Un kernel para el AA1

1 Introducción

En realidad aquí explico como compilar un *kernel* de Linux, en general. Quiero decir que la diferencia entre que el *kernel* aquí producido sirva para el acer aspire one (AA1), o que sirva para otra máquina, depende enteramente del fichero de configuración, del que aquí nada documento.

El parámetro `ARCH="i386"` es necesario si, como es mi caso, hay que compilar un *kernel* para un PC de 32 bits, en un PC de 64 bits.

2 El script

La tarea está semi-automatizada en el *script* `makelinux.sh`, que iremos explicando por partes.

El comienzo sirve para asegurarse de que es el usuario `root` quien ejecuta el *script*, para establecer la versión del *kernel* usado, y para dar un apellido al *kernel* creado.

`../input/makelinux.sh`

```
1  #!/bin/bash
2
3  if [ $(whoami) != 'root' ]
4  then
5   echo 'Only the root can run this script!'
6   exit 1
7  fi
8
9  KERNEL="linux-2.6.34"
10 VERSION="$1"
11
12 ###
```

A continuación se establecen y se crean, si es necesario, los directorios de trabajo.

`../input/makelinux.sh`

```
13 #DIRS
14
15 CDIR=$(dirname $(readlink -f $0))
16
17 if [ -d /mnt/SATA-Projects/AA1 ]
18 then
19  RDIR="/mnt/SATA-Projects/AA1"
20 else
21  if [ ! -d /tmp/AA1 ] ; then mkdir /tmp/AA1 ; fi
22  RDIR="/tmp/AA1"
23 fi
```

```
24
25 cd ${RDIR}
26 if [ ! -d downloads ] ; then mkdir downloads ; fi
27 if [ ! -d CDroot/boot ] ; then mkdir -p CDroot/boot ; fi
28 if [ ! -d working ] ; then mkdir working ; fi
29
30 ###
```

Lo siguiente es descargar las fuentes del *kernel*. Sólo hay que descargarlo una vez, así que se comprueba si está ya descargado.

../input/makelinux.sh

```
31 #WGET
32
33 # Getting the kernel, if needed
34 cd ${RDIR}/downloads
35 if [ ! -f ${KERNEL}.tar.bz2 ] ; then
36  echo 'Getting the kernel ...'
37  wget ftp://ftp.kernel.org/pub/linux/kernel/v2.6/${KERNEL}.tar.bz2
38  cd ${RDIR}/working
39  tar xjf ../downloads/${KERNEL}.tar.bz2
40 fi
41
42 ###
```

Entonces se configuran las opciones.

../input/makelinux.sh

```
43 #CONFIG
44
45 echo 'Configuring ...'
46 cd ${RDIR}
47 if [ ! -f working/${KERNEL}/.config ] ; then
48  cp ${CDIR}/${KERNEL}${VERSION}.config working/${KERNEL}/.config
49 fi
50 cd ${RDIR}/working/${KERNEL}
51 # make mrproper
52 make ARCH="i386" INSTALL_MOD_PATH="${RDIR}/CDroot" menuconfig
53 # make ARCH="i386" INSTALL_MOD_PATH="${RDIR}/CDroot" oldconfig
54 if [ "$(diff -q .config.old .config)" != "" ] ; then
55  echo "   .config was changed, copying to ${CDIR} as new.config"
56  cp .config ${CDIR}/new.config
57 fi
58
59 ###
```

Y, para terminar, se compila.

../input/makelinux.sh

```
60 #MAKE
61
62 echo 'Compiling ...'
63 make ARCH="i386" INSTALL_MOD_PATH="${RDIR}/CDroot" bzImage modules
64 echo 'Installing ...'
65 make ARCH="i386" INSTALL_MOD_PATH="${RDIR}/CDroot" modules_install
66 cd ${RDIR}/working/${KERNEL}
67 cp arch/i386/boot/bzImage ${RDIR}/CDroot/boot/vmlinuz-${KERNEL}${VERSION}
```

```
68 exit
69
70 ###
```

3 Uso

Si `KERNEL="linux-2.6.33.2"` y se quiere compilar un *kernel* `linux-2.6.33.2` para el AA1, entonces hay que tener un fichero de configuración adecuado de nombre `linux-2.6.33.2-AA1.config` en `${CDIR}`, y hacer:

```
# cd /home/papa/src/projects/AA1
# input/makelinux.sh -AA1
```

Los resultados son:

El *kernel*: `${RDIR}/CDroot/boot/vmlinuz-linux-2.6.33.2-AA1`
Los módulos: `${RDIR}/CDroot/lib/modules/2.6.33.2-AA1/`

4 Asuntos resueltos

4.1 Control de versiones

Necesitaba automatizar el control de las versiones de los kerneles creados, básicamente de los ficheros de configuración que gobiernan la compilación. Pero se me ha ocurrido pensar a lo grande. La solución es usar `git`, un verdadero controlador de versiones, para todos los proyectos.

Con `git` podré separar el control de la creación del kernel. No tendré que ir dando nombres distintos sólo para poder dar marcha atrás, y otros trucos similares.

Así que eso es lo que he hecho. La maniobra ha sido la habitual en estos casos:

```
$ cd ~/src/projects
$ git-init
$ gedit .gitignore
$ git-add .
$ git-status
$ git-commit -m "Versión inicial"
```

El `git-status` es para comprobar que los ficheros bajo control son los que tienen que ser, y que `.gitignore` es como debe.

Hay que tener cuidado con el usuario que se usa para menejar `git`. El caso es que para ejecutar el *script* obligo a ser `root` (¿es completamente necesario?), pero si se maneja `git` siendo `root`, entonces `git` crea objetos que sólo puede manejar el `root`, y esto no me interesa. En caso de problemas, la solución es:

```
# cd /home/papa/src/projects/.git
# chown -R papa:papa *
```

4.2 Limpiar

Limpiar lo que sea menester cuando sea menester. Aquí la solución es traer a casa, a un subdirectorio de `~`, los directorios creativos, básicamente `doc` e `input`, y dejar los productos creados en otro, `/mnt/SATA-Projects` en mi caso, `/tmp` en general. El

directorio de productos creados puede borrarse sin remordimientos, ya que podría ser rehecho.