programa, con todas las librerías y dependencias instaladas durante el proceso, en otra máquina diferente a la original; obteniendo un resultado idéntico a la instalación original.

La planificación adecuada de un sistema de copias de seguridad nos permitirá restablecer el sistema (o parte de él) ante un posible fallo.

Las copias de seguridad incrementales nos permiten respaldar la información de forma mas eficiente, solo añadiendo los cambios que se han realizado desde la última copia de seguridad.

A cambio, la tarea de restauración se vuelve mas compleja, puesto que en algunos casos tendremos que recorrer varios ficheros para encontrar el que realmente queremos restaurar o; en el caso de tener que realizar una restauración completa, tendremos que restaurar múltiples ficheros en lugar de uno solo (respecto a la copia de seguridad completa)

En este caso, hemos implementado un sistema de copias de seguridad periódicas, basado en copias de seguridad incrementales que se generan de forma diaria, y se transfieren a otra máquina mediante `rsync`. Una vez obtenidas dos copias de seguridad, hemos simulado la restauración completa del sistema mediante esas copias de seguridad, con resultado exitoso.