

Ingeniería de Servidores (2015-2016)
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

Cuestiones Opcionales

Marta Gómez Macías

21 de enero de 2016

ÍNDICE

1. Práctica 1	3
1.1. Muestre (con capturas de pantalla) cómo ha comprobado que el RAID1 funciona.	3
1.2. ¿Qué relación hay entre los atajos de teclado de emacs y los de la consola de bash? ¿y entre los de vi y las páginas del manual?	3
2. Práctica 2	5
2.1. ¿Qué gestores utiliza OpenSuse?	5
2.2. Instale y pruebe terminator. Con screen, pruebe su funcionamiento dejando sesiones ssh abiertas en el servidor y recuperándolas posteriormente	5
2.3. Instale el servicio fail2ban y pruebe su funcionamiento	6
2.6. Muestre un ejemplo de uso para awk	8
3. Práctica 3	8
3.1. Indique qué comandos ha utilizado para realizar la sustitución del disco RAID1 dañado por uno nuevo así como capturas de pantalla del proceso de reconstrucción del RAID.	8
3.2. Instale Nagios en su sistema (el que prefiera) documentando el proceso y muestre el resultado de la monitorización de su sistema comentando qué aparece.	9
3.2.1. Instalación	9
3.2.2. Monitorización del sistema	11
3.3. Acceda a la demo online de Ganglia de WikiMedia y haga lo mismo que hizo con Munin	13
3.9. Escriba un script en python y analice su comportamiento usando el profiler presentado	13
4. Práctica 4	15
4.1. Seleccione, instale y ejecute un benchmark, comente los resultados. Atención: no es lo mismo un benchmark que una suite, instale un benchmark	15
4.3. Lea el artículo de comparación entre Jmeter y Gatling elaborado por la empresa Flood.io y elabore un breve resumen	16

ÍNDICE DE FIGURAS

1.1. Pantalla al arrancar por primera vez habiendo quitado el disco 1	3
1.2. Pantalla al iniciarse initramfs y pulsar tabulador	4
1.3. Comprobación del estado del RAID	4
1.4. Activación del RAID y su posterior comprobación	4
1.5. Comandos para moverse a través de una página de manual	5
2.1. Prueba con terminator	6
2.2. Listando las sesiones de screen actualmente abiertas	6
2.3. Haciendo un split en la sesión de screen actual	7
2.4. Ejemplo de uso del comando awk	8

3.1. Configuración de la máquina virtual tras añadir un nuevo disco	8
3.2. Mensaje tras añadir un nuevo disco duro en caliente	9
3.3. Consultando cómo está el disco particionado y los discos que tenemos.	9
3.4. Consultando cómo ha quedado particionado el disco tras añadir el disco nuevo al RAID	10
3.5. Página inicial de Nagios	11
3.6. Modificando algunos parámetros de Nagios antes de iniciar el sistema por primera vez	11
3.7. Interfaz de Nagios al acceder al sistema	11
3.8. Monitorizando el estado de los servicios del sistema y del propio sistema	12
3.9. Monitorizando el uso de swap en el sistema	12
3.10. Monitorizando la “salud” del sistema	12
3.11. Información general del grid de Wikimedia	13
3.12. Información general sobre los servidores de aplicación del clúster codfw	14
3.13. Información general sobre uno de los hosts del clúster codfw	14
3.14. Salida obtenida tras ejecutar <i>cProfile</i> sobre un script Python	15
4.1. Salida del benchmark x264 mientras realiza sus pruebas	16

1. PRÁCTICA 1

1.1. MUESTRE (CON CAPTURAS DE PANTALLA) CÓMO HA COMPROBADO QUE EL RAID1 FUNCIONA.

Tras eliminar el disco 1 de la máquina virtual e iniciar el sistema, comprobamos que al arrancar obtenemos la pantalla de la [Figura 1.1](#)

```
Begin: Loading essential drivers ... [ 13.408148] md: linear personality regis
tered for level -1
[ 13.410491] md: multipath personality registered for level -4
[ 13.412627] md: raid0 personality registered for level 0
[ 13.415538] md: raid1 personality registered for level 1
[ 13.489567] raid6: sse2x1 7835 MB/s
[ 13.569556] raid6: sse2x2 9695 MB/s
[ 13.643289] raid6: sse2x4 10247 MB/s
[ 13.643344] raid6: using algorithm sse2x4 (10247 MB/s)
[ 13.643391] raid6: using ssse3x2 recovery algorithm
[ 13.644813] xor: automatically using best checksumming function:
[ 13.739555] avx : 36796.000 MB/sec
[ 13.740824] async_tx: api initialized (async)
[ 13.748840] md: raid6 personality registered for level 6
[ 13.748897] md: raid5 personality registered for level 5
[ 13.748945] md: raid4 personality registered for level 4
[ 13.754424] md: raid10 personality registered for level 10
done.
Begin: Running /scripts/init-premount ... done.
Begin: Mounting root file system ... Begin: Running /scripts/local-top ... [ 1
3.767028] random: lvm urandom read with 25 bits of entropy available
Reading all physical volumes. This may take a while...
No volume groups found
No volume groups found
Begin: Waiting for encrypted source device... ...
```

Figura 1.1: Pantalla al arrancar por primera vez habiendo quitado el disco 1

Tras esto, se iniciará el modo `initramfs`, en el que podremos ejecutar las instrucciones que se ven en la [Figura 1.2](#) (obtenidas al pulsar el tabulador)

Comprobamos, en primer lugar, el estado actual en el que se encuentra el RAID consultado el archivo `mdstat`. Como se ve en la [Figura 1.3](#), vemos que está desactivado.

Para activarlo, vamos a ejecutar `mdadm`, para ello ejecutamos tal y como se indica en las páginas man de `mdadm` ([9]):

```
mdadm -R /dev/md0
```

Lo ejecutamos sobre `md0`, ya que estamos trabajando sobre el dispositivo RAID. Así, tal y como se ve en la [Figura 1.4](#), ya tenemos nuestro dispositivo RAID activado.

Para terminar, pulsamos **Control+D** para seguir arrancando el sistema.

1.2. ¿QUÉ RELACIÓN HAY ENTRE LOS ATAJO DE TECLADO DE EMACS Y LOS DE LA CONSOLA DE BASH? ¿Y ENTRE LOS DE VI Y LAS PÁGINAS DEL MANUAL?

Tal y como se dice en [2], los atajos de teclado tanto en `bash` como en `emacs` son los mismos. También en [2] nos ponen unos ejemplos de atajos de teclado de `emacs` para la consola de `bash`:

♡ CTRL-P: ir al comando anteriormente ejecutado en el historial

```

[ 10.303991] ata4.00: 16777216 sectors, multi 128: LBA48 NCQ (depth 31/32)
[ 10.305624] ata4.00: configured for UDMA/133
[ 10.307320] scsi 3:0:0:0: Direct-Access ATA UBOX HARDDISK 1.0 PQ
: 0 ANSI: 5
[ 10.310644] sd 3:0:0:0: [sda] 16777216 512-byte logical blocks: (8.58 GB/8.00
GiB)
[ 10.313943] sd 3:0:0:0: Attached scsi generic sg1 type 0
[ 10.314970] sd 3:0:0:0: [sda] Write Protect is off
[ 10.316004] sd 3:0:0:0: [sda] Write cache: enabled, read cache: enabled, does
n't support DPO or FUA
[ 10.326598] sda: sda1
[ 10.329861] sd 3:0:0:0: [sda] Attached SCSI disk
[ 10.382332] md: bind<sda1>
[ 12.271071] floppy0: no floppy controllers found

BusyBox v1.21.1 (Ubuntu 1:1.21.0-1ubuntu1) built-in shell (ash)
Enter 'help' for a list of built-in commands.

(initramfs)
biosdevname      dumpe2fs         losetup          nfsmount         run-init
blkid             fstype           lvm              ntfs-3g          setfont
busybox           halt             mdadm            pivot_root       sh
cpio              hwclock          mdmon            plymouth         sleep
cryptsetup        insmod           modprobe         plymouthd        udevadm
date              ipconfig         mount            poweroff         vgchange
dd                kbd_mode         mount.fuse       reboot           wait-for-root
dmesg             kmod             mount.ntfs       resume
dmsetup           loadkeys         mount.ntfs-3g    rmdir
(initramfs)

```

Figura 1.2: Pantalla al iniciarse initramfs y pulsar tabulador

```

(initramfs) cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [ra
id10]
md0 : inactive sda1[1](S)
      8382464 blocks super 1.2

unused devices: <none>
(initramfs)

```

Figura 1.3: Comprobación del estado del RAID

```

(initramfs) mdadm -R /dev/md0
mdadm: CREATE user root not found
mdadm: CREATE group disk not found
[ 812.135212] md/raid1:md0: active with 1 out of 2 mirrors
[ 812.136211] md0: detected capacity change from 0 to 8583577600
mdadm: started /dev/md0
[ 812.153025] md0: unknown partition table
(initramfs) cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [ra
id10]
md0 : active raid1 sda1[1]
      8382400 blocks super 1.2 [2/1] [_U]

unused devices: <none>
(initramfs)

```

Figura 1.4: Activación del RAID y su posterior comprobación

- ♡ CTRL-N: ir al siguiente comando ejecutado en el historial
- ♡ CTRL-R: buscar en el historial de comandos ejecutados al revés
- ♡ CTRL-S: según la página es buscar en el historial de comandos, pero a mí me suspende la terminal y tengo que pulsar CTRL-Q para volverla a iniciar.
- ♡ CTRL-A: mover el cursor al inicio de la línea.
- ♡ CTRL-E: mover el cursor al final de la línea

- ♡ CTRL-W: eliminar la última palabra en la que se encuentra el cursor. Por ejemplo, en el comando `bibtex citas`, eliminaría la palabra `citas`.
- ♡ ALT-D: eliminar la siguiente palabra a la que apunta el cursor. Por ejemplo, si tenemos el cursor al inicio de la línea, se eliminaría la primera palabra.
- ♡ CTRL-F: mover el cursor un carácter hacia delante.
- ♡ CTRL-B: mover el cursor un carácter hacia atrás.
- ♡ ALT-F: mover el cursor una palabra hacia delante.
- ♡ ALT-B: mover el cursor una palabra hacia detrás.
- ♡ ALT-_: escribir la última palabra que has escrito en el historial.

Lo mismo pasa con `vi` y las `man pages`. Por ejemplo, en la [Figura 1.5](#) vemos los comandos para movernos por una página de manual, pero, esos mismos comandos también nos sirven para movernos por un archivo abierto con `vi`, pero combiándolos con la tecla control.

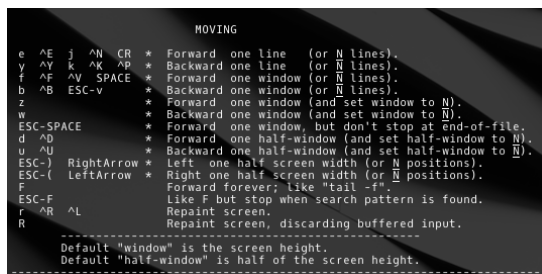


Figura 1.5: Comandos para moverse a través de una página de manual

2. PRÁCTICA 2

2.1. ¿QUÉ GESTORES UTILIZA OPENSUSE?

En la sección *Gestor de paquetes* de [\[5\]](#) se explica que hay dos gestores de paquetes en openSUSE:

- **YaST**: es un gestor de paquetes con interfaz gráfica.
- **Zypper**: es un gestor de paquetes desde la línea de comandos. En [\[6\]](#) podemos encontrar una guía de uso de este gestor de paquetes.

2.2. INSTALE Y PRUEBE TERMINATOR. CON SCREEN, PRUEBE SU FUNCIONAMIENTO DEJANDO SESIONES SSH ABIERTAS EN EL SERVIDOR Y RECUPERÁNDOLAS POSTERIORMENTE

Mi experiencia con terminator se resume en la [Figura 2.1](#). He probado a partir la pantalla, poner un perfil distinto en cada división y, trabajar en tareas independientes en cada terminal de forma paralela.

En [\[1\]](#), se explica un funcionamiento básico de screen. En nuestro caso, tendremos dos sesiones de screen abiertas.

Probamos a abrir una sesión ssh en la sesión 1, y pulsando **Control+a+d** la suspendemos. Tras esto, se cierra la sesión, pero podemos listar las sesiones de screen actualmente en funcionamiento con el comando:

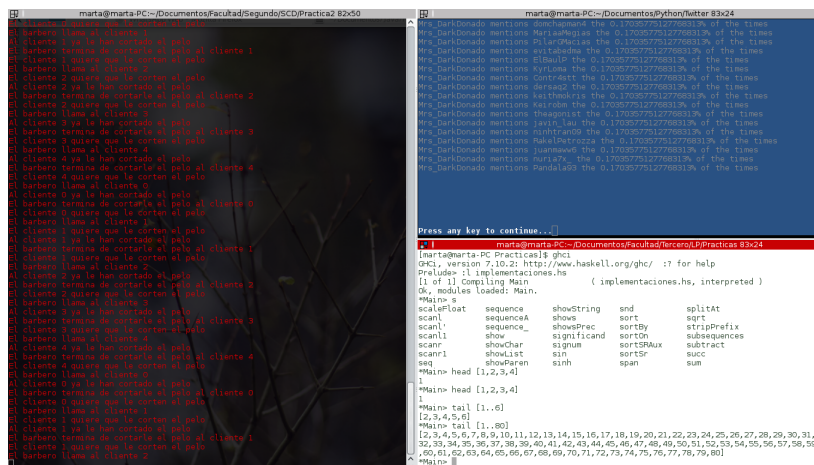


Figura 2.1: Prueba con terminator

```
— listando las sesiones de screen actuales —
screen -ls
```

```
[marta@localhost ~]$ screen -ls
There are screens on:
      4280.pts-2.localhost      (Attached)
      4203.pts-1.localhost      (Detached)
2 Sockets in /var/run/screen/S-marta.
```

Figura 2.2: Listando las sesiones de `screen` actualmente abiertas

Como se ve en la [Figura 2.2](#) tenemos una sesión suspendida. Para recuperarla usamos la opción `-r` acompañada del ID de dicha sesión:

```
— Reanudando la sesión ssh que teníamos en screen —
screen -r 4203
```

También probamos el modo de *split window*. Para ello, pulsamos **Control+a+S**, tras esto, nos movemos a la nueva ventana que hemos hecho con **Control+a+Tab** y una vez ahí pulsamos **Control+a+c** para empezar una nueva sesión de screen. El resultado es el que se ve en la [Figura 2.3](#).

2.3. INSTALE EL SERVICIO fail2ban Y PRUEBE SU FUNCIONAMIENTO

Para instalar `fail2ban` en Ubuntu Server ejecutamos el comando bien conocido:

```
— Instalación de fail2ban en Ubuntu Server —
sudo apt-get install fail2ban
```

Una vez hecho esto, consultamos [\[3\]](#) para saber cómo se usa y cómo se configura. En primer lugar tenemos un cliente y un servidor:

— `fail2ban-client`: es el *frontend* de Fail2ban. Se conecta al servidor definido en el archivo socket y le envía comandos para configurar y operar el servidor. El cliente puede leer archivos de configuración o usarse para mandar órdenes al servidor. Sus opciones son:

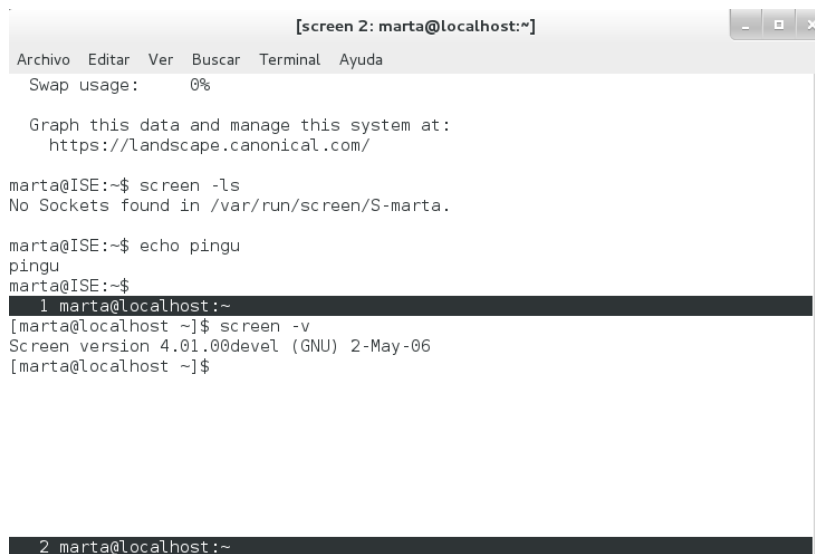


Figura 2.3: Haciendo un split en la sesión de screen actual

- **-c** <DIR>: directorio de configuración. Por defecto */etc/fail2ban*
 - **-s** <FILE>: ruta del socket
 - **-d**: configuración “*dump*”, para depurar.
 - **-i**: modo interactivo
 - **-v**: aumenta la cantidad de información que nos muestra el programa
 - **-q**: disminuye la cantidad de información que nos muestra el programa
 - **-x**: fuerza la ejecución del servidor
 - **-h**: ayuda
 - **-V**: muestra la versión
- **fail2ban-server**: el servidor inicialmente no tiene definida ninguna “cárcel”. No debe usarse directamente, excepto cuando se está depurando. Dispone de las siguientes opciones:
- **-b**: empieza el programa en segundo plano
 - **-f**: empieza el programa en primer plano
 - **-s** <FILE>: ruta del socket
 - **-x**: fuerza la ejecución del servidor
 - **-h**: muestra un mensaje de ayuda
 - **-V**: muestra la versión

Para hacer una simple prueba, iniciamos **fail2ban** con la configuración por defecto:

Iniciamos fail2ban

```
sudo fail2ban-client -v start
```

Como información nos da que el archivo socket en uso es */var/run/fail2ban/fail2ban.sock*.

Si usamos la opción **-i**, nos saldrá algo parecido a cuando usamos **ftp** para introducir comandos, **fail2ban**.

Si miramos el archivo *jail.conf*, podemos configurar cosas tales como las IP que no queremos banear, el tiempo que un host queda baneado, los parámetros para banear a un host (número de peticiones que genera en un tiempo determinado), etc.

También podemos configurar los distintos servicios que queremos controlar, por ejemplo: **ssh**, **xinetd**, **apache**, **vsftpd**, etc.

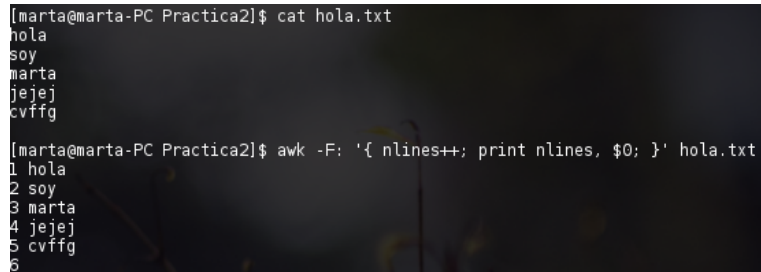
2.6. MUESTRE UN EJEMPLO DE USO PARA awk

En [8] se ponen algunos ejemplos de uso de `awk`. Un ejemplo es *preceder cada línea de su número en un archivo*. Esto se haría con el siguiente comando `awk`:

Añadiendo a cada línea su número de línea

```
awk -F: '{ nlines++; print nlines, $0; }' hola.txt
```

Un ejemplo de uso se ve en la [Figura 2.4](#).



```
[marta@marta-PC Practica2]$ cat hola.txt
hola
soy
marta
jeje
cvffg

[marta@marta-PC Practica2]$ awk -F: '{ nlines++; print nlines, $0; }' hola.txt
1 hola
2 soy
3 marta
4 jeje
5 cvffg
```

Figura 2.4: Ejemplo de uso del comando `awk`

3. PRÁCTICA 3

3.1. INDIQUE QUÉ COMANDOS HA UTILIZADO PARA REALIZAR LA SUSTITUCIÓN DEL DISCO RAID1 DAÑADO POR UNO NUEVO ASÍ COMO CAPTURAS DE PANTALLA DEL PROCESO DE RECONSTRUCCIÓN DEL RAID.

Para hacerlo vamos a seguir los pasos indicados en el guión de prácticas:

1. En primer lugar, vamos a la configuración de la máquina virtual y añadimos un segundo disco. Nos tiene que quedar como se ve en la [Figura 3.1](#).

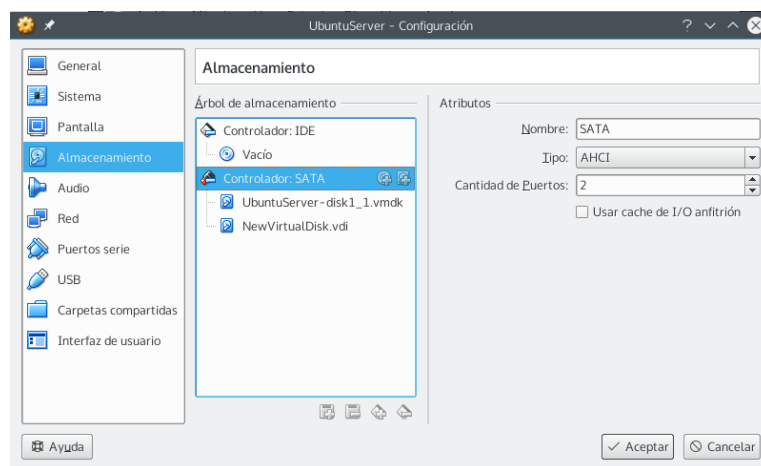


Figura 3.1: Configuración de la máquina virtual tras añadir un nuevo disco

Tras añadir el disco, veremos el mensaje que se ve en la [Figura 3.2](#).

```

marta@ISE:~$ [ 577.737061] ata4: exception Emask 0x10 SAct 0x0 SErr 0x4010000 a
ction 0xe frozen
[ 577.737274] ata4: irq_stat 0x80400040, connection status changed
[ 577.737352] ata4: SError: { PHYRdyChg DevExch }

```

Figura 3.2: Mensaje tras añadir un nuevo disco duro en caliente

2. Usamos el comando `mdadm` para eliminar el disco defectuoso del RAID y añadir el nuevo. Consultando [9], vemos que para eliminar dispositivos del RAID podemos usar la opción `-remove` y para añadir, la opción `-add`.

Primero, tenemos que consultar los discos que tenemos en nuestro ordenador, para ello usamos el comando `lsblk`. Obtenemos el output de la Figura 3.3, en el cual vemos que el disco actual es el `sda1` y el que hemos insertado nuevo, `sdb`. También hemos consultado todos los dispositivos que tenemos en el sistema para saber cuál eliminar, en este caso sería `sda`.

```

marta@ISE:~$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda                                  8:0      0   8G  0 disk
├─sda1                              8:1      0   8G  0 part
│   └─md0                            9:0      0   8G  0 raid1
│       ├─HD-arranq (dm-0)           252:0    0  476M  0 lvm   /boot
│       ├─HD-home (dm-1)            252:1    0  476M  0 lvm
│       └─┬─HD-home_crypt (dm-6)     252:6    0  474M  0 crypt /home
│           ├─HD-raiz (dm-2)         252:2    0   5,7G  0 lvm
│           └─┬─HD-raiz_crypt (dm-4) 252:4    0   5,7G  0 crypt /
│               └─HD-swap (dm-3)      252:3    0   1,4G  0 lvm
│                   └─HD-swap_crypt (dm-5) 252:5    0   1,4G  0 crypt [SWAP]
sdb                                  8:16     0   8G  0 disk
sr0                                  11:0     1 1024M  0 rom

```

```

marta@ISE:~$ ls /dev/ | grep sd
sda
sda1
sdb
marta@ISE:~$

```

Figura 3.3: Consultando cómo está el disco particionado y los discos que tenemos.

Con esa información, ejecutamos el comando `mdadm`:

```

Eliminando el disco defectuoso y añadiendo el nuevo
sudo mdadm /dev/md0 --add /dev/sdb --remove /dev/sda

```

Tras ejecutarlo, nos dice que se ha añadido con éxito el disco que hemos añadido nuevo, pero que no se ha encontrado el disco que íbamos a eliminar. Aún así, vemos que la ejecución ha sido exitosa en la Figura 3.4.

3.2. INSTALE *Nagios* EN SU SISTEMA (EL QUE PREFIERA) DOCUMENTANDO EL PROCESO Y MUESTRE EL RESULTADO DE LA MONITORIZACIÓN DE SU SISTEMA COMENTANDO QUÉ APARECE.

3.2.1. INSTALACIÓN

En [4] se documenta paso a paso cómo instalar *Nagios* en CentOS. Los pasos a seguir son los siguientes:

1. En primer lugar, nos vamos al directorio temporal para trabajar desde ahí:

```

Accediendo al directorio /tmp
cd /tmp

```

```

marta@ISE:~$ sudo mdadm /dev/md0 --add /dev/sdb --remove /dev/sda
[sudo] password for marta:
mdadm: added /dev/sdb
mdadm: hot remove failed for /dev/sda: No such device or address
marta@ISE:~$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda                                  8:0      0   8G  0 disk
├─sda1                              8:1      0   8G  0 part
│   └─md0                           9:0      0   8G  0 raid1
│       ├─HD-arrang (dm-0)          252:0    0  476M  0 lvm   /boot
│       ├─HD-home (dm-1)           252:1    0  476M  0 lvm
│       │   └─HD-home_crypt (dm-6)  252:6    0  474M  0 crypt /home
│       ├─HD-raiz (dm-2)           252:2    0   5,7G  0 lvm
│       │   └─HD-raiz_crypt (dm-4)  252:4    0   5,7G  0 crypt /
│       ├─HD-swap (dm-3)           252:3    0   1,4G  0 lvm
│       └─HD-swap_crypt (dm-5)      252:5    0   1,4G  0 crypt [SWAP]
sdb                                  8:16     0   8G  0 disk
├─md0                              9:0      0   8G  0 raid1
│   ├─HD-arrang (dm-0)            252:0    0  476M  0 lvm   /boot
│   ├─HD-home (dm-1)             252:1    0  476M  0 lvm
│   │   └─HD-home_crypt (dm-6)    252:6    0  474M  0 crypt /home
│   ├─HD-raiz (dm-2)             252:2    0   5,7G  0 lvm
│   │   └─HD-raiz_crypt (dm-4)    252:4    0   5,7G  0 crypt /
│   ├─HD-swap (dm-3)             252:3    0   1,4G  0 lvm
│   └─HD-swap_crypt (dm-5)        252:5    0   1,4G  0 crypt [SWAP]
sr0                                 11:0     1 1024M  0 rom
marta@ISE:~$

```

Figura 3.4: Consultando cómo ha quedado particionado el disco tras añadir el disco nuevo al RAID

2. Descargamos la última versión estable de *Nagios*:

```

Descargando la última versión estable de Nagios
wget http://assets.nagios.com/downloads/nagiosxi/xi-latest.tar.gz

```

3. Extraemos el archivo que hemos descargado:

```

Extrayendo el contenido del archivo descargado
tar xzf xi-latest.tar.gz

```

4. Una vez extraído el archivo vamos al directorio de nagios:

```

Accediendo al directorio de nagios
cd nagiosxi/

```

5. Tras esto ejecutamos el script de instalación:

```

Ejecutando el script de instalación
sudo ./fullinstall

```

A mitad de instalación nos pedirá la contraseña para MySQL.

6. Una vez instalado, para acceder a nagios basta con acceder a la dirección `localhost` o a la IP de nuestro servidor:

```

Accediendo a nagios desde Firefox
firefox http://10.0.2.10/

```

Una vez hecho eso, veremos la página de la [Figura 3.5](#).

7. Hacemos click en *Access Nagios* y nos saldrá una ventana como la de la [Figura 3.6](#) en la que debemos establecer algunos parámetros del sistema antes de empezar a usarlo.
8. Por último nos mostrará una ventana con nuestro nombre de usuario y contraseña para acceder al sistema. Haciendo click en *Login Nagios XI* e introduciendo el nombre de usuario y contraseña proporcionados, ya habremos terminado la instalación.
9. Tras acceder al sistema deberemos de aceptar la licencia y tras aceptar la licencia nos dará un pequeño tutorial inicial. La interfaz inicial de *Nagios* se ve en la [Figura 3.7](#).

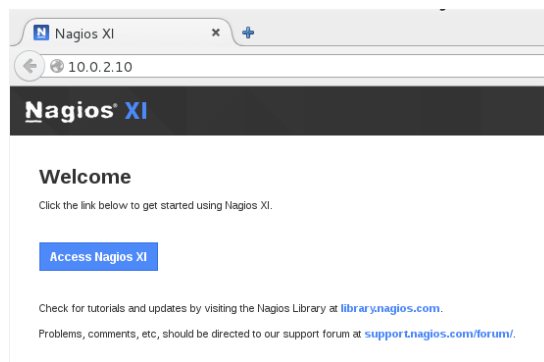


Figura 3.5: Página inicial de Nagios

Figura 3.6: Modificando algunos parámetros de Nagios antes de iniciar el sistema por primera vez

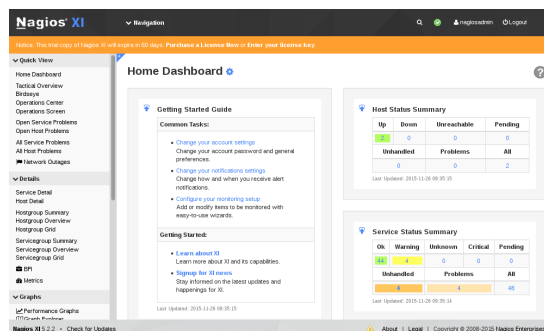
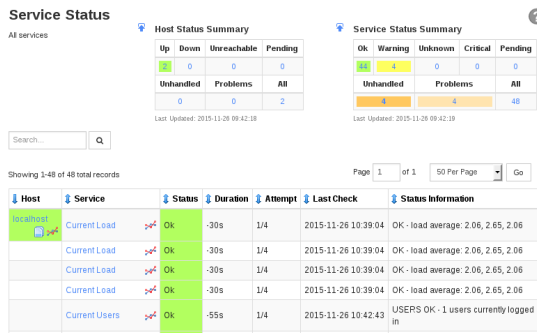


Figura 3.7: Interfaz de Nagios al acceder al sistema

3.2.2. MONITORIZACIÓN DEL SISTEMA

En el menú de la izquierda, tenemos algunas de las operaciones que podemos realizar con *Nagios*. Por ejemplo, en el submenú *Service Detail*, podemos comprobar el estado de cada uno de los servicios que tenemos a activos en el sistema, incluyendo el propio *Nagios* y el estado del propio sistema. En la **Figura 3.8** se ve el ejemplo realizado por nosotros, en el que obtenemos varias advertencias sobre el poco espacio en disco que nos queda.



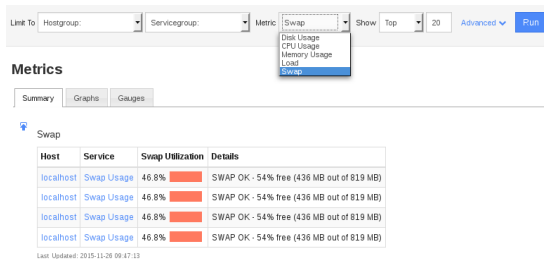
(a) Resumen del estado de los servicios del sistema

PING	OK	-45s	1/4	2015-11-26 10:38:33	PING OK - Packet loss = 0%, RTA = 0.07 ms
PING	OK	-45s	1/4	2015-11-26 10:38:33	PING OK - Packet loss = 0%, RTA = 0.07 ms
PING	OK	-45s	1/4	2015-11-26 10:38:33	PING OK - Packet loss = 0%, RTA = 0.07 ms
PING	OK	-45s	1/4	2015-11-26 10:38:33	PING OK - Packet loss = 0%, RTA = 0.07 ms
Root Partition	Warning	-10s	4/4	2015-11-26 10:41:39	DISK WARNING - free space / 897 MB (13% inode=95%)
Root Partition	Warning	-10s	4/4	2015-11-26 10:41:39	DISK WARNING - free space / 897 MB (13% inode=95%)
Root Partition	Warning	-10s	4/4	2015-11-26 10:41:39	DISK WARNING - free space / 897 MB (13% inode=95%)
Root Partition	Warning	-10s	4/4	2015-11-26 10:41:39	DISK WARNING - free space / 897 MB (13% inode=95%)
Service Status - crond	OK	0s	1/4	2015-11-26 10:39:48	crond.service - Command Scheduler
Service Status - crond	OK	0s	1/4	2015-11-26 10:39:48	crond.service - Command Scheduler
Service Status - crond	OK	0s	1/4	2015-11-26 10:39:48	crond.service - Command Scheduler
Service Status - crond	OK	0s	1/4	2015-11-26 10:39:48	crond.service - Command Scheduler

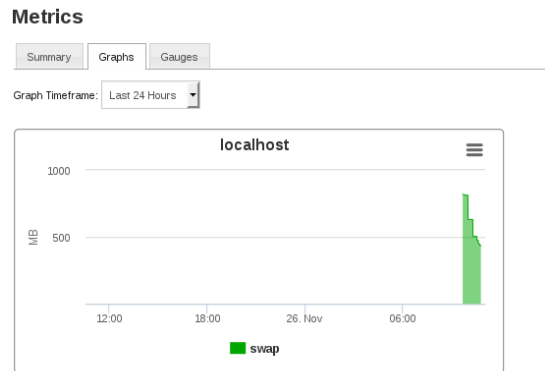
(b) Parte de la tabla detallada sobre cada servicio con varias advertencias

Figura 3.8: Monitorizando el estado de los servicios del sistema y del propio sistema

En el submenú *Metrics*, podemos obtener gráficas y datos sobre el uso de CPU, de memoria, de disco, de swap y la carga del sistema. En la **Figura 3.9** estamos monitorizando el uso de la memoria de intercambio en el sistema, que en este caso vemos que tiene un uso aceptable y que no está saturada ya que tenemos un 54% de memoria libre.

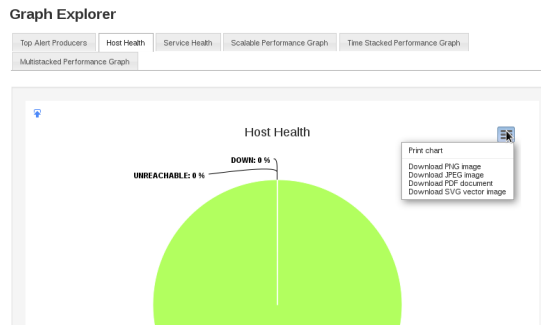


(a) Resumen del uso de swap en el sistema

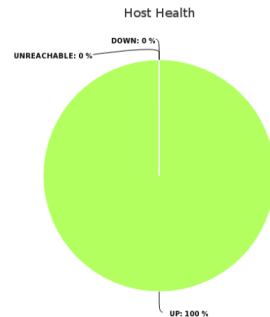


(b) Gráfica del uso de swap en el sistema

Figura 3.9: Monitorizando el uso de swap en el sistema



(a) Gráfico resumen de las veces que nuestro sistema no ha sido accesible



(b) Gráfico que hemos descargado en formato PNG sobre la "salud" de nuestro sistema

Figura 3.10: Monitorizando la "salud" del sistema

En el submenú *Graph Explorer*, podemos consultar algunos gráficos realizados por *Nagios* sobre nuestro servidor y nuestro sistema. En la **Figura 10(a)** se muestra un gráfico que muestra el número de veces que nuestro servidor ha estado innaccesible (o bien porque se ha saturado, o porque se ha perdido la conexión, etc). En este caso, nuestro servidor ha estado siempre accesible y por eso muestra toda la gráfica en color verde. Además, *Nagios* nos permite descargar el gráfico en varios formatos, por ejemplo en la **Figura 10(b)** se ve el archivo PNG de la gráfica descargado directamente desde *Nagios*.

3.3. ACCEDA A LA DEMO ONLINE DE *Ganglia* DE WIKIMEDIA Y HAGA LO MISMO QUE HIZO CON *Munin*

En **WikiMedia** nos ofrecen una demo online del profiler *Ganglia*, mostrándonos datos reales de sus servidores.

Nada más entrar a la página, nos encontramos con la página que se ve en la **Figura 3.11** vemos un resumen estadístico del grid que usa WikiMedia, con el número total de CPUs y servidores tanto activos como no activos. También nos ofrecen una estadística sobre la carga desde los últimos 15 minutos y la utilización media. A la derecha de éstos datos vemos cuatro gráficas con datos sobre la carga del sistema, el uso de memoria, de CPU y de red. El uso de memoria sí es algo alto (más de la mitad), pero tanto la carga del sistema como el uso de CPU es bastante bajo. El uso de red tiene picos que varían bastante, por ejemplo, desde las 20:40 hasta las 21:00 no ha tenido actividad ninguna, sin embargo, sí ha tenido un pico de actividad a las 9.

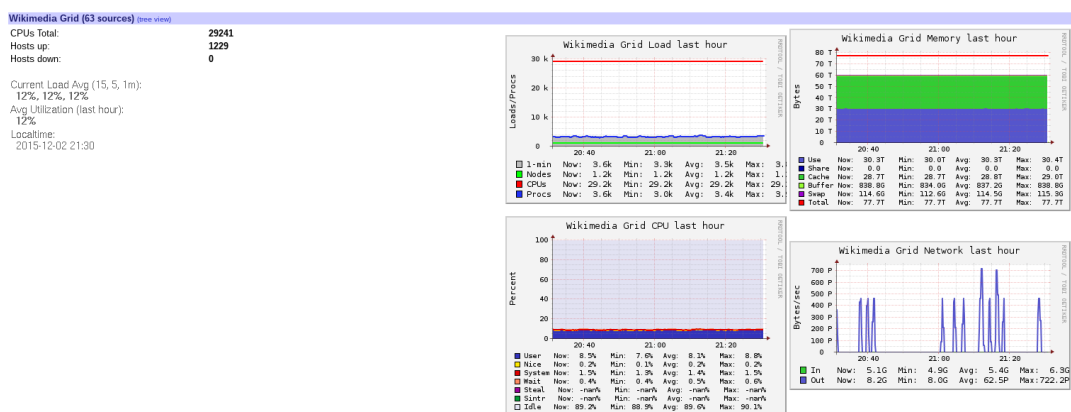


Figura 3.11: Información general del grid de WikiMedia

Después de esto, nos encontramos información sobre cada cluster de computadores de los que dispone WikiMedia (*codfw* y *equiad*). Si hacemos click en una de las gráficas, accedemos a una página en la que nos muestra información general sobre ese clúster, pero, dentro de dicha página podemos elegir ver información individual sobre un nodo concreto del clúster. Ésto se ve en la **Figura 3.12**. Si elegimos un *host*, vemos que tiene unas gráficas de rendimiento bastante parecidas a las del cluster (**Figura 3.13**).

3.9. ESCRIBA UN SCRIPT EN PYTHON Y ANALICE SU COMPORTAMIENTO USANDO EL PROFILER PRESENTADO

El script realizado consiste en cifrar usando el código de Polibio [12]. El script en cuestión es el siguiente:

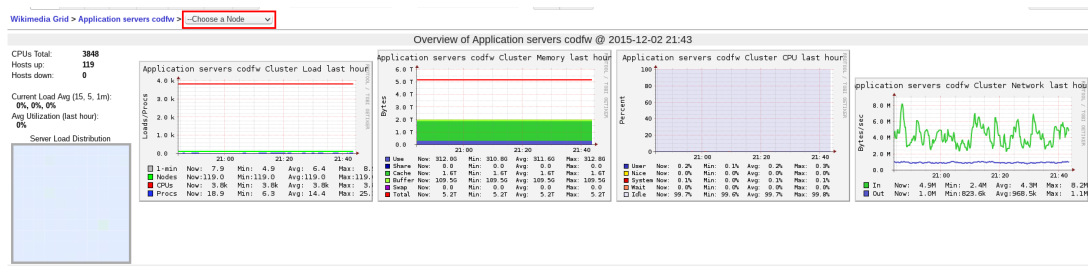


Figura 3.12: Información general sobre los servidores de aplicación del clúster codfw

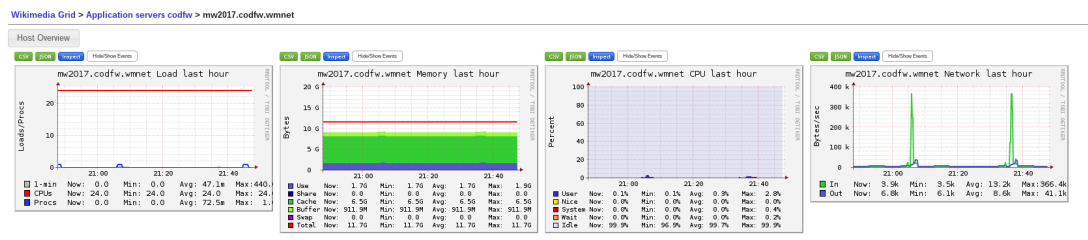


Figura 3.13: Información general sobre uno de los hosts del clúster codfw

```

polibio.py
1 tabla_polibio = [['a','b','c','d','e'], ['f','g','h','i','k'], ['l','m','n','o','p'],
2 ['q','r','s','t','u'], ['v','w','x','y','z']]
3
4 def cifra_polibio (frase):
5     cifrado = []
6     for caracter in frase:
7         for lista in tabla_polibio:
8             for letra in lista:
9                 if letra == caracter:
10                     cifrado.append((tabla_polibio.index(lista)+1)*10+(lista.index(letra)+1))
11
12     return cifrado
13
14 def descrifrado_polibio (cifrado):
15     frase = ""
16
17     for numero in cifrado:
18         frase += tabla_polibio[int(numero/10)-1][int(numero%10)-1]
19
20     return frase
21
22 if __name__ == '__main__':
23     frase_original = input("Introduce una frase: ")
24     frase = frase_original.replace("j", "i");
25
26     cifrado = cifra_polibio(frase)
27
28     print("Texto cifrado: " + ' '.join(str(n) for n in cifrado))
29
30     descrifrado = descrifrado_polibio(cifrado)
31

```

32 `print("Texto descifrado: "+descifrado)`

Para hacer el profiling he usado la librería *cProfile*, la cual, según [10], se usa con el siguiente comando:

Haciendo profile a un script en Python

```
python -m cProfile polibio.py
```

de forma análoga también podemos usar la librería *profile*, con la cual obtenemos también la misma salida.

Tras ejecutar el comando anterior, he obtenido la salida que se ve en la Figura 3.14. En primer lugar, el *profiler* nos muestra el número de llamadas a funciones que ha realizado nuestro programa y el tiempo total de ejecución. El tiempo es tan alto debido a que también cuenta el tiempo que el programa ha estado dormido esperando la entrada desde teclado.

Por último, nos muestra cada una de las funciones a las que ha llamado nuestro programa junto al número de veces que se ha llamado a cada función. Por ejemplo, las dos funciones que incluye el script (*descifrado_colibio* y *cifra_polibio*) se llaman una única vez cada una, como debe de ser pues en el script sólo se hace una llamada a cada función. También se reflejan llamadas a métodos primitivos de Python tales como *index*, *append*, etc los cuales se llaman más veces debido a que se llaman dentro de bucles *for*, en concreto, números relacionados con el tamaño de la frase a cifrar introducida.

```
[marta@marta-PC Practica3]$ python -m cProfile polibio.py
Introduce una frase: el poder desgasta al que no lo tiene
Texto cifrado: 15 31 35 34 14 15 42 14 15 43 22 11 43 44 11 11 31 41 45 15 33 34 31 34 44 24 15 33 15
Texto descifrado: el poder desgasta al que no lo tiene
127 function calls in 5.551 seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
1      0.000    0.000    5.551    5.551 polibio.py:1(<module>)
1      0.000    0.000    0.000    0.000 polibio.py:14(descifrado_polibio)
30     0.000    0.000    0.000    0.000 polibio.py:28(<genexpr>)
1      0.000    0.000    0.000    0.000 polibio.py:4(cifra_polibio)
1      0.000    0.000    5.551    5.551 {built-in method builtins.exec}
1      5.550    5.550    5.550    5.550 {built-in method builtins.input}
2      0.000    0.000    0.000    0.000 {built-in method builtins.print}
29     0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
1      0.000    0.000    0.000    0.000 {method 'disable' of 'lsprof.Profiler' objects}
58     0.000    0.000    0.000    0.000 {method 'index' of 'list' objects}
1      0.000    0.000    0.000    0.000 {method 'join' of 'str' objects}
1      0.000    0.000    0.000    0.000 {method 'replace' of 'str' objects}
```

Figura 3.14: Salida obtenida tras ejecutar *cProfile* sobre un script Python

4. PRÁCTICA 4

4.1. SELECCIONE, INSTALE Y EJECUTE UN BENCHMARK, COMENTE LOS RESULTADOS. ATENCIÓN: NO ES LO MISMO UN BENCHMARK QUE UNA SUITE, INSTALE UN BENCHMARK

En nuestro caso, de toda la lista de benchmarks disponibles, hemos elegido uno para medir la capacidad de la CPU para comprimir video usando la librería *x264* ([7]). Para instalarlo, según [11], usamos el siguiente comando:

Instalando el benchmark *x264*

```
sudo phoronix-test-suite benchmark x264
```


tras esto, empezará a descargar e instalar el benchmark y sus dependencias. Una vez instalado nos mostrará el software y hardware de nuestra máquina y nos preguntará si queremos guardar la información en un fichero y si le decimos que no, empezará a hacer pruebas. En la prueba que hemos hecho en la [Figura 4.1](#), el benchmark nos da como conclusión que podríamos codificar unos 26 *frames* por segundo.

```

Would you like to save these test results (Y/n): n

x264 2015-11-02:
pts/x264-2.0.0
Test 1 of 1
Estimated Trial Run Count: 5
  Started Run 1 @ 15:22:00
  Started Run 2 @ 15:22:28
  Started Run 3 @ 15:22:53
  Started Run 4 @ 15:23:18
  Started Run 5 @ 15:23:43 [Std. Dev: 4.74%]
  Started Run 6 @ 15:24:08 [Std. Dev: 4.60%]
  Started Run 7 @ 15:24:33 [Std. Dev: 4.27%]
  Started Run 8 @ 15:24:57 [Std. Dev: 3.98%]
  Started Run 9 @ 15:25:22 [Std. Dev: 3.73%]
  Started Run 10 @ 15:25:48 [Std. Dev: 3.52%]

Test Results:
  23.37
  26.14
  26.23
  25.98
  25.89
  26.69
  26.31
  26.18
  25.98
  25.77

Average: 25.85 Frames Per Second

```

Figura 4.1: Salida del benchmark x264 mientras realiza sus pruebas

4.3. LEA EL ARTÍCULO DE COMPARACIÓN ENTRE *Jmeter* Y *Gatling* ELABORADO POR LA EMPRESA *Flood.io* Y ELABORE UN BREVE RESUMEN

En *Flood.io* no se fían de los benchmark “competitivos” ya que están hechos para favorecer a las empresas que lo patrocinan y no dan resultados realmente fiables. Por eso, se pasaron al lado del código abierto y probaron *Jmeter* y *Gatling*.

Tras hacer el *test plan* con los dos benchmark, en el cual había 10.000 usuarios y 30.000 peticiones por minuto, el resultado obtenido es que ambos benchmark son bastante parecidos pero tienen una diferencia: *Gatling* no es capaz de guardar el tamaño en bytes de la respuesta, pero sin embargo, a pesar de que *Jmeter* sí lo es consume más recursos de CPU y memoria.

El artículo concluye diciendo que ambos benchmark son muy parecidos en términos de concurrencia y *throughput* y que la elección de uno u otro es puramente subjetiva y hecha sobre alguna otra característica que cada herramienta incluya.

REFERENCIAS

- [1] J. Z. BROCKMEIER, *Taking command of the terminal with gnu screen*. Disponible en <https://www.linux.com/learn/tutorials/285795-taking-command-of-the-terminal-with-gnu-screen->. Consultado el 31/10/2015.

- [2] D. EDWARDS, *Emacs keybindings in bash or, how to be a command-line commando*. Disponible en <http://www.pythian.com/blog/emacs-keybindings-in-bash/>. Consultado el 17/10/2015.
- [3] FAIL2BAN, *Manual 0.8*. Disponible en http://www.fail2ban.org/wiki/index.php/MANUAL_0_8. Consultado el 30/10/2015.
- [4] NAGIOS, *Nagios xi – manual installation instructions*. Disponible en https://assets.nagios.com/downloads/nagiosxi/docs/XI_Manual_Installation_Instructions.pdf. Consultado el 25/11/2015.
- [5] OPENSUSE, *Gestión de paquetes*. Disponible en https://es.opensuse.org/Gesti%C3%B3n_de_paquetes. Consultado el 28/10/2015.
- [6] —, *zypper*. Disponible en <https://es.opensuse.org/Zypper>. Consultado el 28/10/2015.
- [7] V. ORGANIZATION, *x264, the best h.264/avc encoder*. Disponible en <https://www.videolan.org/developers/x264.html>. Consultado el 5/12/2015.
- [8] L. M. PAGES, *awk: pattern scanning/processing*. Disponible en <http://linux.die.net/man/1/awk>. Consultado el 8/11/2015.
- [9] —, *mdadm: manage md devices aka software raid*. Disponible en <http://linux.die.net/man/8/mdadm>. Consultado el 15/10/2015.
- [10] PYTHON, *The python profilers*. Disponible en <https://docs.python.org/2/library/profile.html>. Consultado el 29/11/2015.
- [11] U. WIKI, *Phoronix test suite howto*. Disponible en <https://wiki.ubuntu.com/PhoronixTestSuite>. Consultado el 2/12/2015.
- [12] WIKIPEDIA, *Cuadrado de polibio*. Disponible en https://es.wikipedia.org/wiki/Cuadrado_de_Polibio. Consultado el 29/11/2015.