Laboratorio de Administración y Gestión de Redes y Sistemas
Escuela de Ingeniería de Fuenlabrada
Universidad Rey Juan Carlos
Septiembre de 2022

Índice

1. Introducción a Linux

1.1. Un poco de historia de Unix y Linux

Un poco de historia de Unix y Linux

- UNIX surgió en 1969 en los Laboratorios Bell (Ken Thomson, Dennis Ritchie)
- Dos grandes vertientes
  - BSD: SunOS, NetBSD, OpenBSD, Mac OS
  - System V: Solaris, Iris, Aix, Linux (año 1991)
    Distribuciones Linux
    - Slackware
    - Gentoo
    - Suse
    - RedHat y derivados: Fedora, Mandriva (Mandrake)
    - Debian y derivados: Ubuntu, knoppix, GnuLiNex, guadalinex

Partes de un sistema operativo

- Kernel (Núcleo): elemento más importante. Permite que las aplicaciones accedan al hardware. Es responsable de la gestión de recursos, seguridad, etc
- Procesos de usuario: distintos programas ejecutándose concurrentemente en un sistema
- La interacción entre el núcleo y los procesos se hace mediante llamadas al sistema (*system calls*)

La shell es un interfaz de usuario en modo texto. Es una aplicación como otra cualquiera

1.2. ¿GNU/Linux ó Linux?

¿GNU/Linux ó Linux?

1. La Free Software Foundation (Richard Stallman) considera que:
   - Linux es estrictamente el kernel

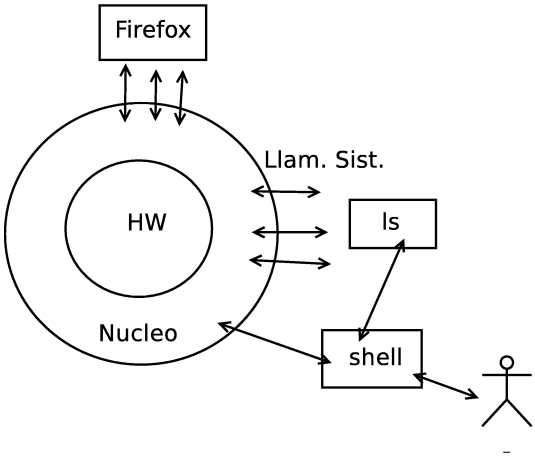


Figura 1: El Sistema Operativo

- Los procesos de usuario (*programas y otras utilidades básicas para el sistema*) provienen del proyecto GNU (y algunos otros).
- Al conjunto se le debe llamar GNU/Linux.

2. Un número importante de personas y organismos se oponen a esta definición. La mayoría de la gente lo llama simplemente Linux

2. Linux es Software Libre

Linux es Software Libre

Linux es el *producto estrella* del Soft. Libre

- Hay software libre para cualquier S.O.
- Hay software propietario para Linux

Cuatro libertades. Quien lo recibe tiene:

- libertad de uso. Usarlo como quiera, donde quiera
- libertad de redistribución. Redistribuirlo a quien quiera, como quiera
- libertad de modificación. Modificar, adaptar, corregir, mejorar
- libertad de distribuir las modificaciones

Imprescindible: disponibilidad de código fuente.

Como cualquier modelo, puede ser criticado. Desarrollar software libre no es ni garantía de calidad ni garantía de éxito. Pero algunos argumentos en contra habituales no tienen ningún sentido:

- *Los médicos, los abogados y los fontaneros no trabajan gratis. ¿Por qué habrían de hacerlo los programadores?*

software libre $\neq$ software gratis
desarrollar software libre $\neq$ no cobrar

¿Qué no es software libre?

- Software gratuito
- Shareware
- Adware
- Versiones de evaluación
- Dominio Público

Tipos de licencias libres:

- Minimalistas. Permiten *cerrar* el código. Pj BSD
- *Protectoras de la libertad.* GPL.
  Redistribuciones con mismos derechos que la primera distribución

Motivos para desarrollar software libre

- Ética, satisfacción personal, pertenencia a una comunidad
- Aprendizaje
- Tesis doctorales, TFG
- Empresas que se dedican a otra cosa
- Organismos públicos
- Empresas que obtienen dinero por servicios
- Empresas de Hardware
- etc etc

2.1. Inconvenientes del software libre para el usuario

Inconvenientes del software libre para el usuario

- Ninguno. El modelo de software libre no tiene, por sí mismo, ningún inconveniente para el usuario

Si bien habrá ocasiones, cada vez menos, en que una solución de software propietario resulta más apropiada:

- Software libre inexistente o de calidad insuficiente
- Hardware no soportado
- Soporte insuficiente
- Otros (discutible) *Quien me rodea usa determinado software*

Es muy habitual usar soluciones propietarias *por inercia*, un pequeño esfuerzo en explorar las alternativas libres puede ser extraordinariamente rentable

2.2. Ventajas del software libre para el usuario

Ventajas del software libre para el usuario

- 4 libertades
- Facilita la reutilización
- Mucho menor coste
- Nadie impone la renovación de Hw, Sw ni formación de usuarios
- Mejor interoperabilidad y escalabilidad
- Garantía de privacidad
- Permite conocer mejor el software y comprobar su calidad
- Igualdad de oportunidades: Mismas herramientas para todos. Promoción de economía local

Más información: Estudio FLOSSImpact

3. El Sistema Operativo

3.1. Procesos

El Sistema Operativo se ocupa de:

- Gestión de procesos
- Gestión de memoria
- Gestión de dispositivos
- Gestión de sistemas de ficheros
- Gestión de red
- Procesos = ejecutables + librerías dinámicas
- Identificadores asociados a cada proceso:
  - PID: Identificación única de cada proceso
  - UID: Identificación de usuario
  - GID: Identificación de grupo (posibilidad de varios grupos por proceso)
- uid=0 ⇒ *super-usuario*, "`root`":
  - Control sobre el resto de procesos
  - Permiso para acceder a todos los ficheros
  - Posibilidad de realizar ciertas tareas privilegiadas

3.2. Servicios principales de un sistema UNIX

- `init`. Primer proceso, padre de todos los demás. Se encarga de arrancar y parar el sistema.
- Terminales remotas: *login* y *logout*
- `syslog`
- Ejecución periódica de órdenes: `cron` y `at`
- Entorno gráfico (X Window)
- Entorno de red (demonios)
- Correo electrónico, sistema de impresión, ...

3.3. Interfaces de usuario

Interfaz gráfico

- En los años 1980 y 1990, la aparición de los interfaces gráficos supuso un gran avance, al facilitar el uso de los ordenadores, especialmente para usuarios no especializados. También para tareas que se hacen eventualmente
- Para tareas exigentes, son mucho menos eficiente que los interfaces de texto: obligan a hacer las cosas *a mano* y de una en una
- Solo se puede hacer lo que el interfaz haya previsto que se haga
- No es la filosofía Unix, no son estándar
- Exigen sesión gráfica (mucho más caro que pj ssh)
- No siempre disponibles (sistemas empotrados, routers, etc)
- Hay gestores gráficos, pero no serán válidos en esta asignatura

Unix dispone de interfaz gráfico desde los 80: *X Window*, año 1984 (No confundir con Microsoft Windows).

- X Window System es un sistema gráfico utilizado fundamentalmente en sistemas Unix, aunque es multiplataforma. También llamado X11, está desarrollado por la fundación X.org
- Proporciona un mecanismo para mostrar ventanas gráficas basado en dos partes: cliente y servidor
  - Servidor X: Se ejectuta típicamente en la máquina en la que está sentado el usuario.
  - Clientes X: Aplicaciones que producen una salida gráfica que envían al Servidor X para que la presente en pantalla. Pueden ejecutarse en ordenadores remotos.

Es un sistema complejo, que requiere componentes adicionales:

- Sobre las X Window va el *gestor de ventanas* (Kwin, Enlightenment, Metacity, Xfwm, MWM...)
- Sobre el gestor de ventanas, va el *escritorio*

  (KDE, Gnome, Xfce...)

Desde finales de la década de 2010, se empieza a popularizar *wayland* como alternativa a X Window

- Es más moderno, ligero y seguro
- También sigue la arquitectura cliente-servidor
- No usa gestores de ventanas sino unos elementos similares llamados *compositors*
- Los escritorios (y las aplicaciones) son los mismos que con X Window, el usuario final no percibe la diferencia

Interfaz de texto: consola

El interfaz de texto resulta más dificil de aprender a manejar, pero resulta mucho más flexible y potente

Write programs that do one thing and do it well. Write programs to work together. Write programs that handle text streams, because that is a universal.

- interfaz texto: teclado
  - terminales x
  - consola: terminales virtuales (Ctrl+Alt+F1) (Ctrl+Alt+F6)
  - Vuelta a sesión X (Ctrl+Alt+F7)
- `exit` (EOF, Ctrl + D)

En MS Windows el interfaz de consola para la administración es una opción viable desde la aparición en 2006 de PowerShell

4. El administrador

El administrador

Persona responsable de:

- Instalación del sistema
- Incorporar nuevo *hardware* y *software*
- Añadir y eliminar usuarios
- Salvaguarda de información
- Seguimiento y monitorización del sistema
- Política de seguridad
- Resolución de problemas
- Soporte técnico

Hay tareas que solo el usuario root puede hacer. Entre otras:

- Editar ficheros de configuración de la máquina, como los interfaces de red, sistemas de ficheros, arranque del sistema...
- Iniciar y detener demonios (en los casos habituales, aunque excepcionalmente un usuario ordinario podría gestionar algún demonio)
- Instalar paquetes de software (para toda la máquina)
- Crear, modificar, eliminar cuentas de usuario

4.1. Procesos de root, método tradicional

Procesos de root, método tradicional

- Por motivos de seguridad, el administrador debería lanzar procesos como root solo para lo que sea imprescindible, el tiempo imprescindible. El resto del tiempo, es preferible trabajar con una cuenta de usuario ordinario

Formas tradicionales de lanzar procesos como root

1. Iniciar sesión con usuario root y contraseña de root (no recomendable, muchas veces no permitido)

2. `su`

   Solicita la contraseña de root y ejecuta una shell como root. Todos los procesos lanzados serán de root, hasta que se cierre la shell

4.2. Procesos de root con sudo

Procesos de root con sudo

Para mejorar la seguridad, aparece `sudo`, que evita que haya una sesión de root siempre abierta

- `sudo <orden>`

  Permite ejecutar una única órden como root, desde una sesión de usuario ordinario

- `sudo` no pregunta la contraseña de root, no es necesario conocerla (de hecho, en los sistemas con sudo, el usuario root no suele tener contraseña)

- `sudo` solicita la contraseña del propio usuario que está lanzando sudo

  - Para llamarle la atención y hacerle tener más cuidado
  - Para verificar que la persona que está en el terminal sigue siendo quien ha abierto la sesión

Solamente algunos usuarios pueden ejecutar sudo:

- Aquellos indicados en el fichero `/etc/sudoers`

- Por omisión, el usuario que instala la máquina está en `/etc/sudoers`

- En ubuntu, se incluye en `/etc/sudoers` a todos los usuarios del grupo *admin*

Por todo esto, `sudo` es más seguro que `su`, aunque

- en algunos sistemas, `sudo` no está instalado y no se puede usar

- en algunos sistemas, el uso de `sudo` es opcional

- en algunos sistemas, como Ubuntu, el uso de `sudo` es obligatorio, los usuarios no pueden ejecutar `su`

- Pero si vamos a ejecutar muchas órdenes como root y sudo nos resulta incómodo, podemos hacer una pequeña *trampa*:
  `sudo su`

Para aplicaciones gráficas, no debe usarse `sudo` sino `gksudo`

- `gksudo` *mi-aplicacion*

sudo y redirecciones

La orden *sudo* por omisión no incluye las posibles redirecciones

- `sudo echo hola > /tmp/aa`

  El proceso *echo* se lanza con la identidad del root (id 0), pero la redirección la ejecuta el usuario ordinario

- Para poder usar redirecciones, ejecutamos una subshell con el parámetro -c

  ```
  sudo bash -c  "echo hola>aa"
  sudo bash -c  "find /root | grep prueba "
  ```

  (o, alternativamente, `sudo su`)

4.3. Algunas ambigüedades

Algunas ambigüedades

1. En el lenguaje oral, la palabra *root* a veces provoca ambigüedades, podemos referirnos a

   - El usuario *root*
   - El directorio *home* del usuario *root*:
     `/root`
   - El punto del que cuelga el sistema de ficheros:
     `/.`

   Por el contexto se distingue fácilmente

2. La palabra *fichero* también puede tener distintos significados

   - El sentido habitual en informática (fichero ordinario)
   - En sentido Unix, incluye ficheros ordinarios, directorios, pipes y ficheros especiales (dispositivos)

5. Usuarios y grupos

5.1. Identificadores de usuario y grupo

Usuarios y grupos

- Cada usuario tiene un identificador (*UID*), un grupo principal (o primario) al que pertenece (*GID*), una serie de grupos adicionales, un nombre de usuario (*login*), un directorio de trabajo (*home*)
- La orden `id` nos da el UID, el GID y los grupos adicionales de un usuario
- Cada usuario puede tener dos tipos de recursos en un sistema UNIX: Procesos y Ficheros
- Cada UID y cada GID puede tener asociado un nombre, especificado en los ficheros `/etc/passwd` y `/etc/group`, respectivamente
- La información de `/etc/passwd` y `/etc/group` la utilizan diversas órdenes de administración. Ambos ficheros deben existir y ser coherentes para que el sistema funcione correctamente
- No es recomendable editar estos ficheros directamente, sino mediante mandatos como `usermod`
- Si se edita `/etc/passwd` y `/etc/group/` directamente, debe usarse `vipw` y `vigr`

5.2. Importancia de las contraseñas

Importancia de las contraseñas

- El campo *passwd* en el `/etc/passwd` y el `/etc/shadow` se encuentra cifrado con una función *hash* para evitar que los usuarios (y administradores) puedan conocer las contraseñas de otros usuarios.
- Se usa un cifrado de un solo sentido: no existe algoritmo para averiguar la contraseña a partir de estos ficheros.
- Pero se pueden probar varias contraseñas, hasta millones por segundo (John the Ripper).

- Es imprescindible elegir palabras clave seguras, que no aparezcan en diccionarios, evitando nombres o fechas significativas, combinando símbolos, y de la mayor longitud posible.

- Ejemplos de malas contraseñas:

```
123456
4312
toby
r2d2
tornillo
fromage
mostoles
```

- Contraseñas que parecen buenas, pero son malas:

```
XCV330
NCC-1701-A
ARP2600V
```

- Buenas contraseñas

  Para usos ordinarios, una contraseña razonablemente buena será parecida (¡pero no igual!) a una de estas

```
Contraseña  Nemotécnico
----------------------------------------------------
00QuReMa:   Queridos Reyes Magos:
3x4igDoze   3x4=doce
1pt,yTp1    uno para todos,todos para uno
19Dy500n    19 dias y 500 noches
R,cmqht?0   Rascayú, cuando mueras que harás tú?
wali1YS!    we all live in a Yellow Submarine
Lh10knpr.   le hare una oferta que no podrá rechazar
```

  Para aplicaciones especialmente sensibles, es necesario ampliar el *keyspace* a 14 caracteres o más

- Es conveniente que busquemos e inutilicemos las contraseñas débiles de nuestros usuarios, ya que suponen un primer punto de entrada en nuestro sistema

- En Unix, el root no puede leer las contraseñas. Pero en otros entornos sí. Y muchos usuarios emplean siempre la misma contraseña

  Ejemplo:

  1. Juan Torpe usa como contraseña dgj441iU en juan.torpe@hotmail.com
  2. Juan Torpe se apunta en `www.ladygaga-videos-prohibidos.com`, con su cuenta de correo y su contraseña de siempre
  3. El administrador malicioso de este web ya conoce el nombre de Juan, su cuenta de correo y su contraseña.
     - Puede usar la función *¿contraseña olvidada?* y colarse en cualquier otra cuentas de Juan

  Debemos instruir a nuestros usuarios sobre esto

- Los usuarios sin duda olvidarán en ocasiones su contraseña y tendremos que generles una nueva, de forma segura

- Pero es muy poco profesional que nosotros como administradores olvidemos una contraseña. Debemos usar varias y guardarlas de forma medianamente segura (gpg, keepassx, lastpass, etc)

5.3. Secret sharing

Secret sharing

Aunque pongamos cuidado extremo en guardar nuestra contraseña, esto puede no ser suficiente.

- ¿Y si a pesar de todo, la perdemos o nos la roban?

- ¿Y si no estamos disponibles? (Viaje, enfermedad, muerte...)

- ¿Y si nos secuestran?

- ¿Y si hacemos algún disparate?

Podríamos dividir la contraseña en n trozos y repartirlos entre n personas de nuestra confianza, de forma que con el acuerdo de todos, el secreto es recuperable

- Problema: Bastaría con que se pierde un trozo para perder el secreto

- Solución: Algoritmos de *secret sharing*

Un esquema *secret sharing* es aquel que permite dividir un secreto en n fragmentos, de forma que basten t ($t < n$) para reconstruirlo

Armory incluye esto de manera nativa, con electrum podemos usar una herramienta independiente, ssss

Con la orden `ssss-split`, dividimos el secreto en n fragmentos

```
koji@mazinger:~$ ssss-split -t 3 -n 5
WARNING: couldn't get memory lock (ENOMEM, try to adjust RLIMIT_MEMLOCK!).
Generating shares using a (3,5) scheme with dynamic security level.
Enter the secret, at most 128 ASCII characters
ABRETE SESAMO
1-378a29cbbe9b38f0d473bdab0654
2-d7cc58bbce72070afc675f0991dd
3-d42a4e6f0dca1d32a6d1f96e4e92
4-629dd862cb052f8774bdb26d91df
5-617bceb608bd35bf2e0b140a4e82
```

Con la orden `ssss-combine`, recuperamos el secreto

```
koji@mazinger:~$ ssss-combine -t 3
WARNING: couldn't get memory lock (ENOMEM, try to adjust RLIMIT_MEMLOCK!).
Enter 3 shares separated by newlines:
Share [1/3]: 2-d7cc58bbce72070afc675f0991dd
Share [2/3]: 3-d42a4e6f0dca1d32a6d1f96e4e92
Share [3/3]: 5-617bceb608bd35bf2e0b140a4e82
Resulting secret: ABRETE SESAMO
```

Si el secreto es mayor de 128 bytes, se cifra con una clave tradicional y se aplica ssss a esta nueva clave

`/etc/passwd`

- Contiene la información de todos los usuarios del sistema.
- Contenido: líneas con campos separados por dos puntos:
  login : *passwd* : *UID* : *GID* : *info* : *home-dir* : *shell*
- El campo "*login*" puede tener hasta 32 caracteres en Linux, pero se recomienda limitarlo a 8, como en los UNIX clásicos

- El campo "*passwd*" contiene la contraseña cifrada (con DES o con MD5) y puede estar en otro fichero, en el `/etc/shadow`.
- El campo "*info*" contiene el nombre real del usuario e información adicional como el teléfono, etc. Por (desafortunados) motivos históricos, también se le denomina GECOS
- En algunos sistemas, puede haber información externa (NIS, LDAP...)
- Programas que lo utilizan directamente: `login`, `su`, `passwd`.

`/etc/group`

- Nombres de grupos del sistema, y miembros de cada grupo.
- Contenido: líneas con campos separados por dos puntos:
  nombre : *passwd* : *GID* : *lista-logins*
- "*lista-logins*" son usuarios separador por comas que pertenecen a ese grupo.
- El campo "*passwd*" no se suele utilizar. Permite ingresar en un grupo en el que no se es miembro.
- En algunos sistemas, puede haber información externa (NIS, LDAP...)

/etc/shadow

- Si existe, contiene las contraseñas cifradas de los usuarios del sistema.
- Contenido: líneas con campos separados por dos puntos:
  login : *passwd* : *a* : *b* : *c* : *d* : *e* : *f* : *g*

  a: momento en que la *passwd* fue cambiada por última vez.

  b: días que deben pasar antes de que pueda cambiarse.

  c: días después de los cuales la *passwd* debe cambiarse.

  d: días antes de la expiración para avisar al usuario.

  e: días después de la expiración para desactivar la cuenta.

  f: momento en que la cuenta se ha desactivado.

  g: campo reservado.

Para mejorar la seguridad se añade un "salt"

salt es un tipo de *nounce*: Number used once. 2 bytes aleatorios que se añaden a la contraseña

Sin salt

```
password  -->  hash (password)
"sesamo" --> zv/coRb$PjGToGEqNZF434TmQ7bAH.rVi3i.o7IWQAI9qqzeGKe/JkJq
bDfQE2gBFYzBTDNCHyoxpZvSLhenkPT3L6aZN0
```

El atacante puede usar una *rainbow table*: El resultado de aplicar hash a un diccionario completo. Si encuentra la hash en la tabla, conoce la contraseña que fue usada

```
rainbow table
hash(palabra1)
hash(palabra2)
hash(palabra3)
```

Con salt

```
password+salt  -->  hash (password+salt)

rainbow table:
hash(palabra1+salt1)
hash(palabra1+salt2)
hash(palabra1+salt3)
```

- *salt* se guarda en abierto: se añade al hash, son los primeros dos bytes
- Esto obliga a que la rainbow table sea mucho mayor, puede hacerla inviable

Desactivar un usuario del sistema

- Bloquear su contraseña en el `/etc/passwd` o `/etc/shadow` (añadiendo un carácter "`-`" o "`*`", por ejemplo).
- Eliminar sus tareas periódicas (`/var/spool/cron`).
- Revisar `/etc/aliases` y `.forward` por si el usuario tuviera acciones a realizar con el correo recibido.

Eliminar un usuario

- `userdel`
- `userdel -r` también borra su correo y su *home*

Usuarios especiales

- No todas las líneas del `/etc/passwd` corresponden con usuarios físicos.
- Super-usuario: uid=0 (su *login* es normalemente `root`).
- Otros usuarios del sistema: se utilizan para:
  - tareas específicas de administración
  - propietarios de determinados ficheros del sistema
  - ejecución de determinadas aplicaciones (bases de datos, servidores de web, ftp, e-mail, noticias, etc)
- Normalmente, los usuarios normales tienen UIDs entre el 1000 y el 30000.

Cambio de contraseña

Para cambiar la contraseña y otros datos se utilizan las órdenes `passwd` (contraseña), `chfn` (info/gecos), `chsh` (*shell*):

- Estas órdenes tienen *set-uid* para que un usuario normal pueda modificar información privilegiada.
- Antes de nada, piden la *passwd* del usuario para verificar que es quien dice ser.
- Bloquean cada fichero a modificar para asegurar exclusión de accesos.
- Realizan las modificaciones.
- Desbloquean ficheros.

Para crear o cambiar la contraseña de un usuario desde un script

- `echo "jperez:sesamo" | chpasswd`

Cambios de usuario y grupo

- `su` ejecuta otra *shell* bajo un usuario distinto.
  - `su jperez`
    ejecuta otra shell, perteneciente al usuario *jperez*
    No hace falta ser root para ejecutar esto, si la invoca un usuario ordinario, le pedirá la contraseña de *jperez*

- `su`
  ejecuta shell con uid=0 (`root`).
- Pide la contraseña del usuario destino, excepto salvo si el origen es `root`.

- `newgrp` ejecuta una *shell* con distinto GID.
  - Tiene *set-uid*
  - `newgrp` permite cambiar el GID a otro grupo al que pertenezcamos (cambia el grupo primario)

Mandatos que sólo pueden ejecutarse como root

- `groupadd` *grupo*
  crea un grupo
- `adduser` *usuario*[1]
  añade un usuario. Copia en su home el directorio `/etc/skel`
  `adduser` *usuario grupo*
  añade un usuario a un grupo
- `usermod -g` *grupo_primario usuario*
  Cambia el grupo primario por omisión del usuario
- `passwd` *usuario*
  Cambia la contraseña de un usuario
- `chown` *dueño fichero(s)*
  cambia el dueño de un fichero
- `chgrp` *dueño fichero(s)*
  cambia el grupo de un fichero

Mandatos para cualquier usuario

- `passwd`
  Cambia la contraseña del usuario
- `newgrp` *grupo*
  Entre los grupos de un usuario, elige el grupo primario

[1] En RedHat, useradd usuario y chfn usuario

6. Máquinas Virtuales

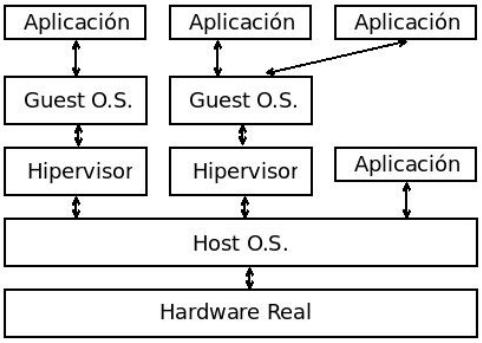
Máquinas Virtuales

Máquina Virtual: Software que crea una capa de abstracción, ofreciendo una máquina diferente a la máquina original

Las máquinas virtuales que nos interesan en administración de sistemas suelen ofrecer a un sistema operativo la percepción de una máquina física

- Las aplicaciones y los usuarios dentro de la máquina virtual se relacionan con la capa de abstracción y no con la plataforma real
- La máquina virtual puede implementar diversos dispositivos virtuales (disco, dispositivos de red, etc) diferentes a los de la plataforma real
- La tecnología sobre Máquinas Virtuales está muy madura. La terminología, no. Es frecuente encontrarse con el distintos nombres para el mismo concepto, o incluso el mismo nombre para cosas distintas
- *Guest*: Sistema Operativo de la máquina virtual

  Host: Sistema Operativo de la máquina real



Uno de los modelos posibles: máquina virtual de sistema

- La máquina virtual se comporta como una aplicación más en el *host*
- El *guest* percibe la máquina virtual como si fuera hardware real

6.1. Utilidad de las máquinas virtuales

Utilidad de las máquinas virtuales

Tecnología tradicional y actual, con muchas utilidades

- Ejecutar aplicaciones hechas para una plataforma sobre una plataforma diferente: p.e Microsoft Windows sobre Mac OS, Java Virtual Machine
- Ofrecer un entorno seguro donde experimentar (*sandbox*)

    - Docencia
    - Probar aplicaciones en desarrollo
    - Probar aplicaciones o webs no confiables
- Señuelos (*Honeypots*)
- Empresas de *hosting* pueden ofrecer servidores virtuales (alimentación y conectividad redundante, soporte 24/365, etc)
- Respaldo (*backup*) de máquinas enteras, no solo de datos. Ante un pequeño problema o un gran desastre, la máquina virtual se recupera inmediatamente
- Seguridad: Cortafuegos, perímetros de seguridad,...
- Portabilidad: Moviendo un directorio se puede mover la máquina virtual de una máquina real a otra
- Independencia del Hardware, p.e. homogeneizar un conjunto de máquinas diferentes
- ...

6.2. Inconvenientes de las máquinas virtuales

Inconvenientes de las máquinas virtuales

Inconveniente principal: pérdida de rendimiento
Aunque no siempre

- La máquina *real* tal vez no existe (p.e. java)
- Existe, pero es una máquina de propósito específico.

  Un guest sobre un host de propósito general puede ser más eficiente

6.3. Tipos de Máquina Virtual

6.3.1. MV de proceso y de sistema

MV de proceso y de sistema

Según su grado de equivalencia sobre una máquina hardware, las máquinas virtuales se pueden clasificar en

- Máquinas virtuales de sistema

  Proporcionan un entorno completo y persistente para ejecutar un sistema operativo y sus procesos

  Ej: VirtualBox, Xen

- Máquinas virtuales de proceso

  Proporcionan una plataforma para ejecutar un único proceso

  - Contenedores
  - Máquina virtual de java, .NET

    Este tipo no lo consideramos dentro del ámbito de la administración de sistemas y no lo tratamos aquí

6.3.2. Emulación Completa o Virtualización Completa

Emulación Completa o Virtualización Completa

Whole-system virtualization. Se emula memoria, disco y otros dispositivos, también la CPU:

Al emular la CPU, son especialmente lentos. La arquitectura Intel tradicional ofrecía muy pocas facilidades

Permiten que *guest* y *host* trabajen con diferente ISA (*instruction set architecture*)

Ejemplos: QEMU, Bochs.

- Emulan una CPU intel, incluso cuando se ejecutan sobre intel.
- Ambos son libres, disponibles para diversos *hosts*.
- Pueden ejecutar distintos *guest*, pero siempre para intel

6.3.3. Virtualización

Virtualización

- Al virtualizador también se le llama *hipervisor*
- Se emula memoria virtual, disco y dispositivos

  Ejemplo: VMware emula tarjeta de audio SoundBlaster 16 y tarjeta ethernet AMD PCnet II. Cualquier aplicación en el *guest* percibe este hardware

- No se emula la CPU. Por tanto *guest* y *host* tienen que usar la misma arquitectura

VMware. Virtualizador. Software muy maduro. Versiones comerciales y versiones *freeware* (con los años va aumentando el número de versiones freeware)

- VMware Workstation, workstation player. Para host Windows y Linux. Permite crear y ejecutar máquinas virtuales
- VMware Fusion. Similar a VMWare Workstation, para Mac OS
- VMware ESXi. Verdarero Sistema Operativo. Se ejecuta directamente sobre el hardware
- VMware vSphere. Computación en la nube. Basado en ESXi

Parallels Desktop

- Virtualizador para los Mac OS basados en Intel
- *guest* soportados: Microsoft Windows, Linux, FreBSD, Sun Solaris y algunos otros

(los Mac anteriores a 2020 y posteriores a 2006, basados en Intel, pueden ejecutar Windows en nativo con Boot Camp)

VirtualBox

- Virtualizador, muy similar a VMware
- Desarrollado por Innotek. Sun compra Innotek en 2008. Oracle compra Sun en 2009
    - Virtual Box Open Source Edition
    - VirtualBox. Software Comercial. Gratuito para uso personal y académico

      Incluye alguna característica adicional, como soporte USB. sATA, iSCSI, Remote Display Protocol (RDP) Server

6.3.4. Paravirtualización

Paravirtualización

Similar a la virtualización, pero exige un versión ligeramente modificada del *guest*

El rendimiento es normalmente mayor que el de los tipos anteriores

Xen

- Muy extendido
- Hay una versión libre que permite Linux sobre Linux
- Hay versiones comerciales que permiten Windows sobre Windows
- Los drivers están paravirtualizados, son más eficientes. (En un virtualizador, los drivers son drivers hw normales)
- También hay que modificar el *guest* (Xen lo llama *Dom0*)

6.3.5. Virtualización nativa

Virtualización asistida por hardware

También llamada *virtualización nativa*

- Es una emulación completa, pero realizada por la CPU con lo que el rendimiento es próximo al nativo
- Exige soporte en la CPU. Para Intel aparece en el año 2006 con KVM: Kernel-based Virtual Machine. Infraestructura para virtualización completa del núcleo de Linux
- Soportado por Xen

Procesadores que lo soportan:

- Intel virtualization (VT-x)

  Pentium 4 6x2, Pentium D 9x0, Xeon 3xxx/5xxx/7xx, Intel Core, Intel Core 2, Intel Quad-Core. Algunos atom (serie Z5xx)

- AMD-V

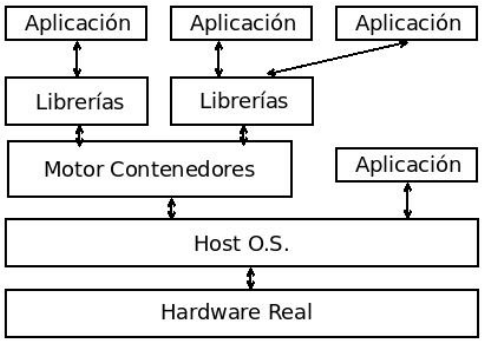
  AMD con Socket AM2, Socket S1 y Socket F. También procesadores Athlon 64 y Turion 64 a partir de mayo de 2006

6.4. Contenedores

6.4.1. Características de los contenedores

Contenedores

- Un contenedor es una encapsulación de una aplicación y todas sus dependencias
- Se pueden considerar una versión aligerada de las máquinas virtuales tradicionales
- Su nombre es una metáfora de los contenedores empleados en el transporte
  - Recipientes de carga que puede transportarse fácilmente en camión, barco o tren sin manipular la mercancía de su interior
  - Revolucionaron la industria en los años 1930
  - Desde los años 1970 son estándares mundiales
- Virtualización a nivel del sistema operativo
- Son máquinas virtuales de proceso, típicamente ejecutan un único proceso, como mucho unos pocos



Contenedores (diagrama simplificado)

Cada aplicacion se ejecuta en su propio contenedor. Cada una de ellas:

- Comparte el mismo sistema operativo
- Tiene la percepción de acceso exclusivo a los recursos
- No percibe a las demás

- No confundir con *web container*, también llamado *servlet container* que es algo completamente distinto:
    - En los servidores web basados en java, un contenedor web es el subsistema que interactúa con los servlets java
- Los contenedores son típicamente un orden de magnitud más eficientes que las máquinas virtuales tradicionales
    - Una máquina virtual suele tardar en arrancar muchos segundos o algunos minutos. Un contenedor, décimas de segundo
    - El rendimiento del proceso en ejecución es casi igual al nativo
- No son incompatibles con las máquinas virtuales tradicionales, al contrario, es muy habitual ejecutarlos dentro de máquinas virtuales
- Los contenedores solucionan el típico problema de *en mi máquina funcionaba*

El desarrollador

- Incluye dentro del contenedor todo lo necesario para que la aplicación se ejecute
- Sabe que la aplicación funcionará de forma idéntica en cualquier entorno (un servidor hardware tradicional, una máquina virtual, un servidor en la nube, un portátil...)

El administrador

- Pueden concentrarse en los recursos de red, sin perder tiempo configurando entornos y dependencias en el sistema

El poco peso de los contendores

- Permite ejecutar docenas de ellos simultáneamente en cualquier máquina
- Facilita su uso en la nube

Los contenedores son una tecnología que está cambiando la forma en la que se desarrolla, distribuye y ejecuta el software.

Historia de los contenedores

- Año 1979. Sistema chroot de Unix V7. En cierta forma pueden considerarse los primeros contenedores, aunque solo encapsulan el sistema de ficheros (no los procesos, ni los usuarios, ni la red...)
- Año 2000. BSD *Jails*
- Año 2001. Linux VServer (año 2001)
- Año 2004. Solaris Containers
- Año 2008. LXC (Linux Containers)
- Año 2013. Solomon Hykes libera Docker como software libre

Docker

Madurez de los contenedores: Docker

- Hasta la aparición de Docker, los contenedores eran una herramienta de nicho. Tenían su utilidad, pero no se puede decir que su uso fuera masivo
- Docker es una herramienta muy fácil de usar y muy eficiente, hace que los contenedores pasen a ser muy populares
- Los organismos que principalmente contribuyen a su desarrollo son, además del *docker team*, Cisco, Google, Huawei, IBM, Microsoft y Red Hat
- Está integrado con las principales plataformas y herramientas: Amazon Web Services, Ansible, CFEngine, Chef, Google Cloud Platform, IBM Bluemix, Jelastic, Jenkins, Kubernetes, Microsoft Azure, OpenStack Nova, Oracle Container Cloud Service, Puppet, Vagrant, VMware vSphere...

Fundamentos de Docker

Docker se basa en la funcionalidad de virtualización ofrecida por el nucleo de Linux:

- cgroups

  El término proviene de *control groups*. Año 2008. Permiten limitar y aislar los recursos consumidos por un grupo de procesos (CPU, memoria, E/S, red...)

- Linux kernel namespaces

  Grupos de procesos que no pueden *ver* los procesos de otros grupos. Quedan aislados los PID, los interfaces de red, las tablas de encaminamiento, el cortafuegos, el nombre de host, los puntos de montaje del sistema de ficheros, la intercomunicación entre procesos y los identificadores de usuario

Union Filesystem

Otra tecnología fundamental integrada en Docker es *Union Filesystem*, también llamado *Union Mounting*

- Docker soporta diversos *drivers* para el UFS: overlay2, aufs, OverlayFS, entre otros

  Emplear uno u otro depende fundamentalmente de la plataforma, el usuario de Docker normalmente no necesita ocuparse de esto

- Las imágenes de los contenedores están formadas por varias capas apiladas

  - Todas las capas son de solo lectura, excepto la última, que es de lectura y escritura
  - UFS permite que cada contenedor perciba todas las capas como un único sistema de ficheros ordinario
  - Pero cada capa de solo lectura puede ser compartida por varios contenedores, de forma que solo la capa de lectura/escritura es exclusiva para cada contenedor

Estas capas diferenciales apiladas representan un enorme ahorro de espacio en el sistema de ficheros

Ejemplo:

- 10 imágenes similares de una máquina virtual de 2 Gb ocuparán necesariamente 20 Gb

- 10 imágenes similares de un contenedor pueden ocupar 2.2 Gb

6.4.2. Inconvenientes de Docker

Inconvenientes de Docker (1)

Todos los contenedores comparten el mismo kernel con el host

Esta es la principal ventaja (son ligeros) pero también el principal inconveniente (no son muy seguros)

- Una vulnerabilidad o un kernel panic en un contenedor afecta a toda la máquina
- Cargar un módulo del kernel en el contenedor es cargarlo en el host
- Existe el riesgo de ataques DOS (Deny of Service)
- No hay un espacio de nombres propio para los usuarios en el contenedor. En Docker, si un usuario es root en el contenedor y consigue salir del contenedor, es root en el host

Inconvenientes de Docker (2)

En Docker, para poder lanzar un contener son necesarios privilegios de superusuario en el host. Es necesario

- O bien ser root o usar sudo
- O bien pertenecer al grupo *docker*, lo que resulta equivalente

Previsiblemente este problema se resolverá en futuras versiones de Docker, con un espacio de nombres propio para los UID

Inconvenientes de Docker (3)

Otros elementos compartidos por host y contenedores:

- Los dispositivos: discos, tarjetas gráficas, de sonido...
- La hora
- El anillo de claves del núcleo

6.4.3. Podman

Podman

Podman es una herramienta de gestión de contenedores muy similar a Docker

- Creada por Red Hat en 2017

- Cada vez más popular, aunque no tanto como Docker
- No exige privilegios de root
- No requiere usar demonios
- Más seguro que Docker
- Su interface de usuario básico es idéntico a Docker

Si tenemos un sistema funcionando con Docker, para migrar a Podman basta reemplazar la orden `docker` por la orden `podman`

- Incluso podemos hacerlo automáticamente

  ```
  alias docker=podman
  ```

- Así están configurados los laboratorios Linux de la EIF, cuando un estudiante escribe `docker` en realidad se ejecuta `podman`. Pero todo funciona exactamente igual

6.5. Otros tipos de virtualización

Otros tipos de virtualización

Como acabamos de ver, hipervisores, paravirtualizadores, virtualización nativa y, recientemente, contenedores, son las herramientas de virtualización más habituales en administración de sistemas

Pero hay muchas técnicas posibles, entre otras

- Máquinas virtuales cooperativas
- User Mode Linux

Máquinas Virtuales Cooperativas

- *Cooperative Virtual Machines.* UML y similares
- Término no demasiado extendido, acuñado para coLinux
- Dos sistemas operativos en paralelo acceden al Hw
- El Hw no se virtualiza
- No muy usado

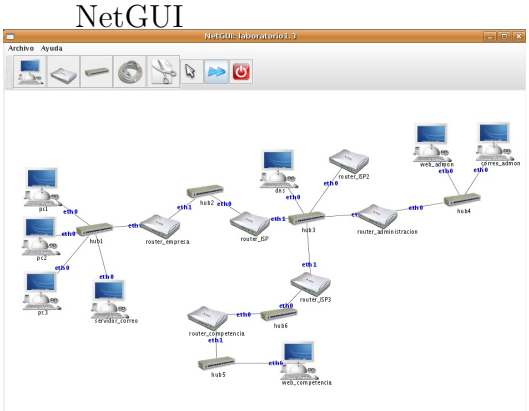
User Mode Linux

- UML. No confundir con *Unified Modeling Language*
- Es un tipo de máquina virtual muy diferente a las anteriores: Un nucleo Linux ligeramente modificado para ejecutarse como un proceso de usuario sobre otro nucleo Linux
- Permite ejecutar diferentes versiones de Linux sobre diferentes versiones de Linux
- Diseñado para Intel, hay versiones para IA-64 y PowerPC
- Los dispositivos del *guest* no están virtualizados. Por tanto en el *guest* se percibe el hardware real
- No muy usado

Netkit

- Entorno basado en UML para emular redes: PCs, routers, conmutadores
- Software libre, desarrollado por la Universidad de Roma

NetGUI



- *Front-end* gráfico para Netkit
- Desarrollado en GSyC

coLinux, AndLinux

coLinux:

- Año 2004. Basado en UML
- Actualmente en desuso

- Versión del núcleo de Linux que se ejecuta sobre otro S.O, como Windows

AndLinux

- Distribución basada en Ubuntu con versión del núcleo de Linux para ejecutarse sobre Windows 2000, XP, 2003, Vista, 7 (solo las versiones de 32 bits)
- Usa coLinux
- Algunos servicios van sobre Windows nativo: Servidor de X Window (Xming), servidor de sonido (Pulse Audio)

Windows Subsystem for Linux (WSL)

WSL 1

- Año 2016
- Capa de compatibilidad que permite hacer funcionar ejecutables Linux sobre Microsoft Windows. Análogo a Wine, pero al revés
- Mucho más ligero que una máquina virtual. El hardware no está virtualizado, el subsistema Linux comparte recursos con los procesos Windows
- El núcleo de Windows se modifica para ejecutar directamente procesos Linux

WSL 2

- Año 2019
- Incluye un verdadero núcleo de Linux
- El hardware se virtualiza, pero con una máquina muy ligera (ligera como p.e. un contenedor)
- Compatible con practicamente cualquier ejecutable Windows, no solo x64, también x32
- Permite usar módulos Linux convencionales

En ambos casos (WSL 1, WSL 2)

- Es una herramienta estándar en Windows, desarrollada por Microsoft. Gratuita (aunque no libre)
- Hay varias distribuciones Linux preparadas para ejecutarse en este entorno, siende Ubuntu la más habitual
- Permite usar aplicaciones gráficas mediante un servidor X Window para Microsoft Windows.
- El objetivo es que los programadores puedan desarrollar código para Linux en sus máquinas Windows con mejor rendimiento y mayor comodidad que en una máquina virtual tradicional. No poner servicios en producción
- Podríamos usarlo para esta asignatura, parece funcionar bastante bien. El problema es que es un Linux un poco *raro*, con algunas peculiaridades
    - A diferencia de un Linux dentro de una máquina virtual tradicional, prácticamente indistinguible de un Linux en nativo

6.6. Técnicas sin virtualización

Técnicas sin virtualización

Instalación, reconfiguración y replicación automática, independecia de la plataforma, portabilidad... son características deseables en una buena administración de sistemas

- Todas ellas pueden conseguirse mediante virtualización
- Pero la virtualización no es la única forma. Hay multitud de técnicas de administración alternativas que también ofrecen estas cualidades
- Un buen administrador de sistemas evaluará lo más adecuado para cada caso

Veremos a continuación alguna de estas técnicas

Jaulas chroot

- Se cambia el directorio raiz que percibe un proceso, (y sus hijos) de forma que no puede acceder fuera de cierto directorio.
- No se aisla el acceso a otros procesos, memoria, CPU, red u otros dispositivos

Simuladores

- Simulan algunas características del comportamiento externo de un sistema. P.e. simuladores de red (*GloMoSim*, *JSIM*, *ns-2*, *OPNET*, *OMNet*, etc)
- Los mal llamados *simulador de Zx-Spectrum para PC*, *simulador de Commodore 64 para PC*, etc, no son simuladores. Son emuladores completos.

Capas de Compatibilidad

- Wine. Reimplementación de la API de Win16 y Win32 para sistemas operativos basados en Unix bajo plataformas Intel. Permite ejecutar algunas aplicaciones para Windows en Linux.

  Cedega es un *fork* comercial de Wine
- Cygwin. Año 1995. Entorno para portar software POSIX a Windows, compuesto por:

  1. DLL que ofrece la funcionalidad de las llamadas al sistema de Linux
  2. Colección de herramientas habituales en sistemas Unix

  Siempre es necesario recompilar las aplicaciones

Implementación de protocolos de red

- En redes Windows los directorios e impresoras se exportan mediante los protocolos smb/cifs NetBIOS.

  Samba es una implementación de estos protocolos, permite usar máquinas Unix en redes Windows
- En Unix los directorios se exportan normalmente mediante NFS. Hay implementaciones de NFS para Windows. Permiten acceder a directorios Unix desde máquinas Windows

  - En Unix las impresoras se exportan normalmente mediante LPD (*Line Printer Daemon Protocol*). Estándar basado en TCP, RFC 1179.

    Windows entiende este procolo, no hace falta software adicional

Clonación

Permite replicar el disco de una máquina, y con ello todo su S.O. , configuración, aplicaciones y datos

- El administrador prepara *a mano* una máquina con el S.O completo, todo el software y toda la configuración necesaria.
- El disco de esta máquina se guarda en un fichero imagen
- El software de clonación se encarga de distribuir automáticamente la imagen entre todas las máquinas necesarias. También se puede usar la imagen para *refrescar* la máquina original en el futuro. De esta forma, tendremos máquinas con una configuración y estado idéntico al de la primera instalación manual
- Principal inconveniente de esta técnica: suele exigir máquinas idénticas (o muy parecidas)
- Las herramientas suelen poder clonar cualquier máquina, con independencia de su S.O. Ejemplo: desde Linux puedo clonar un disco Windows

Herramientas de clonación

- Hasleo backup. Soft gratuito para Windows
- Clonezilla. Libre, multiplataforma
- Rescuezilla. *Frontend* de Clonezilla
- Norton Ghost. Soft propietario para Windows
- Acronis True Image. Soft propietario para Windows
- Partition Saving. Freeware para Windows
- Partimage. Soft libre, basado en linux, permite clonar cualquier S.O. Viene incluido en *SystemRescueCd*, una distro *live* orientada a recuperar y reparar un sistema
- SystemImager. Soft libre para Linux.

  Uso típico: Se instala un PC, el *cliente de oro*. La imagen se almacena en el servidor. Esta imagen de distribuye por la red (local), clonando el PC. Si es necesario recuperar una imagen, solo se distribuyen los cambios

Instalación automática del S.O.

Sistema que contesta automáticamente a las preguntas que hace un SO en su instalación.

- *preseed* (debian)
- *kickstart (Red Hat)*
- *nLite* (Windows XP)
- *vLite* (Windows Vista)

Instalación automática de aplicaciones web

Librerías de scripts que instalan aplicaciones web

- Muy usadas por los servicios de hosting
- El interfaz de usuario suele ser via web
- Las aplicaciones que soportan suelen ser aplicaciones para el web
- Ejemplos: Softaculous, Installatron, Fantastico

Herramientas de administración centralizada

Herramientas que se encargan de que los ficheros de configuración se *mantengan* en cierto estado (sin necesidad de preparar scripts que busquen las inconsistencias y las corrijan)

- cfengine

  Herramienta tradicional, muy potente. Manejo de cierta complejidad

- landscape

  Para ubuntu. De pago

- spacewalk

  Para Red Hat y CentOS

- Puppet

  Herramienta muy popular. Basada en Ruby. Algo pesada

- Ansible

  Muy ligera, no necesita demonio en el cliente, solo ssh. En auge

- MSCCM (Microsoft System Center Configuration Manager)
  Herramienta nativa en Windows para administración centralizada
- Chef, Bcfg2, otras alternativas

7. Configuración de VirtualBox

7.1. Estructura de los laboratorios del GSyC

Estructura de los laboratorios del GSyC

- Para las prácticas de esta asignatura, tendrás una cuenta en los laboratorios Linux del Departamento GSyC
- La misma cuenta la usarás en las prácticas de muchas asignaturas del Departamento, durante toda la carrera/todo el máster

Para conocer la dirección IP de la máquina en la que estás trabajando puedes usar `hostname -i`

- Las direcciones IP de cada máquina pueden consultarse en el fichero `/etc/hosts` de cualquier equipo
  - Este fichero equivale a
    `%SystemRoot%\system32\drivers\etc\hosts` (MS Windows)
    `/private/etc/hosts` (Mac OS)
- La misma cuenta permite entrar en todas las máquinas
- Cada usuario verá el mismo *home* en todas las máquinas de Fuenlabrada
- Cada usuario verá el mismo *home* en todas las máquinas de Móstoles, distinto al de Fuenlabrada
- Los servidores están dimensionados para mover ficheros del orden de KBytes o MBytes, no GBytes
- Los ficheros de los directorios `/tmp/` y `/var/tmp` son locales a cada ordenador
- Como en todo linux,
  - El directorio `/tmp` se borra cada vez que se reinicia el ordenador
  - El directorio `/var/tmp` se borra cada vez que al administador le parece oportuno, sin que debamos esperar aviso previo

7.2. Imágenes de máquinas virtuales

Imágenes de máquinas virtuales

- Una de las ventajas de las máquinas virtuales es que pueden clonarse (copiarse) de un *host* a otro. Para ello basta copiar un fichero o ficheros: la *imagen de la máquina*
- VirtualBox llama a estas imágenes *servicio virtualizado*. En VirtualBox 4, es un fichero `.ova` [2]

Para clonar una máquina virtual de un *host* a otro

- En el *host* origen, exportamos la imagen indicando dónde queremos guardar el .ova
- Llevamos este ficheros al *host* destino
- Importaremos el .ova, esto generará automáticamente una nueva copia del disco duro virtual, en el directorio especificado en

```
Archivo|Preferencias|General|
Carpeta predeterminada de máquinas
```

- Podemos borrar el .ova, pero normalmente será preferible conservarlo por si queremos en el futuro otrá máquina *como nueva*

Observa que entonces tenemos 3 copias del disco duro virtual

1. La del *host* origen, en formato `.vmdk`
2. La *que viaja*, incluida dentro del fichero `.ova`
3. La del *host* destino, en formato `.vmdk`

Ejemplo típico de uso de máquinas virtuales

- Un profesor instala una máquina virtual partiendo de cero

  Crea una máquina, especifica su tamaño de disco, de memoria, etc
- El profesor exporta la máquina virtual como fichero .ova (que dentro lleva un .vmdk) y la deja en algún lugar, como p.e. el directorio `/var/lib/vms` de las máquinas de sus alumnos

[2] En VirtualBox 3 eran 3 ficheros: .ovf .vmdk .mf

- Cada alumno instala la máquina virtual en su VirtualBox partiendo de la imagen creada por el profesor
- Ahora el alumno podría exportar de nuevo la máquina virtual y llevársela en un *pendrive* al pc de su casa

Algo muy parecido podría hacerlo un administrador con los equipos de sus usuarios, o un administrador que quiera conservar un servidor recién instalado para recuperarlo rápidamente si hay problemas

Instalación de una m.v. partiendo de cero

Si ya contamos con una imagen de la m.v, podemos omitir estos pasos. Pero en otro caso:

1 Lanzamos `VirtualBox` desde la shell

2 Pulsamos el botón *nueva*, que ejecuta el asistente para crear una nueva máquina virtual

3 Indicamos el nombre de la m.v. (p.e. `pc01`, `ro01`, `auditor01`)

Sistema operativo Linux, Versión Ubuntu

4 El tamaño de memoria base dependerá de lo que tenga el *host* y necesite en *guest*. Como referencia:

- OpenWrt: 32 Mb
- Ubuntu Server: 192 Mb
- Ubuntu con gráficos, BackTrack con gráficos: 512 Mb

5 Activamos *Crear disco virtual* y seguimos el asistente

El tamaño del disco dependerá de lo que tenga el *host* y necesite en *guest*. Como referencia:

- OpenWrt: 64 Mb
- Ubuntu, Backtrack: 8Gb

6 En la ventana *resumen* revisamos todo y pulsamos *Terminar*

Una vez que hemos especificado los componentes de la máquina, habrá que instalar el sistema operativo, que normalmente tendremos en un cdrom/dvd o en una imagen iso de un cdrom/dvd

- En el apartado de configuración de la máquina virtual, en

  ```
  Almacenamiento | controlador IDE
  (pulsamos en el icono que representa un CD con el signo +)
  (pulsamos en el CD recién creado, llamado "vacio")
  (Vamos a "dispositivo cd/dvd")
  ```

  Aquí indicamos si usaremos el lector físico del *host* (anfitrión) o una imagen iso del cdrom/dvd

Instalación de una m.v. partiendo de una imagen

Desde VirtualBox

- Comprobamos en

  ```
  Archivo|Preferencias|General|
  Carpeta predeterminada de máquinas
  ```

  que el disco duro virtual quedará copiado en el lugar adecuado.

  - En casa, el lugar por omisión es válido:
    `~/VirtualBox VMs`
  - En el laboratorio, es imprescindible que sea `/var/tmp/tulogin`

- `Archivo|Importar servicio virtualizado|Seleccionar`

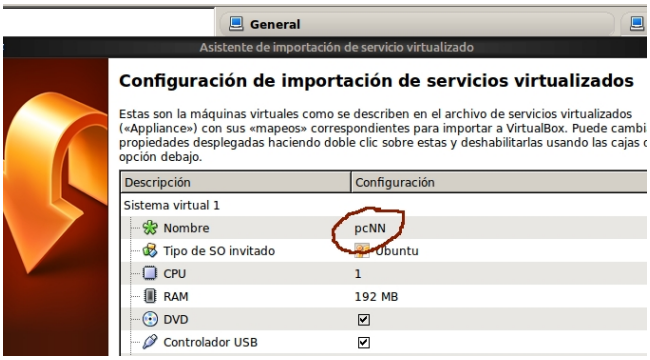
  (Elegimos el fichero .ova)

- En la ventana *Configuración de importación de servicios virtualizados* podemos cambiar algunos parámetros del *guest* (nombre, memoria, disco....)

- Si dos máquinas van a compartir segmento de red, es necesario cambiar su dirección MAC

- Si la imagen original se llama p.e. `pcNN`, haciendo clic sobre este nombre en la pantalla de configuración de importación, podemos cambiarlo. P.e. para llamarla `pc01`

Este es el nombre de la máquina visto desde VirtualBox.
Para cambiar el nombre visto desde dentro de la propia máquina, hay que



- O bien

  1. Editar `/etc/hostname`
  2. En ubuntu 18.04 y posteriores:
     Editar `/etc/cloud/cloud.cfg` para poner la opción `preserve_hostname` a `true`

  Esto es persistente pero tiene efecto en el próximo reinicio

- O bien ejecutar la orden

  `hostname <NUEVO_NOMBRE>`

  Esto es inmediato pero no es persistente

Fragmentación de ficheros

Si necesitas trocear una imagen de gran tamaño en ficheros que quepan en un *pendrive* o cdrom

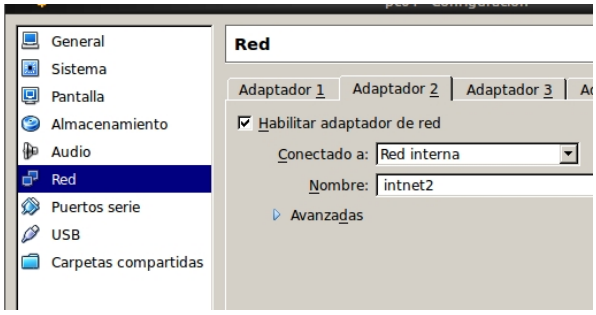
- Empaquetar y comprimir un directorio:
  `tar -cvzf mi_imagen.tgz mi_directorio`

- Mostrar contenido:
  `tar -tzf mi_imagen.tgz`

- Trocear:
  `# tamaño fichero prefijo split -b 500MB mi_imagen.tgz mi_ima`
  (Observa que el segundo parámetro es igual al primero, pero añadiendo un punto)

- Habremos generado

  ```
  mi_imagen.tgz.aa  mi_imagen.tgz.ab  mi_imagen.tgz.ac
  ```



En la máquina destino (no importa si en el *host* el S.O. es distinto)

- Unir los fragmentos `cat mi_imagen.tgz.* > mi_imagen.tgz`

  (En MS Windows para este paso podemos emplear HjSplit, Free File Splitter o cualquier otro programa similar)

- Descomprimir y desempaquetar:
  `tar -xvzf mi_imagen.tgz`

  (En MS Windows podemos usar 7-Zip o similares)

7.3. Interfaces de red en VirtualBox

Interfaces de red de VirtualBox

- Cada máquina virtual puede tener hasta 4 interfaces *aka* adaptadores de red

  adaptador 1 será eth0, *adaptador 2* será eth1, etc

- Cada interfaz puede conectarse a 5 tipos de segmento de red: No conectado, NAT, Adaptador puente, Red interna, Solo anfitrión

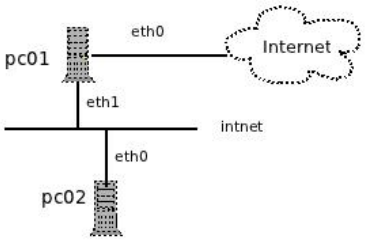
- Not attached / No conectado

  Emula una tarjeta con el cable de red desconectado

- Network Address Translation (NAT)

  Configuración por defecto. El *guest* tiene acceso al exterior (típicamente internet) a través de NAT. El *host* no tiene acceso al *guest*

  Podemos usar varios *guest*, pero cada uno tiene su propio NAT y está aislado en su propio segmento de red



- Bridged networking / Adaptador puente

  Interfaz en el *guest* conectado virtualmente al mismo *hub* (real) que el *host*

  El *guest* está en el segmento de red *normal* del *host*

- Internal networking / Red interna

  Red entre diferentes *guests* en un mismo *host*

  Sin acceso al *host* ni al exterior

- Host-only networking / Sólo anfitrión

  Red entre el *guest* y el *host*, sin acceso al exterior

  Permite tener varios *guests* en el mismo segmento de red

Supongamos que deseamos configurar dos *guest* de esta forma:

- En pc01, el interfaz eth0 estaría conectado a NAT

  Dentro de la máquina virtual, lo configuraríamos para obtener sus parámetros por DHCP

- En pc01, el interfaz eth1 lo conectaríamos a una red interna. El nombre por omisión de este segmento de red es *intnet* (atención, significa *internal net*, no *internet*)

  Podemos ponerle el nombre que queramos al segmento

- Dentro de la máquina virtual pc01, configuraríamos estáticamente los parámetros de eth1
- En pc02, conectaríamos eth0 a una red interna, con el mismo nombre que la red interna de eth1 en pc01 (en este ejemplo, *intnet*)
- Dentro de pc02, configuraríamos estáticamente los parámetros de eth0

7.4. Configuración que seguiremos en prácticas

Uso de VirtualBox en los laboratorios del GSyC

Un disco duro virtual será típicamente un fichero de varios GBytes almacenado en

`~/VirtualBox VMs`

- En tu PC esto no será un problema
- En el laboratorio sí, el rendimiento sería muy pobre. Por tanto cambiaremos la ubicación por omisión de los discos duros virtuales
  - En el *host*

    `mkdir /var/tmp/tulogin`

    (Donde *tulogin* es tu usuario del laboratorio, p.e. mgarcia, jperez...)
  - En VirtualBox:

    `Archivo|Preferencias|General|`
    `Carpeta predeterminada de máquinas`

    Indicamos

    `/var/tmp/tulogin`

    Muy importante: asegúrate de cambiar esta preferencia y mantenerla siempre. De lo contrario, cargarás mucho el servidor, perjudicandote a tí y a tus compañeros

Problema: las imágenes son ficheros grandes

Con lo visto hasta ahora, ya podríamos hacer las prácticas de la asignatura. Pero sería poco práctico.

Supongamos una práctica que consista en configurar en red 3 máquinas virtuales

- Cada alumno tendría que guardar en su cuenta del laboratorio 3 imágenes (con sus 3 discos duros virtuales completos)

- Para trabajar en casa, tendría que llevarse las 3 imágenes con sus 3 discos duros virtuales completos
- Para que 70 alumnos entreguen su práctica, el profesor tendría que manejar 210 discos duros virtuales completos

Si en la asignatura se hacen 2 o 3 prácticas, seguimos multiplicando...

Solución: almacenar solo los ficheros importantes

Administrar un Unix/Linux consiste en editar diversos ficheros de texto

- Solamente manejaremos estos ficheros, que estarán guardados en la cuenta de bilo y respaldados en la nube de Dropbox
- Las máquinas virtuales serán *de usar y tirar*, tomarán la configuración de estos ficheros
- Dentro de las máquinas virtuales, los ficheros de configuración serán enlaces simbólicos a ficheros en un directorio, que a su vez estará montado por red desde un directorio en el laboratorio

Ejemplo:

Para configurar los interfaces de red de una máquina, hay que editar `/etc/network/interfaces`

- Dentro de la máquina virtual `pc01`, este fichero será un enlace simbólico que apuntará a `/media/nube/interfaces`
- El directorio `/media/nube` de `pc01`, estará montado desde el directorio `~/Dropbox/pc01` del laboratorio
- Por tanto, `/etc/network/interfaces` en la máquina virtual pc01 y `~/Dropbox/pc01/interfaces` en el laboratorio serán el mismo fichero, podrá editarse indistintamente cualquiera de los dos

Para montar el directorio remoto, aquí emplearemos sshfs
Ventaja principal:

- Basta con tener acceso por ssh a la máquina remota para poder montar un directorio

Pero esta idea de tener los ficheros importantes por separado y luego colocarlos automáticamente en su sitio puede aplicarse de muchas otras formas

- Tanto en máquinas físicas como virtuales

- En un entorno docente, doméstico, de oficina, granja de servidores...
- Mediante nfs, o scp, o rsync, o unison, o smb/cifs, o vboxsf...
- Con la oportuna atención a la seguridad si se trata de un sistema en producción

Montar un directorio con sshfs

Punto de montaje: directorio local donde veremos el directorio remoto

- Montar el *home* remoto:

```
sshfs  usuario@maquina: /punto/de/montaje
```

- Montar un directorio remoto cualquiera

```
sshfs  usuario@maquina:/un/directorio  /punto/de/montaje
```

  (Siempre path absoluto, no soporta ~)

- Desmontar:

```
fusermount -u /punto/de/montaje
```

No siempre es necesario tener privilegios de root (es configurable)

En conexiones lentas puede ser conveniente añadir la opción -C para que comprima el tráfico

```
sshfs -C usuario@maquina:/path  /punto/de/montaje
```

7.5. Cambio de host en el laboratorio

Cambio de host en el laboratorio

La m.v. está en un directorio local del pc donde trabajas, no en tu cuenta de pantuflo/bilo

Si te sientas en un puesto del laboratorio distinto al del dia anterior:

1. Sal de VirtualBox y borra la máquina vieja

```
rm -rf ~/.VirtualBox
rm -rf /var/tmp/tulogin/*  # si este directorio existe
```

2. Vuelve a indicar en

   ```
   Archivo|Preferencias|General|
   Carpeta predeterminada de máquinas
   ```

   que la carpeta predetermina de máquinas tiene que ser `/var/tmp/tulogin`

3. Vuelve a importar el servicio virtualizado (O copia `/var/tmp/tulogin` desde el host anterior)

Observaciones

- Recuerda que el administrador puede borrar tu máquina virtual en cualquier momento, no guardes dentro nada de valor, todos tus ficheros deben estar en el directorio compartido
- En esta asignatura, cada máquina Ubuntu tendrá por omisión un usuario de nombre `user` y contraseña `user` autorizado a ejecutar la orden `sudo`
  - Recuerda que en el caso de Ubuntu, se espera que no empleemos el usuario `root`
- En las máquinas sin gráficos, podemos usar varias consolas pulsando `Alt F1`, `Alt F2`, etc
- Si dejamos la máquina virtual desatendida algunos minutos, puede saltar el salvapantallas y quedarse en negro. En tal caso, llevamos el foco a la máquina virtual (haciendo clic dentro) y pulsamos cualquier tecla

Algunos errores posibles

Si has empezado a importar una máquina virtual, te has equivocado en algo y has vuelto a empezar, VirtualBox puede mostrará un error indicando que ese disco duro ya está registrado y no puede importarse de nuevo

Soluciones

1. En

   ```
   Archivo | Administrador de medios virtuales
   ```

Elimina esa imagen de la lista de medios conocidos, o elimínala por completo (una ventana te informará). Fíjate si es la imagen *que viaja* o la del *host destino*

2. Alternativa más drástica: Cerrar VirtualBox y borrar todo el directorio `~/.VirtualBox`

   (esto elimina toda la configuración y todas las máquinas)

Si intentamos usar dos instancias de un *guest* en el mismo *host* nos dará un error indicando que ambos discos tienen el mismo identificador. En este caso, hay que clonar el disco

Ejecutamos desde la shell

```
VBoxManage clonehd   <filename> <outputfilename>
```

Reiniciar VirtualBox

- Para borrarlo todo y volver a empezar, elimina los directorios `~/.VirtualBox` y `/var/tmp/tulogin`

  Pero recuerda volver a indicar `/var/tmp/tulogin` en

  ```
  Archivo|Preferencias|General|
  Carpeta predeterminada de máquinas
  ```

7.6. Configuración del teclado

Configuración del teclado

- El teclado habitual en los PCs españoles es el pc105 (O el pc102 si no tiene teclas *menú*, *windows*)
- El equivalente en los PCs estadounidenses es el pc104 y el pc101, respectivamente
- Programadores y usuarios normalmente trabaja con versiones del SO adaptadas a su idioma
- Un administrador frecuentemente se encontrará con un SSOO en inglés
  - Normalmente podrá configurarlo para que admita su propio idioma (si no en menús y documentación, sí en la configuración del teclado)
  - Pero mientras lo configura, tendrá que saber manejarse mínimamente con el teclado desconfigurado

Si el ordenador tiene X Window (Gráficos), podemos configurarlo con

- `setxkbmap us` Fija el teclado en la disposición pc104
- `setxkbmap es` Fija el teclado en la disposición pc105

En Debian/Ubuntu podemos usar

- `dpkg-reconfigure console-setup`
- O más globalmente

  `sudo dpkg-reconfigure locales`

En gnome:

- `Sistema | Preferencias | Teclado | Distribuciones |`
  `| Añadir | Español | Subir (hasta colocarlo el primero)`

  Necesitarás cambiar esto por ejemplo si entras desde casa al laboratorio por VNC y lanzas una máquina virtual

En OpenWrt esto no está disponible, la mejor opción es entrar por ssh (lo que exige que la red ya funcione)

Si nuestro teclado es español, pero el sistema operativo espera un teclado norteamericano:

```
Obtener    Pulsar
--------------------------
Esc        Esc
:          Ñ
;          ñ
/          _
!          !
|          shift Ç
'          ,    (apóstofre, coma)
```

8. Docker

8.1. Prerrequisitos

Plataformas para ejecuta docker

Docker tiene arquitectura cliente servidor

- El cliente acepta la entrada del usuario, le muestra la salida, maneja los ficheros con los que preparar las imágenes
- El servidor ejecuta el contenedor

El servidor solo está disponible para Linux 64 bits

Hay versiones para macOS y Microsoft Windows, donde el cliente se ejecuta en nativo contra un servidor dentro de una máquina virtual

- Esta virtualización es transparente para el usuario

Docker dentro de una máquina virtual

Si vamos a ejecutar docker dentro de una máquina virtual,

- guest y host deben tener arquitectura 64 bits
- Es necesario que el host tenga soporte para Intel VT-x
  - Los equipos muy antiguos o muy baratos no lo permiten (VT-x es del año 2006, pero no se generaliza en los equipos de gama básica/media hasta varios años después)
  - Muchos equipos actuales tienen esta opción deshabilitada por omisión en la BIOS/UEFI

Algunos conceptos

Imágenes:

- La imagen de un contenedor (o simplemente *imagen*) es un fichero en el sistema de ficheros del host
- Un contenedor se ejecuta a partir de una imagen

Esto es análogo a un proceso que se ejecuta a partir de un fichero

Manejaremos diversos identificadores, que no debemos confundir

- Nombre de la imagen. Ejemplos:

  debian

  test/c01

- Identificador de la imagen. Ejemplo:

  cc8393a39248

- Nombre de contenedor. Si no lo indicamos explícitamente, docker usará nombres aleatorios como *focused_yonath* o *wonderfuld_goldberg*

- Identificador de contenedor

  Ejemplo: 18009dd9f349

- Nombre de host

  Nombre de máquina que se percibirá dentro del contenedor. (variable de entorno `$HOST`, nombre en el *prompt*, fichero `/etc/hostname`, etc)

  Atención: Este *host* de Docker se corresponde con lo que en VirtualBox sería el *guest*

Nombres de imagen

El nombre de la imagen

- Un nombre sin prefijo, por ejemplo *debian* indica una imagen oficial aprobada por docker

- Un nombre con prefijo, por ejemplo *test/c01* es una imagen no oficial. El prefijo puede ser una etiqueta que hayamos definido o un nombre de usuario en un registro de imágenes

8.2. Instalación de Docker

Instalación de docker

- Podemos instalar el paquete incluido en nuestra distribución de ubuntu

```
apt update; apt upgrade -y ; apt install docker.io
```

- Si por algún motivo ese paquete resulta antiguo y necesitamos la última versión estable disponible de docker, ejecutamos el script disponible en `https://get.docker.com`

```
wget https://get.docker.com -O get-docker.sh  #letra "O" mayúscula
bash get-docker.sh
```

8.3. Ejecución de imágenes

Lanzar una imagen

Para ejecutar docker, tenemos dos opciones

- Añadir nuestro usuario al grupo docker

```
addgroup docker
adduser $USER docker
# (abrir una nueva sesión)
```

- Ejecutar docker con sudo

Para comprobar que la instalación ha sido correcta, lanzamos una imagen sencilla

```
docker run debian echo "hola,mundo"
```

Esto busca en el *registry* oficial de docker una imagen llamada *debian*, ejecuta en ella la orden indicada, muestra su salida por stdout y concluye

Otro holamundo

```
koji@mazinger:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
5b0f327be733: Pull complete
Digest: sha256:1f19634d26995c320618d94e6f29c09c6589d5df3c063287a00e6de8458f8242
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash
```

Servidor docker remoto

- En la configuración más sencilla, el servidor de docker está en la misma máquina que el cliente, el ejecutable incluye ambas funciones
- Pero también puede ubicarse en una máquina remota. Esto es útil, por ejemplo
  - Cuando el cliente no es linux 64 bits
  - Cuando el cliente no tiene privilegios de root en la máquina local
  - Para arquitecturas distribuidas, equilibrio de carga, en la nube, etc

Configuración del servidor remoto

- Es necesario el paquete `docker.io`
- El usuario tiene que poder entrar por ssh en la máquina remota
- Debería poder autenticarse por ssh sin escribir la contraseña cada vez (lo contrario sería muy incómodo)
- El usuario debe pertenecer al grupo `docker`

Configuración del cliente

- Es necesario el paquete `docker.io`
- El usuario necesita la variable de entorno

```
DOCKER_HOST="ssh://jperez@servidor_remoto"
```

Recuerda que:

- Puedes crear la variable de entorno en `~/.bashrc`. Pero los cambios no son inmediatos, es necesario una nueva sesión o leer el fichero explícitamente con `source`
- Puedes comprobar las variable de entorno con `env`

Podman sobre NFS

Para usar Podman sobre NFS, como en el laboratorio, es necesario configurar el sistema para que los volúmenes vaya a un disco local, p.e. `/tmp/MILOGIN`

Para ello, creamos un fichero

`~/.config/containers/storage.conf`

con el siguiente contenido:

```
[storage]
  driver = "vfs"
  runroot = "/run/user/NNNN"
  graphroot = "/tmp/MILOGIN/.local/share/containers/storage"
  [storage.options]
    ignore_chown_errors = "true"
    mount_program = ""
```

- Reemplazando `MILOGIN` por nuestro nombre de usuario
- Reemplazando `NNNN` por nuestro uid[3]

Contenedores dentro de máquina virtual VirtualBox / Vagrant

- Un desarrollador trabajando en su propio portatil, puede crear y lanzar contenedores con tranquilidad
- En un entorno en producción, como el del laboratorio, esto no es conveniente
  - Hay muchos alumnos de muchas asignaturas
  - Está basado en NFS
- En estos casos, los normal es lanzar los contenedores dentro de una máquina virtual

  En esta asignatura, dentro de una máquina VirtualBox lanzada con Vagrant

Recuerda que si en el laboratorio tenemos

`~/lagrs/vbox01`

Y desde allí lanzamos una máquina virtual de VirtualBox con Vagrant

- Dentro de la máquina vagrant el directorio

  `/vagrant`
- Se corresponderá con el directorio del laboratorio

  `~/lagrs/vbox01`

[3] que podemos conocer con el comando id

Repositorio de imágenes

Además de guardarse localmente, las imágenes están disponibles en los *registry*

- Registro (Registry)

  Servicio responsable de almacenar y distribuir imágenes. El registro por omisión es `https://hub.docker.com`

  Aunque hay otros similares, públicos. Y quien lo desee puede establecer su propio registro

- Repositorio (Repository)

  Una colección de imágenes relacionadas, normalmente ofrecen diferentes versiones de la misma aplicación o servicio

- Etiqueta (Tag)

  Identificador alfanumérico asociado a una única imagen

Docker run

Esta instrucción lanza un contenedor a partir de una imagen
`docker run <opciones> <imagen>`

- Observa que las opciones deben escribirse antes del nombre de imagen. De lo contrario, docker las ignora

- La imagen se puede identificar mediante su nombre o mediante su id

- Las opciones `-i` y `-t` normalmente se usan juntas, para indicar que queremos una sesión interactiva en un terminal

- `--name <nombre_contenedor>`

- `-h <nombre_host>`

  `--hostname=<nombre_host>`

Ejemplo

```
docker run -it --name c01 -h c01 test/im01
```

Consulta de imágenes y contenedores

- `docker ps`

  Muestra los contenedores

- `docker images`

  Muestra las imágenes

- `docker inspect <imagen>`

  Muestra un json con descripción detallada del contenedor

- `docker diff <imagen>`

  Muestra los cambios en el sistema de ficheros del contenedor

- `docker logs <imagen>`

  Muestra las instrucciones ejecutadas en el contenedor

Exited containers

Cuando un contenedor finaliza su ejecución, queda en estado *exited*, al que informalmente se suele llama *parado*

- `docker ps -a`

  Muestra los contenedores, incluyendo los parados

- `docker rm <contenedor>`

  Borra un contenedor

- `docker rmi <imagen>`

  Borra una imagen

Si el contenedor se lanza con la opción `--rm`, se borrará automáticamente al concluir

Borrado de imágenes y contenedores

- Borrar todas las imágenes (que no estén siendo usadas)

  `docker rmi $(docker images -a -q)`

- Borrar todos los contenedores detenidos

  `docker rm $(docker ps -a -f status=exited -q)`

- Borrar todos los contenedores creados (y nunca ejecutados)

  `docker rm $(docker ps -a -f status=created -q)`

8.4. Creación de imágenes

Creación de imágenes

La orden `docker build` nos permite construir imágenes.

Para construir una imagen, normalmente usaremos tres cosas:

- Un directorio contexto, que será un directorio vacio en nuestra máquina, donde iremos añadiendo los ficheros necesario para construir la imagen
- Un fichero `Dockerfile` dentro del directorio contexto, con las instrucciones para crear la imagen
- Un fichero `entrypoint.sh`, que será un script de shell que
  - Crearemos en el directorio contexto
  - Llevaremos a la imagen
  - Se ejecutará cada vez que se lance un contenedor con esa imagen

- Si la imagen es muy sencilla, puede que no necesite `entrypoint.sh`

  Ejemplo:

```
FROM ubuntu:22.04
RUN apt update && apt upgrade -y
ENTRYPOINT /bin/bash
```

- También es posible crear una imagen sin usar un fichero Dockerfile

  Para ello basta con

  1. Entrar en el contenedor
  2. Configurarlos: instalar paquetes, añadir ficheros, modificar ficheros...
  3. `docker commit <CONTENEDOR> <IMAGEN>`

     `<CONTENEDOR>`: Nombre o id del contenedor que será punto de partida da la imagen

     `<IMAGEN>`: Nombre que tendrá la imagen

Ejemplo: banner

Vamos a crear una imagen llamada *test/banner* basada en la orden *banner* que al ejecutarse mostrará los siguiente:

```
koji@mazinger:~/lagrs/banner$ docker run -h c01 --name c01 test/banner

 #####     #   ######  #    #  #    #  ######  #    #     #    #####   ####
 #    #    #   #       ##   #  #    #  #       ##   #     #    #    #  #    #
 #####     #   #####   # #  #  #    #  #####   # #  #     #    #    #  #    #
 #    #    #   #       #  # #  #    #  #       #  # #     #    #    #  #    #
 #    #    #   #       #   ##   #  #   #       #   ##     #    #    #  #    #
 #####     #   ######  #    #    ##    ######  #    #     #    #####   ####


   ##
  #  #
 #    #
 ######
 #    #
 #    #

         ###      #
  ####   #   #    ##
 #    # # #   #  # #
 #      #  #  #    #
 #      #   # #    #
 #    #  #   #     #
  ####    ###    #####
```

Creamos un *directorio contexto* en el host, y en él escribimos un fichero `entrypoint.sh`

```
#!/bin/bash
banner bienvenido
banner a
banner $HOSTNAME
```

Cuando sea posible, es muy conveniente probar este script antes de construir la imagen, los errores aquí son uno de los problemas más habituales preparando contenedores

En el directorio contexto también creamos un fichero `Dockerfile`

```
FROM ubuntu:22.04
RUN apt update && apt upgrade -y && apt install -y sysvbanner
COPY entrypoint.sh /
ENTRYPOINT ["/entrypoint.sh"]
```

- La instrucción FROM indica la imagen de partida
- La instrucción RUN indica las modificaciones a realizar en la imagen

  Puede haber más de un RUN, pero eso crea imágenes intermedias, por lo que lo habitual es encadenar varias órdenes de shell con `&&`

- Aunque todas las instrucciones `apt` tiene que estar todas en la misma sentencia RUN, para asegurarnos de que se encadenan correctamente

- La opción `-y` en

  `apt upgrade`

  `apt install`

  es imprescindible (contesta a todas las preguntas con *yes*)

- La instrucción COPY copia un fichero desde el directorio contexto (que está en el host) hasta el sistema de ficheros del futuro contenedor que se ejecute a partir de la imagen

- La instrucción ENTRYPOINT especifica el fichero que se ejecutará al iniciar cada contenedor

  Es habitual llamarlo `entrypoint.sh` y colocarlo en el directorio raiz del contenedor

- En el Dockerfile se pueden crear comentarios con el caracter `#`

- El contenido del Dockerfile es *case insensitive*, aunque el convenio es usar mayúsculas para las instrucciones

- Si no existe un fichero `Dockerfile`, docker busca un fichero `dockerfile`

Una vez preparados los ficheros, construimos la imagen

- Desde el directorio padre del directorio contexto ejecutamos

  `docker build -t test/banner directorio_contexto`

Recuerda que los nombres de las imágenes que crearemos siempre llevarán prefijo (puesto que no son imágenes oficiales)

Almacenamiento de la configuración:

- La configuración de docker se guarda en /var/lib/docker

- Las imágenes, depende del driver que docker use para el almacenamiento. Por omisión se usa aufs, que guarda las imágenes en `/var/lib/docker/aufs`

La sentencia COPY

- Con mucha frecuencia querremos tener cierto fichero dentro de la imagen. En el Dockerfile escribiremos algo como

  ```
  COPY  ejemplo /home/jperez
  ```

  Esto es, indicamos nombre del fichero (en el directorio contexto) y directorio destino (en la imagen). El nombre del fichero no cambia.

- Otras veces será conveniente que el nombre del fichero en el directorio contexto sea distinto al nombre en la imagen. Típicamente si se trata de un fichero de configuración oculto

  ```
  COPY  ejemplo_config  /home/jperez/.ejemplo_config
  ```

  Esto es, el primer argumento de COPY será el nombre del fichero, no oculto, y el segundo argumento, el trayecto completo del fichero *en su sitio*, con el nombre precedido por un punto para que esté oculto.

El fichero entrypoint

- El fichero *entrypoint* normalmente será un script de shell, con el nombre `entrypoint.sh`. Aunque podría ser cualquier otro fichero que indiquemos en la instruccion `ENTRYPOINT` del Dockefile

- Como cualquier script de shell

  - Necesita permiso de ejecución
  - Su primera línea debe ser exactamente
    `#!/bin/bash`
    Y ninguna otra cosa
    `#! /bin/bash  # MAL`
    `!#/bin/bash   # MAL`
    `/bin/bash   # MAL`

- Si queremos que el usuario trabaje de manera interactiva dentro del terminal, añadimos una llamada a la shell dentro del entrypoint

```
#!/bin/bash
/bin/bash
```

Recuerda que, además, es necesario añadir las opciones `-it` en `docker run`

- Para que se ejecute algo en el terminal *antes* de que el usuario empieze a trabajar, lo añadimos en el entrypoint *antes* de la llamada a la shell

```
#!/bin/bash
echo Esto se ejecuta ANTES
/bin/bash
echo Esto se ejecuta cuando concluye la sesión
echo justo antes de finalizar el contenedor
```

Normalmente querremos que `/bin/bash` sea la última línea del entrypoint

Gestión de datos en docker

El sistema de ficheros interior al contenedor es volátil

- Todo lo escrito durante la ejecución del contenedor se pierde al borrar el contenedor
- Es complicado acceder a esos datos sin usar el mismo contenedor

Podríamos guardar datos en una nueva capa creando una nueva imagen, pero sería poco práctico, no es recomendable

Un contenedor no debería tener estado. O en su defecto, el mínimo estado posible

Docker ofrece 3 mecanismos para la persistencia de datos

- Bind mounts
- Volumes
- tmpfs

Por supuesto, dentro del contenedor se puede usar cualquier otro protocolo o servicio no específico de Docker: NFS, sshfs, SMB, rsync, almacenamiento en la nube, bases de datos relacionales, bases de datos no relacionales...

Bind mounts

Un *bind mount* es un directorio del host que se comparte con uno (o varios) contenedores

- Muy eficientes
- Muy prácticos para compartir datos con el host
- Dependen del sistema de ficheros del host y de su estructura, con lo que tienen problemas de portabilidad
- Evidentes problemas potenciales de seguridad, al tener el contenedor acceso directo al sistema de ficheros del host

Para hacer un bind mount, basta añadir los siguientes parámetros a la orden `docker run`

- Sintaxis tradicional

  `-v <DIR_ORIGEN>:<DIR_DESTINO>`

- Sintaxis moderna, disponible a partir de Docker 17.06

  `--mount type=bind,source=<DIR_ORIGEN>,target=<DIR_DESTINO>`

- `DIR_ORIGEN` es el directorio en el host
- `DIR_DESTINO` es el directorio en el contenedor
  - En el montaje de ficheros tradicional en Unix, es necesario que exista el punto de montaje. Aquí, no
- Ambos directorios deben estar especificados con path absoluto
- No puede haber espacios antes ni después de la coma

Suponiendo que los nombres de usuario coincidan en el host y en el contenedor, podríamos hacer por ejemplo

```
docker run -it -h jperbind01  --name jperbind01 --rm \
   -v $HOME:/home/$USER  \
   jperez/bind
```

Es necesario prestar mucha atención a los montajes bind, son potencialmente peligrosos. El usuario que accede al sistema de ficheros fichero en el servidor de contenedores es el mismo que en el contenedor

Ejemplos

- Un proceso que sea root en el contenedor, también puede acceder como root al servidor. Por eso es tan delicado que un usuario pueda lanzar un contenedor
- Supongamos el *home* de un usuario del servidor configurado para que solo él tenga acceso
  - Para que un usuario del contenedor pueda acceder a este directorio con un montaje bind, el usuario dentro del docker tiene que tener el mismo id que el usuario en el servidor (no importa el nombre, solo el id)
  - Sucede lo mismo con el gid: el gid del usuario dentro del docker será el gid que vea el servidor

Una vez más: los montajes bind son potencialmente peligrosos

Diferencia importante:

- En las máquinas virtuales *tradicionales* (p.e. hipervisores) es extremadamente difícil que un proceso del *guest* consiga *escaparse* y acceder al *host*
- En los contenedores, muy fácil

Volumen

Es un disco virtual creado y gestionado por docker.

Se puede almacenar

- Como subdirectorio del host (en linux, por omisión en `/var/lib/docker/volumes`)

  Aunque no se recomienda que el host acceda directamente al volumen
- En host remotos o en la nube, Docker ofrece para ello diferentes drivers

Características:

- Son más fáciles de transportar y respaldar que los bind mounts
- Tienen mejores prestaciones para ser compartidos entre varios contenedores
- Se pueden cifrar

tmpfs

Un montaje de tipo *tmpfs* se usa para datos temporales

- Es un sistema de ficheros especialmente eficiente porque se almacena en RAM
- Si creamos una imagen a partir del contenedor, el contenido de los montajes tmpfs no se almacena

Uso de sshfs

Como hemos visto, los bind mounts permiten montar dentro de un contenedor directorios ubicados en el host docker

- Para montar directorios en cualquier otro lugar de internet, podemos usar por ejemplo sshfs
- Para ello es necesario añadir a la orden `docker run` los siguientes parámetros
  - En docker 17.10

    `--privileged`
  - En versiones más modernas de docker

    ```
    --cap-add SYS_ADMIN --device /dev/fuse
    --security-opt apparmor:unconfined
    ```

  Para averiguar tu versión de docker: `docker --version`

  - En un entorno de producción habría que usar estas opciones con precaución, puesto que incremeta mucho los privilegios del contenedor dentro del host

docker hub

Para subir nuestras imágenes al registro docker hub

1. Creamos una cuenta en `hub.docker.com`
2. Creamos nuestras imágenes usando como prefijo nuestro login en dockerhub

   `docker build -t mi_usuario/mi_imagen`
3. Abrimos una sesión en docker hub desde la shell con la orden

   `docker login`

4. Subimos la imagen

   `docker push mi_usuario/mi_imagen`

Usuarios dentro del contenedor

La orden para crear usuario en Unix/Linux es *adduser*

- Solicita de forma interactiva contraseña, nombre, etc
- Para construir una imagen con *docker build*, usamos otra orden distinta, *useradd*, que no hace preguntas sino que permite introducir la información mediante opciones

En el Dockerfile añadimos

```
RUN useradd -rm -d /home/jperez -s /bin/bash -u 1001 jperez
```

- `-rm`

  Cuenta de sistema, con directorio *home*

- `-d`

  Directorio *home*

- `-s`

  Especifica la shell

- `-u`

  Especifica uid

Si queremos que el usuario pueda ejecutar `sudo`

- Instalamos el paquete *sudo*
- Asignamos al usuario el grupo primario *root*, y el grupo adicional *sudo*, añadiendo a `useradd` las opciones

  ```
  -g root -G sudo
  ```

Si queremos que al poner en marcha el contenedor con una sesión interactiva sea este el usuario, añadimos al final del *entrypoint*

```
su jperez
/bin/bash
```

Tendremos el usuario ejecutando una shell sin necesidad de escribir contraseña, pero si queremos que tenga contraseña, añadimos al *Dockerfile*

```
RUN echo 'jperez:sesamo' | chpasswd
```

8.5. Networking

Configuración de red

Al instalar docker se crean 3 redes

- bridge

  Segmento privado dentro del host, `172.17.0.0/16`, al que se conectan por omisión todos los contenedores

- null

  Red nula, aisla los contenedores de la red

- host

  El contenedor comparte la red con el host, mismos interfaces, direcciones y puertos

```
koji@mazinger:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
787cf305d42c        bridge              bridge              local
256d470b6133        host                host                local
086e801223bb        none                null                local
```

- Para conectar un contenedor a una red, basta lanzarlo con `--network=<nombre_red>`

  Ejemplo

  ```
  docker run -it -h c03 --name c03 --rm  --network=host test/im03
  ```

- Para crear una nueva red (un nuevo segmento), ejecutamos en la shell, directamente o en un script:

  ```
  docker network create --subnet 192.168.12.1/24  mired
  ```

Servidor de SSH en el contenedor

Para un contenedor en producción, no es recomendable habilitar el demonio de ssh

- Implica tener un segundo proceso, que no es natural en Docker
- No es buena idea dejar contraseñas dentro de un contenedor ¿cómo actualizarlas?
- El código dentro del contenedor es responsabilidad del equipo de desarrollo. Pero el acceso y las políticas, compete a explotación

Sin embargo, en esta asignatura sí configuraremos un servidor de ssh dentro de un contenedor, porque el objetivo es enseñar cómo funciona el acceso por ssh, que es lo habitual en máquinas físicas y máquinas virtuales tradicionales

¿Es necesario acceder por ssh?

- Para actualizar el sistema

  No. El contenedor entonces tendría estado (las actualizaciones). Lo recomendable es crear un nuevo contenedor con la actualización.

- Para ver logs

  No. El contenedor tendría estado. Lo recomendable es llevar los logs a un volumen

- Para iniciar y detener demonios

  No. Se pueden enviar señales

- Para editar la configuración

  No. Lo recomendable es crear un nuevo contenedor

- Para depurar el servicio

  No. Se puede abrir una shell desde el servidor de contenedores

Si a pesar de esto queremos instalar sshd en un contenedor:
Dockerfile

```
FROM ubuntu:22.04
RUN apt update && apt install -y openssh-server

# Con sshd, ENV no funciona. Para fijar una variable de entorno:
# RUN echo "export MI_VARIABLE=mivalor" >> /etc/profile

RUN service ssh start

COPY entrypoint.sh /

EXPOSE 22
ENTRYPOINT ["/entrypoint.sh"]
```

entrypoint.sh

```
#!/bin/bash
/usr/sbin/sshd
```

- La instrucción EXPOSE indica en qué puerto (TCP) atiende peticiones el contenedor
- Realmente esta instrucción no hace nada, es solo un *mensaje* del autor del contenedor para quien vaya a usar el contenedor

Configuración en español

Las imágenes base de las distribuciones son esqueletos mínimos, normalmente tendremos que personalizarlas

Por ejemplo, para configurar el idioma. En nuestro caso, español. Instalaremos en la imagen el paquete `locales`, invocaremos a localedef con los parámetros adecuados y definiremos la variable de entorno LANG

```
FROM ubuntu:22.04
RUN apt update && apt upgrade -y && \
  apt install -y  locales  && \
  localedef -i es_ES -c -f UTF-8  \
  -A /usr/share/locale/locale.alias es_ES.UTF-8
ENV LANG es_ES.UTF-8
COPY entrypoint.sh /
ENTRYPOINT ["/entrypoint.sh"]
```

- La instrucción ENV del Dockerfile define variables de entorno dentro de la imagen

Lo habitual es usar una única instrucción RUN en cada Dockerfile, para evitar las imágenes intermedias.

Pero también podemos usar varias instrucciones, para que resulte más legible.

Ejemplo:

```
FROM ubuntu:22.04
RUN apt update && apt upgrade -y
RUN apt install -y  locales
RUN localedef -i es_ES -c -f UTF-8  \
    -A /usr/share/locale/locale.alias es_ES.UTF-8
ENV LANG es_ES.UTF-8
COPY entrypoint.sh /
ENTRYPOINT ["/entrypoint.sh"]
```

Observa que

- El slash invertido al final de línea (\) une dos líneas físicas en una misma línea lógica
- El doble ampersand (`&&`) separa sentencias dentro de la misma instrucción RUN

Sesiones gráficas

En un contenedor podemos lanzar aplicaciones gráficas

- Si el cliente y el servidor de Docker están en la misma máquina y ambas son Unix, podemos usar *X11 Forwarding*

  ```
  docker run -ti --rm \
         -e DISPLAY=$DISPLAY \
         -v /tmp/.X11-unix:/tmp/.X11-unix \
         mi_imagen
  ```

- Una solución más general es VNC

  Aquí se describe:

  `http://gsyc.urjc.es/~mortuno/vnc.pdf`

8.6. Troubleshooting

Troubleshooting / Resolución de problemas (1)

Si tu contenedor / contenedores no funciona como debe, repasa lo siguiente

- ¿Puedes ejecutar un contenedor *holamundo*?
- ¿El usuario que lanza el contenedor pertenece al grupo *docker*?
- ¿Has probado los scripts fuera del contenedor? ¿Tienen los permisos adecuados?
- ¿Has escrito todas las opciones antes del argumento principal (la imagen) ?

```
docker run -it imagen   # ok
docker run imagen -it   # MAL
```

Troubleshooting / Resolución de problemas (2)

- Si la tarea exige privilegios de root ¿eres root o usas sudo correctamente?
- ¿Tienes claro en qué máquina estás? (máquina física, máquina virtual, contenedor). ¿Tienes claro qué ficheros van en cada cual?
- ¿Tienes claro qué va en el Dockerfile y qué en el entrypoint?
- Cuando copias un fichero desde el directorio contexto hasta la imagen del contenedor ¿lo haces correctamente? Puedes revisarlo de forma interactiva, esto es, desde una shell en el contenedor, comprobar que el fichero está en el lugar adecuado dentro del contenedor

https://www.vagrantup.com

Vagrant

- Es una herramienta para construir y gestionar entornos de máquinas virtuales.
- Creado en 2010, es software libre, muy popular
- Funciona sobre Linux, FreeBSD, macOS, y Microsoft Windows
- Soporta las principales plataformas de virtualización: Docker, VirtualBox, VMware, AWS, Azure, entre otras.

  Vagrant las denomina *providers*
- Su función básica es reemplazar el interfaz (tanto gráfico como de texto) de estas plataformas, proporcionando un interfaz de texto, programable y homogéneo que permite preparar las máquinas, levantarlas, configurarlas, etc.
  - Lo fundamental es ser *homogéneo*, esto es, como administradores podemos configurar y gestionar muchas plataformas distintas, como si fueran la misma
- Usando Vagrant, resulta muy sencillo migrar entre diferentes tecnologías de virtualización
- Para la configuración, se integra con Ansible, Chef y Puppet, entre otras

Uso de Vagrant

Con Vagrant, es muy fácil crear y poner en marcha una máquina virtual, por ejemplo con VirtualBox

- Si no indicamos el *provider*, Vagrant usa VirtualBox. Es necesario haberlo instalado previamente
- No es necesario lanzar el GUI de VirtualBox, pero también podemos usarlo simultáneamente
- Todo lo relativo a una máquina virtual a manejar con vagrant se guarda en un directorio denominado *project directory*

Vagrant Box

- Vagrant cuenta con repositorios de imágenes preconfiguradas. Las denomina *boxes*. Hay *boxes* oficiales, y también cualquier usuario puede preparar sus boxes y hacerlos públicas gratuitamente
  - Se pueden preparar boxes privados, estos son de pago

Puesta en marcha de un Box

1. Creamos en nuestro *host* el *project directory*
2. Accedemos al *project directory*
3. Ejecutamos `vagrant init <NOMBRE_DE_BOX>`

   p.e.

   `vagrant init ubuntu/jammy64`
4. Encendemos la máquina

   `vagrant up`
5. Entramos en la máquina

   `vagrant ssh`

   De esta forma tenemos una sesión con el usuario *vagrant*, preconfigurado para poder lanzar procesos como root sin escribir contraseña

Observa que nunca indicamos con qué máquina queremos trabajar, basta con lanzar la orden `vagrant` desde el *project directory* que necesitemos en cada momento

- Vagrant redirecciona automáticamente un puerto del *host* al puerto 22 del *guest* para poder hacer ssh

  Si está libre, el 2222. Si no, usará otro. Lo indicará en el arranque de la máquina
  - Podemos entrar en el *guest* ejecutando en el *host* la orden

    ```
    ssh -p 2222 usuario@localhost
    ```
  - Pero por omisión no podemos autenticarnos mediante contraseña, sino mediante *authorized keys*
- Vagrant monta automáticamente el *project directory* del *host* en el directorio `/vagrant` del *guest*

Parada de una máquina

Tres formas distintas:

- `vagrant suspend`

  Duerme la máquina

- `vagrant halt`

  Para la máquina

- `vagrant destroy`

  Para la máquina, borra su imagen y todos sus ficheros

Naturalmente, estas instrucciones debemos ejecutarlas desde la máquina donde está vagrant, esto es, el *host*, no el *guest*

Vagrantfile

- La orden `vagrant init` crea automáticamente en el *project directory* un fichero `Vagrantfile`, que es el fichero de configuración de la máquina virtual

- Las opciones de configuración se escriben entre las líneas

  `Vagrant.configure("2") do |config|`

  y

  `end`

Cambiar el nombre de la máquina virtual (el nombre que usa el *provider*

- `config.vm.hostname= "MI_MAQUINA"`

Cambiar el nombre de host:

- `config.vm.define "MI_MAQUINA"  # sin '='`

Redireccionamiento de un puerto del *host* al *guest* al

- `config.vm.network "forwarded_port", guest: 80, host: 8080, host_ip: "127.0.0.1"`

Los boxes preconfigurados suelen tener un usuario *vagrant*, su claves se guardan en el *project directory*, en

`.vagrant/machines/default/virtualbox/private_key`

Provisionamiento de la máquina

- Se denomina *provisionar* la máquina a configurar de forma automática los ajustes que necesitemos en la máquina virtual
- La forma más sencilla es con scripts de shell, en entornos más exigentes podemos usar puppet, ansible, etc

En el *project directory* de Vagrant, en el fichero *Vagrantfile*, al final, encontraremos estas líneas, comentadas:

```
# config.vm.provision "shell", inline: <<-SHELL
#   apt-get update
#   apt-get install -y apache2
# SHELL
```

Para provisionar la máquina con comandos de shell

1. Descomentamos esas líneas
2. Reemplazamos las órdenes entre `<<-SHELL` y `SHELL` por las que necesitemos: instalación de paquetes, creación de grupos, usuarios, etc

Estos comandos se ejecutarán cada vez que preparemos una máquina con *vagrant up*

- Si creamos un usuario con `adduser`, fallará la lectura de la contraseña desde la entrada estándar
- Podemos ignorar este error, posteriormente el *root* podrá abrir una sesión de este usuario o cambiarle la contraseña
- Recuerda que también podemos poner la contraseña con las órdenes

  `echo "jperez:sesamo" | chpassw`

  aunque este enfoque no es muy seguro, porque la contraseña quedaría visible en el fichero de provisionamiento

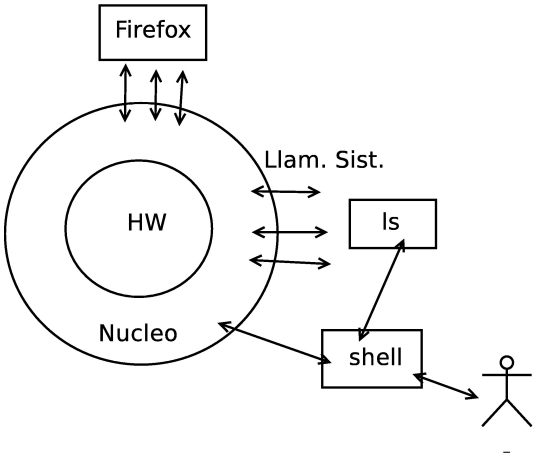


Figura 2: El Sistema Operativo

9. Shell:Intérprete de órdenes

- La shell más habitual es *bash*, pero hay muchas otras *sh*, *csh*, *dash*
- Las **órdenes** generalmente son solo pequeños programas ejecutables
- El nombre original es *shell command*. En español puede decirse *comando*, *orden* o *mandato*.

9.1. ¿Quién soy? ¿Dónde estoy? ¿Qué tengo?

¿Quién soy? ¿Dónde estoy? ¿Qué tengo?

- `whoami`

  Muestra el usuario

- `id`

  Muestra usuario y grupos

- `uname`
  `uname -a`

  Versión de Linux

- `hostname`

  Nombre de máquina

- `pwd`

  Directorio de trabajo actual

- `w`

  Usuarios conectados a la máquina

- `du` Espacio de disco ocupado por los ficheros de un directorio
  `du -s` Espacio de disco ocupado por un directorio
  `du -h` Unidades legibles para un humano

- `ncdu` Versión de du interactiva, mejorada

- `df`

  Espacio de disco libre

- `lsblk -f`

  Listado de todos los discos (dispositivos de bloques)

- `ls -l` Formato largo
  `ls -a` Muestra ficheros ocultos (empiezan por punto)
  `ls -lh` Formato largo, unidades legibles por humano
  `ls -R` Recursivo
  `ls -ld` Lista el directorio, no su contenido

Unix es *case sensitive*

9.2. Metacaracteres de la Shell

Metacaracteres de la Shell

- `$` Variable
- `*` 0 o más caracteres cualquiera
- `?` exactamente 1 carácter cualquiera
- `[]` 1 carácter de la clase

ejemplo:
`ls *.txt`
el shell lo expande a
`ls  texto1.txt texto2.txt texto3.txt`
La orden recibe 3 argumentos, no sabe nada de metacaracteres

9.3. Funcionamiento de la shell

Funcionamiento (simplificado) de la shell

La shell:

1. Lee texto de fichero stdin (por ejemplo, el teclado). Aporta algunas facilidades al usuario (borrar, autocompletar)
2. Analiza el texto (expande metacaracteres y variables)
3. Toma la primera palabra y busca una orden con ese nombre en los directorios indicados por PATH
4. Si puede, ejecuta la orden y se queda dormida esperando a que acabe
   Por ejemplo

   ```
   koji@mazinger:~$ xcalc
   ```

   (Mientras usamos la calculadora, la shell permanece inactiva)

   - Si queremos que la shell siga activa, lanzamos el proceso en segundo plano (*background*)

     ```
     koji@mazinger:~$ xcalc&
     ```

- Una aplicación lanzada sin `&`, se dice que está lanzada en primer plano (*foreground*).
- La shell se cierra con la orden `exit`. (O con `ctrl d`, que representa el fin de fichero)

Autocompletado

Con frecuencia pasaremos a los mandatos nombres de fichero (como argumento). La función de autocompletar evita teclear nombres completos

Supongamos que tenemos dos ficheros en el directorio actual

```
.
|-- mi_fichero_del_martes
`-- un_fichero_ejemplo
```

No es necesario teclear

```
koji@mazinger:~$ ls -l mi_fichero_del_martes
```

Como solo hay un fichero que empiece por *mi*, basta escribir

```
koji@mazinger:~$ ls -l mi
```

y luego pulsar tab

Si hay más de un fichero que empiece por *mi*

```
.
|-- mi_fichero_del_martes
|-- mi_fichero_del_miercoles
`-- un_fichero_ejemplo

koji@mazinger:~$ ls -l mi_fichero_del_m
mi_fichero_del_martes     mi_fichero_del_miercoles
```

Autocompletar rellena hasta donde puede, nos ofrece los ficheros que encajan en lo que hemos escrito, y espera a que introduzcamos una letra más para deshacer la ambigüedad (en este ejemplo, `'a'` o `'i'`)

La shell también autocompleta nombres de ejecutables (si tienen permiso de ejecución y están en el path)

```
koji@mazinger:~$ pass<TAB>
```

Se autocompleta a

```
koji@mazinger:~$ passwd
```

De esta manera no hace falta teclear todas las letras. Ni recordar el nombre exacto de órdenes largas, basta saber cómo empiezan

history

La shell recuerda las últimas órdenes ejecutadas. Podemos desplazarnos sobre ellas con los cursores arriba/abajo

9.4. Variables

Variables

- `variable=valor`
  `echo $variable`
  Sin espacios antes y despues del igual
  con $ para acceder al contenido de la variable
  sin $ en la asignación
  sólo son visibles en ese proceso

```
nombre=juan
echo $nombre
```

9.4.1. Variables de entorno

Variables de entorno

- `export VARIABLE=valor`
  hace que los procesos hijos del proceso donde se declara la variable, la reciban. Por convenio se usan mayúsculas
- Para que el cambio sea permanente, hay que exportar la variable en algún fichero de configuración como p.e. `.bashrc`
- `printenv`
  muestra todas las variables de entorno
- HOME
- HOSTNAME
- USER
- PATH
  Contiene la lista de directorios donde la shell buscará los ejecutables (si no se indica path explícito)

La variable de entorno HOME

- Indica el directorio *hogar* de un usuario: el sitio donde se espera que cada usuario escriba sus cosas

```
koji@mazinger:~$ echo $HOME
/home/koji
```

- Se le suele llamar `$HOME`, pero esto no es muy preciso
  - La variable se llama `HOME`, el dólar se antepone a todas las variables en bash cuando se están referenciando (y no cuando se asignan)
  - Es un error frecuente intentar usar `$HOME` en otros lenguajes o en cualquier programa. Solo es válido en bash y shells similares

Virgulilla

La virgulilla (~) representa el directorio *home* de un usuario

- Equivale a `$HOME`, con la ventaja de que se puede usar en muchos lenguajes, aplicaciones y librerías (no todos)
- No aparece en los teclados, pero está accesible en `AltGr 4`
- Seguida de un nombre de usuario, representa el *HOME* de ese usuario

```
koji@mazinger:~$ echo ~jperez
/home/jperez
```

Si el nombre del usuario no es una cadena literal sino una variable es necesario volver a evaluar la expresión

```
koji@mazinger:~$ nombre=koji
koji@mazinger:~$ echo ~$nombre
~koji
koji@doublas:~$ eval echo ~$nombre
/home/koji
```

La variable de entorno PATH

Un usuario principiante ejecuta

```
koji@mazinger:~/pruebas$ ls -l
total 4
-rw-r--r-- 1 koji koji 27 2009-10-07 19:02 holamundo
```

Intenta invocar el mandato *holamundo* escribiendo

```
koji@mazinger:~/pruebas$ holamundo
```

pero obtiene

```
bash: holamundo: orden no encontrada
```

Problema 1

El fichero no tenía permisos de ejecución

Problema 1: Solución

```
koji@mazinger:~/pruebas$ chmod ugo+x holamundo
```

¿Problema resuelto?

```
koji@mazinger:~/pruebas$ ls -l
total 4
-rwxr-xr-x 1 koji koji 27 2009-10-07 19:02 holamundo
```

No ha bastado. El usuario vuelve a ejecutar

```
koji@mazinger:~/pruebas$ holamundo
```

pero vuelve a obtener

```
bash: holamundo: orden no encontrada
```

Problema 2

Aunque el fichero está en el directorio actual (directorio *punto*), la shell no lo buscará allí, sino donde indique la variable de entorno PATH, que contiene una lista de directorios, separados por el carácter *dos puntos*

```
koji@mazinger:~/pruebas$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Lo buscará en `/usr/local/sbin`

Si no lo encuentra, lo buscará en `/usr/local/bin`

Si sigue sin encontrarlo, lo buscará en `/usr/local/sbin`

etc

Pero no lo buscará en el directorio *punto*

Problema 2: Solución 1 (recomendada)

Invocar el mandato indicando explícitamente que el fichero está en el directorio *punto*

```
koji@mazinger:~/pruebas$ ./holamundo
¡hola mundo!
```

Problema 2: Solución 2

Indicar el trayecto absoluto del mandato

```
koji@mazinger:~/pruebas$ /home/koji/pruebas/holamundo
¡hola mundo!
```

Problema 2: Solución 3

Modificamos la variable de entorno PATH para añadir al final el directorio *punto*

Como queremos que el cambio sea permanente, debemos modificar la variable en un fichero de configuración [4] , por ejemplo `~/.bashrc`

```
export PATH=$PATH:.
```

[4] Más detalles en el apartado *invocación de la shell*

El cambio no se produce de inmediato, sino cuando se ejecute de nuevo `~/.bashrc`

- Al invocarlo explícitamente

```
koji@mazinger:~/pruebas$ source ~/.bashrc
```

- Al abrir una nueva terminal

Problema 2: Solución 4 ¡Muy peligrosa!

Modificamos la variable de entorno PATH para añadir al principio el directorio *punto*

`export PATH=.:$PATH`

Supongamos que un atacante escribe un script con el nombre `ls` y el contenido

```
#!/bin/bash
rm -rf  $HOME
```

Al escribir la orden `ls`, se ejecutaría este script, y no `/bin/ls`

9.5. Ficheros

9.5.1. Árbol de directorios

Árbol de directorios

```
/
├── bin
├── etc
├── home
│   ├── alcorcon
│   │   └── juan
│   │       ├── d1
│   │       │   ├── f1
│   │       │   └── f3
│   │       ├── d2
│   │       └── f1
│   └── fuenla
│       ├── luis
│       └── marta
└── var
```

- Árbol, todo cuelga de un único directorio raiz
- Dentro de cada directorio, habrá ficheros o subdirectorios
- jerarquía clásica unix:
  - /bin
  - /etc
  - /home
  - /var
  - (...)

Nombres de fichero

- Hasta 256 caracteres
- Mayúsculas y minúsculas son distintas
  - Se puede tener en un mismo directorio los ficheros `ejemplo`, `EJEMPLO` y `EjemPlO`
  - Pero si llevamos estos ficheros a una unidad externa (pendrive, disco) que mantenga su formato por omisión (FAT32), deja de ser legal
- Los que empiezan por punto (`.`)se consideran ocultos (por defecto no se muestran), suelen usarse para ficheros o directorios de configuración
- Casi cualquier carácter es legal, pero es preferible usar solo números, letras, guión y barra baja.
  - Es preferible evitar los espacios
  - También es buena idea evitar eñes y tildes (Naturalmente, hablamos del nombre del fichero, no de su contenido)

9.5.2. Permisos

Permisos

ls -l: Muestra los contenidos de los directorios en **formato largo**:

```
drwxr-xr-x 2 jperez al-07-08 4096 2007-10-09 22:51 d1
-rw-r--r-- 1 jperez al-07-08 8152 2007-10-16 09:42 f1
-rw-r--r-- 1 jperez al-07-08   24 2007-10-16 09:42 f3
```

El primer carácter indica:

```
-       Regular file - Fichero ordinario
d       Directory - Directorio
l       (Symbolic) Link - Enlace simbólico
p       Named pipe - Pipe con nombre
s       Socket - Socket
c       Character device - Dispositivo orientado a carácter
b       Block device - Dispositivo orientado a bloque
```

Para cada entrada, aparece, además:

- permisos: Los 9 primeros caracteres

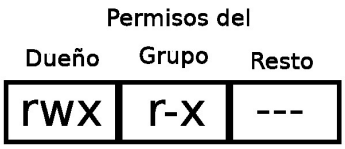
- número de nombres del fichero (enlaces duros)
- usuario del dueño
- grupo del dueño
- tamaño en bytes
- fecha y hora de la última modificación
- nombre

En español habitualmente usamos la palabra *permisos* para referirnos al *access mode*, cuya traducción literal sería *modo de acceso*

Los permisos se indican en una secuencia de caracteres como p.e.

```
rwxr-x---
```

- Los primeros tres caracteres representan los permisos para el dueño del fichero
- Los siguientes tres caracteres, los permisos de los usuarios del mismo grupo que el fichero
- Los últimos tres caracteres, los permismos del resto de usuarios



En cada grupo de tres caracteres

- La letra `r` en la primera posición significa permiso de lectura concedido
- La letra `w` en la segunda posición significa permiso de escritura concedido
- La letra `x` en la tercera posición significa permiso de ejecución concedido
- Si en vez de la letra aparece un guión, el permiso está denegado

Los permisos se suelen indicar también como un dígito decimal[5]

- Si el permiso está concedido, se considera un dígito 1 binario
- En otro caso, un dígito 0 binario
- Estos tres dígitos binarios (rwx), se expresan en decimal

```
Permisos    Binario    Decimal
---         000        0
--x         001        1
-w-         010        2
-wx         011        3
r--         100        4
r-x         101        5
rw-         110        6
rwx         111        7
```

- Permisos de un fichero:
  - El de **lectura**: permite ver su contenido
  - El de **escritura**: permite modificar su contenido
  - El de **ejecución**: permite ejecutarlo
- Permisos de un directorio:
  - El de **lectura**: permite hacer `ls` del contenido
  - El de **escritura**: permite crear y borrar ficheros y subdirectorios dentro de él
  - El de **ejecución**: permite hacer `cd` a él

Para cambiar permisos se usa `chmod`, que tiene dos sintaxis equivalentes, se puede usar la que resulte más cómoda

1. `chmod  754 mi_fichero`

   No importan los permisos que tuviera previamente el fichero, pasa a tener:

```
  7   5   4        (decimal)
111 101 100        (binario)
rwx r-x r--
```

[5]en rigor, octal

2. `chmod [ugo] [+-] [rwx] mi_fichero`
   `chmod o+x mi_fichero`

   A partir de los permisos que tuviera el fichero, se suman o se restan los permisos indicados a u,g,o (user, group, other)

Permisos de los directorios

`chmod -R` Cambia permisos recursivamente

- `r` y `x` normalmente van juntos. (Ambos o ninguno).

  Permiten entrar en el directorio y listar

- `w` permite añadir añadir ficheros o borrarlos

Muy Importante:
Comprueba los permisos de tu `HOME`, en muchos sistemas por omisión está abierto

Atención,
un fichero sin permisos de escritura, p.e. `rwxr-xr-x`
pero con permiso de escritura en el directorio que lo contiene, `rwxrwxrwx`
no podrá ser modificado pero sí borrado o renombrado

9.5.3. path

Directorios Especiales

- Todo directorio contiene dos subdirectorios especiales:

`.`	El subdirectorio `.` de un directorio es él mismo
`..`	El subdirectorio `..` de un directorio es su directorio padre

```
/
├── bin
├── etc
├── home
│   ├── alcorcon
│   │   └── juan
│   │       ├── d1
│   │       │   ├── f1
│   │       │   └── f3
│   │       ├── d2
│   │       └── f1
│   └── fuenla
│       ├── luis
│       └── marta
└── var
```

- Ejemplos:
  - El subdirectorio `.` de `juan` es `juan`
  - El subdirectorio `..` de `luis` es `fuenla`
  - El subdirectorio `..` de `home` es `/`

9.6. Operaciones básicas con ficheros y directorios

touch

Cambia la fecha a un fichero, o lo crea si no existe

`touch <fichero>`

- Si `<fichero>` existe, le pone la fecha actual
- Si `<fichero>` no existe, crea un fichero vacío con este nombre

`touch -d <fecha/hora>  <fichero>`

Modifica la fecha de último acceso al fichero

```
touch -d 2007-02-28 fichero      # cambia la fecha
touch -d 15:41 fichero           # cambia la fecha
```

mkdir: Creación de directorios

mkdir: Crea directorios (*make directory*)

`mkdir <fichero>`

- `mkdir d3`

  Crea d3 como subdirectorio del directorio actual

- `mkdir d4 d5`

  Crea d4 y d5 como subdirectorios del directorio de trabajo actual

- `mkdir /tmp/ppp`

  Crea el directorio `/tmp/ppp`

- `mkdir -p d6/d7`

  Crea debajo de directorio de trabajo `d6` (si no existe), y crea `d7` debajo de `d6`

Copiar, mover y renombrar

- La órden `cp` copia ficheros
- La órden `mv` mueve y renombra ficheros

En primer lugar mostraremos el uso básico, después las opciones completas

Copiar un fichero:

tengo

```
/tmp/probando/quijote.txt
```

quiero

```
/tmp/probando/quijote.txt
/tmp/probando/quijote_repetido.txt
```

hago

```
cd /tmp/probando
cp quijote.txt quijote_repetido.txt
```

Renombrar un fichero:
tengo

```
/tmp/probando/quijote.txt
```

quiero

```
/tmp/probando/don_quijote.txt
```

hago

```
cd /tmp/probando
mv quijote.txt don_quijote.txt
```

Copiar un fichero en un directorio distinto

```
tengo
/tmp/probando/quijote.txt
```

quiero

```
/tmp/probando/quijote.txt
/tmp/otro_probando/quijote.txt
```

voy al directorio destino

```
cd /tmp/otro_probando/
```

```
#copio     "el fichero"         "aquí"
cp    /tmp/probando/quijote.txt    .
```

Mover un fichero a un directorio distinto
tengo

```
/tmp/probando/quijote.txt
```

quiero

```
/tmp/otro_probando/quijote.txt
```

voy al destino

```
cd /tmp/otro_probando/
```

```
# muevo "el fichero"        "aquí"
mv /tmp/probando/quijote.txt   .
```

cp: Copiar 1 fichero ordinario

`cp <origen> <destino>`

cp (*copy*) con dos argumentos. `<origen>` es un fichero ordinario

- Si el segundo argumento es un directorio

  Hace una copia del fichero `<origen>` dentro del directorio `<destino>`

- Si el segundo argumento NO es un directorio (es un fichero o no existe nada con ese nombre)

  Hace una copia del fichero `<origen>` y le pone como nombre `<destino>`

Como siempre, tanto `<origen>` como `<destino>` pueden indicarse con trayecto relativo o con trayecto absoluto

Ejemplos:

```
cp holamundo.py /tmp
cp ~/prueba.txt .
cp /home/jperez/prueba.txt prueba2.txt
```

cp: Copiar 1 directorio

`cp -r <origen> <destino>`

Si `<origen>` es un directorio, es necesario añadir la opción `-r` (*recursive*)

- Si `<destino>` es un fichero ordinario, se produce un error
- Si `<destino>` es un directorio, el directorio `<origen>` se copia dentro
- Si `<destino>` no existe, se le pone ese nombre a la copia

Ejemplos

```
cp -r ~  /tmp
cp -r /var/tmp/aa .
cp -r ~  /tmp/copia_de_mi_home
```

cp: Copiar varios ficheros ordinarios

`cp <origen1> <origen2> .... <destino>`

cp (*copy*) con varios argumentos. Los ficheros `<origen1> <origen2> ....` se copian en el directorio `<destino>`

- `<destino>` tiene que ser un directorio (o se producirá un error)
- `<origen1>, <origen2>,...` tienen que ser ficheros ordinarios (o un mensaje indicará que no se están copiando)

Ejemplos:

```
cp holamundo.py /home/jperez/prueba1.txt ../prueba2.txt /tmp
cp bin/*.py /tmp
```

cp: Copiar varios ficheros o directorios

`cp -r <origen1> <origen2> .... <destino>`

Este caso es idéntico al anterior, solo que si `<origen1>` o `<origen2>` o `...` son directorios, es necesaria la opción `-r`

Ejemplos:

```
cp -r holamundo.py /home/jperez /tmp
```

mv: mover o renombrar ficheros y directorios

`mv <origen> <destino>`

Mover dentro del mismo directorio equivale a renombrar

`<origen>` es un fichero o un directorio

- Si el segundo argumento es un directorio

  Mueve `<origen>` dentro del directorio `<destino>`

- Si el segundo argumento no existe

  Mueve `<origen>` a `<destino>`

- Si `<destino>` es un fichero

  - y `<origen>` es un fichero, `<origen>` pasa a llamarse `<destino>` y el anterior `<destino>` despararece
  - y el primero es un directorio, se produce un error

Ejemplos:

```
mv holamundo.py /tmp
mv ~/prueba.txt .
mv /home/jperez/prueba.txt prueba2.txt
```

mv con más de dos argumentos

`mv <origen1> <origen2> ...  <destino>`

`<destino>` debe ser un directorio existente

`<origen1>, <origen2>...` pueden ser ficheros ordinarios o directorios

Ejemplos:

```
mv holamundo.py /home/jperez/prueba1.txt ../prueba2.txt /tmp
mv *.txt texto
```

Tipos de fichero

- Tradicionalmente en Unix los ficheros no llevaban extensión

  No hay un programa asociado a cada extension

  `file mifichero`

  Indica el tipo del fichero. No importa si tiene extensión, si no la tiene, o si es errónea

Supongamos que tenemos un fichero y no sabemos con qué programa podemos abrirlo. P.e. desconocemos que tenemos instalado evince para abrir ficheros pdf

- En Linux
  - Si nuestro escritorio es gnome, podemos ejecutar
    `gnome-open  fichero.extension`
  - Si usamos KDE, `kde-open fichero.extension`
  - Para gnome, KDE y muchos otros `xdg-open fichero.extension`
- En Mac OS
  `open  fichero.extension`

Borrado de un fichero

- `rm` *fichero*
  borra fichero [6]
  `rm  -r` *directorio*
  Borra un directorio y todo su contenido

Un usuario de MS-DOS podría intentar hacer
`mv *.txt *.doc  # ¡MAL!  No funciona, y puede ser fatal`
Supongamos que tenemos en el directorio actual

```
carta1.txt
carta2.doc
```

Tras expandir los asteriscos, el resultado es
`mv carta1.txt carta2.doc  # destruimos el segundo fichero!`
Una solución posible [7]:

```
#!/bin/bash
for fichero in *.txt
do
    nombre=$(echo $fichero | cut -d. -f1)
    extension=$(echo $fichero | cut -d. -f2)
    mv $fichero $nombre.doc
done
```

[6] Cuando hablemos de enlaces veremos una definición más exacta
[7] Siempre que solo haya un punto en el nombre

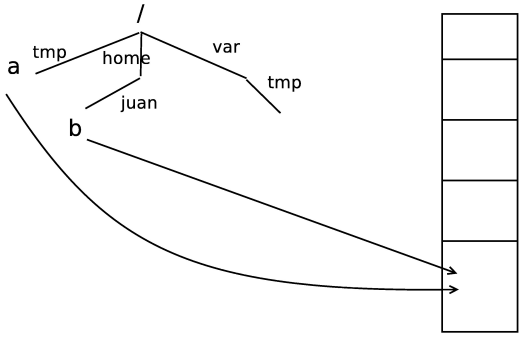


Figura 3: Enlace Duro

9.7. Enlaces

Enlace duro

Un nuevo nombre para el fichero

`ln a b`

- Ambos nombres deben pertenecer al mismo sistema de ficheros
- Dado un fichero, se sabe cuántos nombres tiene. Para saber cuáles son sus nombres, habría que buscarlos
- La mayoría de los S.O. no permiten enlaces duros a directorios, puesto que podría provocar bucles difíciles de detectar

`rm` borra un nombre de un fichero
si es el último, borra el fichero.

Enlace blando o simbólico

Un nuevo fichero que apunta a un nombre

`ln -s /home/juan/b c`

- Sirven principalmente para mantener ficheros ordenados y *a mano*
- Puede hacerse entre distintos sistemas de ficheros
- Puede enlazarse un directorio
- Con enlaces simbólicos, si se borra el original
  el enlace queda roto

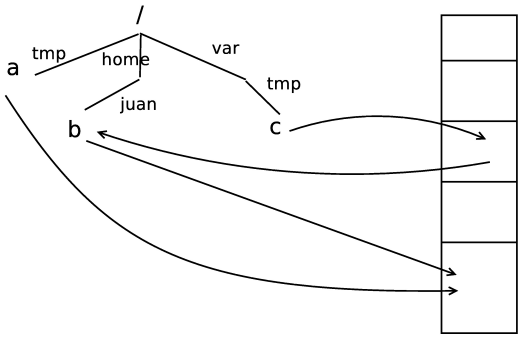


Figura 4: Enlace Simbólico

- El fichero original podemos especificarlo
  - Con su path absoluto
  - Con su path relativo

    En este caso, si movemos el enlace simbólico pero no movemos el original, se pierde la referencia

Utilidad de los enlaces

Tanto los *blandos* como los *duros*
son útiles:

- Para tener acceso a un fichero en un trayecto más *cómodo*, más *a mano*
- Si cambio de criterio sobre el lugar o el nombre de un fichero. Mediante un enlace, el fichero sigue accesible tanto por el nombre antiguo como por el nuevo

Ventaja de los enlaces duros:

- Protegen frente a borrados accidentales de un nombre. Pero no frente a ningún otro problema que pueda tener el fichero, por tanto su utilidad es mínima

Ventaja de los enlaces simbólicos:

- Se pueden establecer entre sistema de ficheros distintos
- Se pueden usar para directorios

Los enlaces simbólicos se usan mucho más que los enlaces duros

Directorio de Trabajo

- La shell en todo momento se encuentra en un cierto punto del árbol de ficheros. A ese punto se le llama **directorio de trabajo** (*working directory*)
- Normalmente la shell indica el directorio de trabajo en el *prompt*
- pwd: Muestra el directorio de trabajo actual (*print working directory*)

  ```
  pwd
  ```

Trayectos (Paths)

- Un trayecto (path) consiste en escribir el camino hasta un fichero o directorio, incluyendo directorios intermedios separados por el carácter `/`
- Trayecto absoluto:
  - Escribe el camino desde el **directorio raíz**
  - **Siempre** empieza por `/`
- Trayecto relativo:
  - Escribe el camino desde el directorio de trabajo
  - **Nunca** empieza por `/`
- Cualquier programa acepta (o debería aceptar) que cuando se especifica un nombre de fichero, se use o bien la forma relativa o bien la forma absoluta.

  Esto es aplicable a casi cualquier programa de casi cualquier Sistema Operativo

¿Un trayecto con virgulilla es relativo o absoluto?
`~/mi_directorio`
En cierta forma es relativo

- No empieza por `/`
- Depende del usuario que lo ejecuta

En cierta forma es absoluto

- No depende del directorio de trabajo
- Lo que sucede realmente es que se reemplaza la virgulilla por el trayecto absoluto del *home* del usuario

Posiblemente lo más adecuado es considerarlo un caso un poco especial de path absoluto

```
/
├── bin
├── etc
├── home
│   ├── alcorcon
│   │   └── juan
│   │       ├── d1
│   │       │   ├── f1
│   │       │   └── f3
│   │       ├── d2
│   │       └── f1
│   └── fuenla
│       ├── luis
│       └── marta
└── var
```

- Ejemplos:
  - Trayecto absoluto de `f3`:

    `/home/alcorcon/juan/d1/f3`
  - Trayecto relativo de `f3` si el directorio de trabajo es `juan`:

    `d1/f3`
  - Trayecto relativo de `f3` si el directorio de trabajo es `d2`:

    `../d1/f3`
  - Trayecto relativo de `var` si el directorio de trabajo es `luis`:

    `../../../var`

- cd: Cambia el directorio de trabajo (*change directory*)

`cd d1`	Cambia desde el directorio de trabajo actual a su subdirectorio `d1`
`cd /home`	Cambia desde cualquier directorio al directorio `/home`
`cd ..`	Cambia desde el directorio de trabajo actual a su directorio padre (sube un directorio)
`cd`	Cambia al directorio por defecto (hogar) del usuario

- ls: Muestra los contenidos de un directorio (*list*)

`ls`	Muestra el contenido del directorio de trabajo
`ls d1`	Muestra el contenido del subdirectorio `d1`
`ls /home`	Muestra el contenido de `/home`

9.8. Comandos de uso básico de la red

Mandatos de uso básico de la red

ping: Comprueba si una máquina responde en la red

`ping gsyc.es` Sondea la máquina `gsyc.es` indefinidamente mostrando el doble de la latencia con ella. `CTRL-c` para terminar y mostrar un resumen

`ping -c 4 gsyc.es` Sondea la máquina `gsyc.es` 4 veces

traceroute: Muestra encaminadores intermedios hasta un destino

`traceroute gsyc.es` Muestra encaminadores intermedios desde la máquina en la que se está hasta `gsyc.es`. Muestra el doble de las latencias hasta cada punto intermedio.

```
traceroute to gsyc (193.147.71.64), 30 hops max, 60 byte packets
 1  ap (192.168.1.1)  0.730 ms  1.376 ms  1.345 ms
 2  10.213.0.1 (10.213.0.1)  9.927 ms  15.040 ms  15.029 ms
 3  10.127.46.153 (10.127.46.153)  15.003 ms  15.632 ms  15.607 ms
 4  mad-b1-link.telia.net (213.248.90.85)  28.549 ms  28.720 ms  28.691 ms
 5  dante-ic-125710-mad-b1.c.telia.net (213.248.81.26)  28.822 ms  28.959 ms  35.580 ms
 6  nac.xe0-1-0.eb-madrid0.red.rediris.es (130.206.250.22)  36.344 ms  35.077 ms  34.967 ms
 7  cam-router.red.rediris.es (130.206.215.66)  34.940 ms  12.015 ms  12.689 ms
 8  * * *
 9  gsyc.escet.urjc.es (193.147.71.64)  14.675 ms  14.934 ms  15.500 ms
```

ssh

Ejecuta mandatos de shell en una máquina remota

`ssh jperez@zeta12.pantuflo.es`
Se conecta a la máquina `zeta12.pantuflo.es` (pide password) y permite ejecutar mandatos en ella. Toda la sesión entre la máquina origen y destino viaja cifrada por la red

`ssh jperez@zeta12.pantuflo.es ls /`
Se conecta a la máquina `zeta12.pantuflo.es` (pide login y password), ejecuta el mandato `ls /` y sale de ella.

- La primera vez que abrimos una sesión en una máquina, ssh nos indica la huella digital de la máquina remota

```
The authenticity of host 'gamma23 (212.128.4.133)' can't be established.
RSA key fingerprint is de:fa:e1:02:dc:12:8d:ab:a8:79:8e:8f:c9:7d:99:eb.
Are you sure you want to continue connecting (yes/no)?
```

- Si necesitamos la certeza absoluta de que esta máquina es quien dice ser, deberíamos comprobar esta huella digital por un medio seguro, alternativo
- La sesión se cierra cerrando la shell remota (`exit` o `ctrl d`)

scp

```
scp [[loginname@]maquina:]<origen> [[loginname@]maquina:]<destino>
```

Copia ficheros desde/hacia máquinas remotas. El contenido de los ficheros viaja cifrado por la red.

Igual que `cp`, pero ahora hay que añadir o bien a `origen` o bien a `destino`

- ¿Cuál es la máquina remota?
- ¿Qué nombre de usuario tenemos en la máquina remota?

  `usuario@maquina:`

- En caso de que el nombre de usuario en la máquina local sea el mismo que en la máquina remota, puede omitirse `usuario@`
- Los dos puntos del final nunca pueden omitirse
- No puede haber espacios después de los dos puntos
- La máquina se puede indicar por su nombre o por su dirección IP
- Naturalmente, origen y destino pueden indicarse con trayecto relativo o con trayecto absoluto
  - En la máquina remota, los trayectos relativos parten del *home* del usuario remoto

Ejemplos:

`scp f1 jperez@alpha.aulas.gsyc.urjc.es:d1/f1`
: Lleva una copia del fichero `f1` desde la máquina local hasta la máquina `alpha`, entrando como usuario `jperez`, con trayecto `~jperez/d1/f1`

`scp f1 jperez@alpha.aulas.gsyc.urjc.es:`
: Lleva una copia del fichero `f1` desde la máquina local hasta la máquina `alpha` , entrando como usuario `jperez`, con trayecto `~jperez/f1`

`scp jperez@alpha.aulas.gsyc.urjc.es:f1 .`
: Trae desde la máquina `alpha`, entrando con el usuario `jperez`, el fichero `~jperez/f1` hasta el directorio de trabajo de la máquina local

Recuerda:

`~jperez`	*home* de jperez
`~/dir1`	subdirectorio `dir1` dentro de mi *home*

Si scp resulta nuevo para tí y no quieres equivocarte, puedes seguir estos pasos:

1. Ten dos sesiones abiertas, una la máquina origen y otra en la máquina destino
2. Mediante `cd`, vete al directorio origen en la máquina origen y haz `pwd` para asegurarte de que estás donde debes
3. Mediante `cd`, vete al directorio destino en la máquina destino y haz `pwd` para asegurarte de que estás donde debes
4. En la máquina origen, haz `ls` del fichero, indicando el path de forma absoluta. El `pwd` anterior te ayudará. Si te equivocas, te darás cuenta ahora

   ```
   ls /path/absoluto/al/fichero.txt
   ```

5. Ejecuta el `scp` en la máquina destino. Especifica el origen con la ayuda de un copia-y-pega del paso anterior. Especifica el destino con '.'

   ```
   scp usuario@maquina:/path/absoluto/al/fichero.txt .
   ```

9.9. Entrada y salida

Entrada y salida

Todo proceso en Unix / Linux tiene:

- Una entrada estándar (*stdin*), de donde puede leer texto
- Una salida estandar (*stdout*), donde puede escribir sus resultados, en modo texto
- Una salida de error (*stderr*), donde puede describir los errores que se produzcan, en modo texto

Inicialmente la entrada se toma desde la *consola* (el teclado) y las salidas también se escriben en la consola (la pantalla). Pero estas entradas y salidas se pueden cambiar con las *redirecciones*

Paso de argumentos a órdenes

Muchas órdenes se comportan así (no todas)

- Sin argumentos: Entrada estándar
  `wc`
- 1 argumento: Nombre de fichero
  `wc fichero`
- n nombres de fichero
  `wc fichero1 fichero2`
- `cat`
  lee lo que hay en stdin y lo escribe en stdout
  (Ctrl D: fin de fichero)
- `cat fichero1 fichero2`
  lee los ficheros que se pasan como argumento y los escribe (concatenados) en stdout
  (Ctrl D: fin de fichero)
- `echo argumento`
  escribe en stdout el texto que se le pasa como argumento. Añade retorno de carro
- `echo -n argumento`
  escribe en stdout el texto que se le pasa como argumento
- `less fichero`
  escribe un fichero en stdout, permitiendo paginación

Redirecciones

```
<    redirige stdin desde fichero
>    redirige stdout a fichero, reemplazando
>>   redirige stdout a fichero, añadiendo
|    redirige stdout de un proceso a stdin del siguiente
```

- `cat`
- `cat file1 file2 > file3`
  `cat file1 | less`
  `cat > file1`

- `less fichero`
  `cat fichero | less`
  `less < fichero`
  (El resultado es el mismo, pero es importante distinguirlo)

1 representa stdout
2 representa stderr

- `mkdir /a/b/c 2> mi_fichero_errores`

  Redirige stderr al fichero

- `cp fichero_a fichero_b 2>/dev/null`

  Redirige stderr al fichero *sumidero* (Lo que se copia en `/dev/null` desaparece sin mostrarse)

Para escribir en 1 o en 2, es necesario anteponer `&` (para que no se confunda con un fichero que se llame `"1"` o `"2"`)

- `echo "ERROR: xxxx ha fallado"  >&2`

  Redirige el mensaje a stderr

`&` representa stdout y stderr

- `find /var  &>mi_fichero`

sudo y redirecciones

La orden *sudo* por omisión no incluye las posibles redirecciones

- `sudo echo hola > /tmp/aa`

  El proceso *echo* se lanza con la identidad del root (id 0), pero la redirección la ejecuta el usuario ordinario

- Para poder usar redirecciones, ejecutamos una subshell con el parámetro -c

  `sudo bash -c  "echo hola>aa"`

  `sudo bash -c  "find /root | grep prueba "`

9.10. Programación de Scripts

Programación de Scripts

- En esta asignatura generalmente programaremos los scripts en python, que es más potente y sencillo que bash
- Pero para tareas muy básicas (cp, mv, ln -s, etc) puede ser más conveniente un script de bash

```
#!/bin/bash
a="hola mundo"
echo $a
```

Para invocarlo:

```
koji@mazinger:~$ ./holamundo
hola mundo
```

Es recomendable que un script empiece por `#!/bin/bash`, pero no es imprescindible

```
a="hola mundo"
echo $a
```

En este caso podemos ejecutar una shell y pasarle como primer argumento el fichero

```
koji@mazinger:~$ bash holamundo
hola mundo
```

o bien ejecutar una shell y redirigir el fichero a su entrada estándar

```
koji@mazinger:~$ bash <holamundo
hola mundo
```

Esto también puede ser útil para ejecutar un script sin permiso de ejecución (basta el de lectura)

9.11. Filtros

Filtros

- Los filtros son muy importantes en el scripting Unix: grep, sed, sort, uniq, head, tail, paste...
- Un mandato genera una salida, un filtro procesa la salida (selecciona filas o columnas, pega, reemplaza, cuenta, ordena...) y lo pasa al siguiente mandato
- Ejemplo

```
who | cut -c1-8 |sort |uniq | wc -l

ps -ef | grep miguel | grep -v gvim
```

- En esta asignatura programaremos en python (de nivel más alto y más intuitivo), así que solo usaremos filtros muy básicos

grep

- grep es un filtro que selecciona las filas que contengan (o que no contengan) cierto patrón
- Para definir patrones de texto, emplea expresiones regulares (regexp)
  - Las regexp de grep, sed y awk son *clásicas*.
  - Las regexp de perl, python y ruby son una evolución de las regexp clásicas. Son mucho más intuitivas
  - Para tareas muy sencillas, podemos usar grep o sed. Si nuestras necesidades son más complejas y podemos elegir qué herramienta usar, mejor python (o ruby)

grep con un argumento

- `grep <patrón>`

  Lee stdin y escribe en stdout las líneas que encajen en el patrón
- `grep -v <patrón>`

  Lee stdin y escribe en stdout las líneas que **no** encajen en el patrón

- `grep -i <patrón>`

  Lee stdin y escribe en stdout las líneas que encajen en el patrón, ignorando mayúsculas/minúsculas

Ejemplos

```
ps -ef | grep -i ejemplo
ps -ef | grep -v jperez
dmesg | grep eth
```

grep con dos o más argumentos

- `grep <patrón> <fichero_1> ... <fichero_n>`

  Lee los ficheros indicados y escribe en stdout las líneas que encajen en el patrón

Ejemplos

```
grep linux *.txt
grep -i hidalgo quijote.txt
grep -v 193.147 /etc/hosts
```

Atención: Si el patrón a buscar incluye espacios, es necesario escribirlo entre comillas.

- `grep "la mancha" quijote.txt`

  Busca el patrón *la mancha* en el fichero *quijote.txt*

- `grep la mancha quijote.txt`

  Busca el patrón *la* en el fichero *mancha* y en el fichero *quijote.txt*

Atención:

- Hablamos de patrones, no de palabras. El patrón *ana* encaja en la palabra *ana* pero también en *rosana*

- Los metacaracteres de las regexp no son iguales que los metacaracteres (comodines) del bash

Algunos metacaracteres:

- `grep -i '\<ana\>'`

  Principio de palabra, patrón *ana*, final de palabra. Insensible a mayúsculas. (Dicho de otro modo, la palabra *ana*, sin confusión con *Mariana*)

- `grep -i '\<ana p.rez\>'`

  El punto representa cualquier carácter (equivalente a la interrogación en las shell de bash)

- `grep -i '\<ana p[eé]rez\>'`

  Después de la *p* puede haber una *e* con tilde o sin tilde

xargs

Mediante pipes podemos formar filtros concatenando órdenes. Pero ¿qué sucede cuando la información la necesitamos como parámetro, no en la entrada estándar?

```
locate -i basura | rm    # ¡Esto NO FUNCIONA!
```

Podemos usar la orden xargs

```
locate -i basura | xargs rm
```

Ejecuta rm tantas veces como líneas haya en stdin. Y le pasa cada línea como argumento

- Cuando necesitamos que la línea de entrada vaya en una posición distinta, usamos la opción `-I replstr` , donde `replstr` es la *replace string*, la cadena que reemplazaremos por el argumento

- El valor recomendado es `{}`, porque no es fácil que aparezca en otro sitio

```
locate  basura | xargs -I {} mv {} /tmp/papelera
find . | grep -i jpg | xargs -I {} mv {} /tmp/fotos
```

9.12. Usos no estándar de la barra

Usos no estándar de la barra

Un principio básico para hacer buenos programas es
se laxo con lo que aceptas y estricto con lo que generas

- `/d1//d2///d3/d4`

  En rigor es un nombre incorrecto. Aunque normalmente se admite, porque la shell y las librerías lo *limpian* y generan

  `/d1/d2/d3/d4`

  No hay garantía de que funcione siempre, es mucho mejor evitarlo

- `/d1/`

  Algunas órdenes y algunos documentos muestran una barra al final de un directorio para indicar que se trata de un directorio y no un fichero ordinario (de la misma manera que puede usarse un color distinto)

  Algunas órdenes pueden esperar que un nombre acabado en barra sea un directorio

  Pero no es un nombre estándar, es preferible evitarlo

- Para la orden cp de Mac OS

  `cp -r d/ .`

  significa

  `cp -r d/* .`

  (pero solo para cp -r y solo para Mac OS)

9.13. Ordenes internas

Ordenes internas

La mayoría de las órdenes son externas

Pero todas las shell interpretan ciertas órdenes por sí mismas: Las órdenes internas (*builtin commands*)

- Por razones de eficiencia: `echo, kill, pwd, test...`

  Son internas aunque también tienen versión externa

- Necesariamente internas: `cd, export, alias, unset, exit...`

  Realizan funciones que tienen que hacerse forzosamente en el proceso de la shell, harían algo completamente diferente si se implementan como ejecutables externos

```
koji@mazinger:~$ type echo
echo es una función integrada en la shell
```

alias

Reemplaza una cadena por otra

- `alias c='clear'`

  Expande c, se convierte en *clear*

- `alias`

  Muestra todos los alias

- `unalias c`

  Deshace el alias

alias suele definirse en `.bashrc`

Hay ataques/bromas basados en alias

Funcionamiento de la shell

1. La shell lee texto de cierto fichero (stdin). Frecuentemente el texto lo está escribiendo el usuario, así que aporta algunas facilidades (borrar, autocompletar, history)
2. Analiza el texto (expande metacaracteres, variables, alias)
3. Busca la primera palabra, para ver si se trata de un ejecutable
   - Primero la busca entre las órdenes internas
   - Si no es interna, busca el ejecutable en ciertos directorios (los indicados en el PATH)
4. Aplica las redirecciones que correspondan
5. Ejecuta, pasando el resto de palabras como argumento
6. Duerme
   - A menos que lancemos el ejecutable en *background*
     `acroread file.1 &`

History

Facilita la entrada de líneas

- (cursor arriba y abajo)

  Muestra, una a una, las órdenes introducidas

- `!<cadena>`

  Repite la última orden que empiece por `<cadena>`

- `history`

  Muestra el historial de órdenes introducidas

- `!<n>`

  Repite la órden `<n>`

9.14. Permisos especiales

9.14.1. SUID

SUID

Sea un fichero perteneciente a un usuario

```
-rwxr-xr-x 1 koji koji 50 2009-03-24 12:06 holamundo
```

Si lo ejecuta un usuario distinto

```
invitado@mazinger:~$ ./holamundo
```

El proceso pertenece al usuario que lo ejecuta, no al dueño del fichero

```
koji@mazinger:~$ ps -ef |grep holamundo
invitado  2307  2260 22 12:16 pts/0    00:00:00 holamundo
koji      2309  2291  0 12:16 pts/1    00:00:00 grep holamundo
```

Este comportamiento es el normal y es lo deseable habitualmente

Pero en ocasiones deseamos que el proceso se ejecute con los permisos del dueño del ejecutable, no del usuario que lo invoca

- Esto se consigue activando el bit SUID (*set user id*)

  `chmod u+s fichero`

  `chmod u-s fichero`

  En un listado detallado aparece una `s` en lugar de la `x` del dueño (o una `S` si no había `x`)

- El bit SUID permite que ciertos usuarios modifiquen un fichero, pero no de cualquier manera sino a través de cierto ejecutable

```
-rwsr-xr-x 1 root root 29104 2008-12-08 10:14 /usr/bin/passwd
-rw-r--r-- 1 root root  1495 2009-03-23 19:56 /etc/passwd
```

- El bit SUID también puede ser un problema de seguridad
- En el caso de los scripts, lo que se ejecuta no es el fichero con el script, sino el intérprete

  Un intérprete con bit SUID es muy peligroso, normalmente la activación del SUID en un script no tiene efecto
- Para buscar ficheros con SUID activo:

  `find / -perm +4000`
- El bit SGID es análogo, cambia el GID

  `chmod g+s fichero`

9.14.2. Sticky bit

Sticky bit

- En ficheros ya no se usa
- En un directorio, hace que sus ficheros solo puedan ser borrados o renombrados por el dueño del fichero, del directorio o el *root*

Se representa con una `t`, en el listado y en chmod

`chmod [+-]t directorio`

`drwxrwxrwt 15 root root 4096 2007-02-21 13:36 /tmp/`

Si el directorio no tuviera permiso de ejecución, aparecería `T`

`drwxrwx-wT`

9.14.3. Umask

Umask

Orden interna que muestra y cambia la variable umask
(*user file creation mode mask*)

- `umask`

  Devuelve el valor umask

- `umask nuevo-valor`

  Cambia el valor umask

¿Qué permisos tiene por omisión un fichero recién creado?

- Ficheros: `666 and not umask`
- Directorios: `777 and not umask`

Ejemplo. Creación de un fichero
Calculamos el valor de umask negado

```
umask       022            000    010    010
not umask   755            111    101    101
```

Hacemos *and lógico* entre 666 y el valor de umask negado

```
            666            110    110    110
                  and
not umask   755            111    101    101
--------------------------------------------
            644            110    100    100
                           rw-    r--    r--
```

9.15. source

source

Ejecuta un fichero en el entorno de la shell actual, que no muere. Las variables usadas en el fichero importado serán por tanto variables del proceso actual

El mandato *punto* (.) es equivalente, (aunque puede resultar menos legible)

- `. ~/.bashrc`
  ```
  # Ejecuta el código de .bashrc
  # en el entorno actual
  ```

- `source ~/.bashrc # Forma equivalente`

9.16. Invocación de la shell

Invocación de la shell

- Es frecuente desear que todas nuestras sesiones ejecuten o configuren algo, sin necesidad de teclearlo a mano cada vez. Para hacer esto necesitamos saber cómo funciona la *invocación de la shell*
- Cada vez que se invoca una shell, esta ejecuta (con source) cierto fichero
- Típicamente esto se emplea para definir y exportar variables de entorno, modificar el prompt, declarar alias...
- Cada tipo de shell ejecuta un fichero diferente
  - Una shell puede ser de login o no de login
  - Una shell puede ser interactiva o no interactiva
- Una shell de login es aquella en la que el usuario ha tenido que autenticarse.
  - O bien introducido login y contraseña
  - O bien el demonio de ssh lo hace por él (configurando una clave pública, una privada, etc)
- En general, una shell interactiva es aquella que tiene `stdin` redirigida desde la consola de un usuario, y `stdout` y `stderr` redirigidos a la consola de un usuario

Bash interactivo y de login

Ejemplos:

- Una sesión en una máquina sin gráficos (p.e. un Unix antiguo, un router...)
- Una sesión sin gráficos en una máquina con gráficos, que se inicia pulsando Ctrl+Alt+F1
- Entrar por ssh en una máquina

En este caso, la shell

- Lee y ejecuta `/etc/profile`

- Después, ejecuta el primero que encuentre de

  ```
  ~/.bash_profile
  ~/.bash_login
  ~/.profile
  ```

  No se ejecuta `.bashrc`, a menos que `.bash_profile` lo llame.

  Al terminar ejecuta

  ```
  ~/.bash_logout
  ```

Bash interactivo, no de login

Ej: Un terminal en Gnome o en Fluxbox

Se ejecuta

- `~/.bashrc`

No se ejecuta `~/.bash_profile`

Bash no interactivo, no de login

Ej: Un script

- Se ejecuta el fichero `$BASH_ENV`
- Antes del `.bashrc` de cada usuario, se ejecuta `/etc/bash.bashrc`, común para todos los usuarios
- Cuando se crea un usuario con `adduser`, se copia en su *home* todos los fichero que haya en `/etc/skel` (aquí se guardan los ficheros de configuración por omisión para cada usuario)
- Hablamos siempre del inicio de la shell. No debemos confundir todo esto con los niveles de ejecución, que se refienen al inicio de la máquina (directorios `/etc/rc2.d`, `/etc/rcS.d`, etc)

- Actualmente la diferencia entre shell de login y shell no de login es algo artificial [8]

[8] En un linux con gráficos, una sesión ordinaria no ejecuta ninguna shell de login, mientras que en macOS todas las shell que ejecuta el usuario son de login

- Hoy no suele resultar conveniente tener un fichero para las de login (`~/.bash_profile`) y otro distinto para las que son no de login (`~/.bashrc`)
- Por tanto, lo normal es configurar todo lo necesario en `~/.bashrc` y tener en `~/.bash_profile` únicamente una llamada a `~/.bashrc`, de la siguiente manera:

```
if [ -f ~/.bashrc ]; then   # si existe .bashrc
    . ~/.bashrc             # ejecuta .bashrc
fi
```

O lo que es lo lo mismo

```
if test  -f ~/.bashrc ; then   # si existe .bashrc
    source ~/.bashrc             # ejecuta .bashrc
fi
```

9.17. Manejo básico de procesos

Manejo básico de procesos

- `ps` Información sobre los procesos
  `ps -e` Información sobre todos los procesos de la maquina
  `ps -ef` Formato largo

- `top` Muestra los procesos que consumen más cpu
- `kill` Envia una señal a un proceso

Señales

La orden kill envía señales a procesos
`kill [señal] [proceso]`

- 15 SIGTERM (valor por defecto)
- 9 SIGKILL
- 2 SIGINT (Ctrl C) Lo envia tty a todos los programas que se estén ejecutando en primer plano en el terminal, y a todos los programas lanzados por estos.

- 19 SIGSTOP (Ctrl Z) Detiene
- 18 SIGCONT Continua si estaba detenido

Las señales SIGKILL y SIGSTOP no se pueden ignorar ni bloquear
Ejemplos:
`kill -9 2341`
`kill -sigstop 49322`
Tabla con las señales:
`man 7 signal`

- Una manera típica de localizar un proceso *a mano* es `ps -ef | grep <cadena>`

  o

  `ps -ef | grep <cadena> | less`

- `killall` envía señales a procesos a partir de su nombre. (El nombre de la señal se indica de manera ligeramente distinta a como se emplea en kill)
- `pkill` envía señales a procesos, identificables mediante nombre u otros atributos

9.18. Tareas

Control de tareas (jobs)

- Para lanzar varios procesos que se ejecuten en paralelo lo más cómodo suele ser abrir varias shells (una nueva terminal o una nueva pestaña en el terminal o un multiplexor de terminal como tmux)
- Pero también es posible desde una única shell manejar varios procesos simultáneamente: mediante el control de tareas (jobs)

Un proceso puede ejecutarse en primer o en segundo plano

- En primer plano (*foreground*) recibe órdenes desde el teclado, como Ctrl Z (detener temporalmente) o Ctrl C (finalizar)

  Cada shell solo puede tener un proceso en primer plano

- En segundo plano no tiene vinculada su entrada estándar desde el teclado, no recibe las señales Ctrl Z o Ctrl D. Es necesario emplear *kill*

  Puede haber varios procesos en segundo plano

De la misma manera, un proceso detenido puede estar tanto en primer como en segundo plano

- La orden *jobs* indica, en cada línea, número de tarea, estado y nombre

```
koji@mazinger:~$ jobs
[1]   Ejecutando              xcalc &
[2]-  Ejecutando              evince &
[3]+  Detenido                gedit
```

- El signo `+` indica tarea por omisión, aquella que se sobreentiende si no se indica número de tarea. Si la tarea por omisión muere, la siguiente será la marcada con el signo `-`
- `kill %n` envía señal al proceso con el job `n` (El símbolo de porcentaje indica nº de job, su ausencia indica pid)
- Algunos programas multiproceso, complejos, aunque los lancemos desde una shell no son hijos de esa shell y no figurarán en la lista de tareas. Por ejemplo firefox o nautilus

- `fg n`

  pone la tarea `n` en ejecución en primer plano
- `bg n`

  pone la tarea `n` en ejecución en segundo plano

  El resultado es el mismo que si hubiéramos lanzado la orden con el símbolo `&`

Las órdenes `bg` y `fg` pueden lanzarse sin indicar `n`, entonces se sobreentiende la tarea por omisión.

La orden `kill` necesita que se le indique siempre explícitamente el número de tarea o el numero de pid

```
vmstat 1 Lanzo vmstat, indicando que se actualice cada 1 segundo.
Ctrl Z   Detengo el proceso. La shell me indica su número de trabajo.
fg 1     El trabajo 1 vuelve a primer plano. No puedo usar la shell.
Ctrl Z   Vuelvo a detenerlo.
```

```
jobs     Listado de todos los trabajos.
bg 1     El trabajo 1 se ejecuta en segundo plano. Sigue escribiendo
         en stdout, pero puedo usar la shell.
         En este momento no puedo matarlo con ctrl C.
fg       El trabajo pasa a primer plano, puedo matarlo.
```

9.19. Tmux

nohup

- Normalmente, cuando un usuario cierra una sesión, todos sus procesos reciben la señal SIGHUP y mueren
- Si tenemos procesos que queremos que se continúen ejecutando aunque el usuario cierre la sesión, podemos usar `nohup`
  - `nohup <orden>`
    `<orden>` ignorará la señal SIGHUP. Escribirá stdout en ./nohup.out (o en `~/nohup.out`)
  - Si necesitamos stdin, es necesario redirigirla desde un fichero

tmux

- Los *multiplexores de terminales* son una alternativa a `nohup` mucho más potente: Además de mantener el proceso vivo cuando el usuario se desconecta, posteriormente se puede seguir usando interactivamente stdin y stdout
- Otra ventaja:
  - Normalmente, si deseo tener n sesiones en una máquina remota, es necesario abrir n conexiones mediante ssh
  - Usando un multiplexor, puedo abrir una única conexión ssh a una sesión del multiplexor, y en ella usar n ventanas
- El más tradicional es GNU Screen (año 1987). Aquí veremos Tmux (año 2007), un programa similar con algunas mejoras
- Inconvenientes:
  - Es necesario memorizar algunos atajos de teclado

tmux maneja *sesiones*

- Una sesión de tmux permite que un usuario se asocie (attach) y se desasocie de ella (dettach). El usuario puede desconectarse y la sesión permanece (todos los procesos se siguen ejecutando). Cuando el usuario vuelva a conectarse (típicamente por ssh) puede reasociarse (reattach)

En cada sesión puede haber diferentes *ventanas*

- No son ventanas al estilo Microsoft Windows / X Window ni incluso ncurses, porque cada ventana de tmux ocupa toda la consola disponible
- Se parece a tener varias pestañas en un gnome-terminal, o a diferentes sesiones en alt F1, alt F2
- A su vez, cada una de las ventanas se puede dividir en *paneles*, que sí se parecen a las ventanas de Microsoft Windows / X Window

El uso de tmux se basa en comandos, que tendremos que memorizar

- Cada comando está formado por la pulsación de la tecla *bind*, y a continuación, una letra
- La tecla *bind* por omisión es `Ctrl b`

Tal vez prefieras que la tecla *bind* sea otra. Por ejemplo `Ctrl a`, puede tener dos ventajas

- Posiblemente es más ergonómico
- Es la tecla que usa *screen*

En ese caso, añadiríamos lo siguiente en `~/.tmux.conf`

```
set -g prefix C-a
bind C-a send-prefix
unbind C-b
```

Uso típico de tmux

```
tmux       Creamos una sesión de tmux y nos asociamos a ella.
           Si ya había sesión, también nos asociamos pero creando
           una ventana nueva.
<bind> w   Vemos las ventanas de la sesión. Nos desplazamos entre
           ellas con los cursores (flechas del teclado) e intro.
<bind> d  Nos desasociamos de la sesión actual.
```

Desconectamos ssh o cerramos el terminal. Volvemos a conectarnos

```
tmux attach   Nos reasociamos a la sesión, sin crear más ventanas.
<bind> w      Vemos las ventanas y nos desplazamos a la deseada.
```

Para añadir ventanas la sesión actual pulsamos `<bind> c`

Para cerrar definitivamente una ventana, cerramos la shell de la forma habitual (`exit` o `Ctrl d`)

Con lo visto hasta ahora, tmux ya resulta de gran utilidad. Pero seguramente querremos dividir cada ventana en *paneles*

Una vez dentro de una ventana de una sesión, haremos lo siguiente

```
<bind> %              Divide la ventana en dos paneles
<bind> %              Vuelve a realizar otra division
                      (repetir esto las veces deseadas)
                      ...

<bind> [espacio]      Cambia la disposición de los paneles
<bind> [espacio]      Cambia la disposición de los paneles
                      (repetir esto las veces deseadas)
                      ...
```

Una vez que hemos ajustado a nuestro gusto el número de paneles y su disposición, para mover el foco (esto es, marcar cuál es el panel activo), usamos los cursores del teclado

```
<bind>  Derecha     Foco al panel a la derecha del actual
<bind>  Izquierda   Foco al panel a la izquierda del actual
<bind>  Arriba      Foco al panel encima del actual
<bind>  Abajo       Foco al panel debajo del actual
```

Aquí puedes ver una demostración de todo esto: https://youtu.be/Ks-a4YsYciM

No es necesario conocer más atajos para manejar lo fundamental de sesiones, ventanas y paneles. Aunque tmux tiene muchos otros comandos. P.e.

```
<bind> n         Moverse a la siguiente ventana
<bind> p         Moverse a la ventana previa
<bind> ,         Renombrar ventana actual
<bind> {         Mover el panel actual a la izquierda
<bind> }         Mover el panel actual a la derecha
<bind> [número]  Llevar el foco a la ventana [número]
```

Si pulsamos `<bind>`, y sin soltarlo, pulsamos uno los cursores del teclado, redimensionamos el panel en la dirección del cursor

10. Copias de seguridad

Copias de Seguridad

Los datos almacenados en un disco se pueden perder en cualquier momento, están expuestos a

- Fallos hardware
- Errores humanos
- Malware (virus)
- etc

En cualquier sistema, las copias de seguridad son de vital importancia

- Para uso personal, cuando tenemos pocos datos y buena conexión a internet, el almacenamiento en la nube con servicios *freemium* como Google Cloud o Dropbox es una buena solución
- Para entornos más exigentes, es preferible usar nuestros propios dispositivos (pendrives o discos externos)

¿Por qué necesitamos una aplicación de sincronización?

Supongamos que queremos respaldar nuestro directorio de trabajo en un disco externo recién comprado

- El primer día, lo copiamos todo
- El segundo día queremos guardar también las copias recientes.

  ¿Qué hacemos?

  - Podemos copiarlo todo de nuevo. Pero estamos desperdiciando recursos (tiempo, acceso a la red, tiempo de vida del disco)
  - Podemos seleccionar a mano las novedades y no copiar nada más. Pero es un trabajo tedioso y propenso a errores

La solución es una herramienta de sincronización, que se ocupa de copiar sólo los ficheros y directorios que se hayan añadido o modificado, automáticamente

10.1. rsync

rsync

Permite sincronización unidireccional de dos directorios

- Herramienta habitual en Unix (incluyendo Linux y OS X). Hay versiones para Windows, pero su uso no es muy cómodo, requiere de componentes adicionales
- El esclavo contendrá una réplica del maestro. El maestro ignora el contenido del esclavo
- Alternando las funciones de maestro y esclavo podríamos conseguir una especia de sincronización unidireccional, pero no es recomendable, no está diseñado para ello
- Trabaja con sistemas de ficheros locales o remotos, pero al menos uno de los dos debe ser local
- Funciona sobre rcp, ssh o un demonio de rsync
- Es muy útil para hacer copias de segurad mediante *mirrors* de disco duro sobre disco duro: automatizarlo es muy sencillo y el disco duro es el medio de almancenamiento más barato de la actualidad
- Pero también resulta un sistema de respaldo muy volátil, deberíamos tener, adicionalmente, otros sistemas de respaldo más permanentes

```
rsync   <opciones>    dir_maestro  dir_donde_escribir_esclavo
```

- El directorio remoto se especifica anteponiendo `usuario@maquina:`

Ejemplo

```
rsync -e ssh -va --delete /dir1/dir2  jperez@epsilon03:/dir3/dir4
```

- Crea dentro de dir4 un directorio dir2, réplica del dir2 del maestro

  Error frecuente: creer que dir4 será una réplica de dir2
- Error frecuente: insertar espacio tras los dos puntos

Opciones

- `-e` especifica cómo acceder a los ficheros en la máquina remota
- `-v` verbose (prolijo en detalles)
- `-a` modo archive: incluye subdirectorios, copia enlaces como enlaces, conserva permisos, fechas y grupo. Si lo ejecuta el root, también conserva el dueño
- `--delete` indica que si se borra un fichero en el maestro, también se borrará en el esclavo

  Atención: si montamos un nuevo maestro, vacío, y aplicamos rsync con esta opción, destruiremos el esclavo

10.2. FreeFileSync

FreeFileSync

FreeFileSync es una aplicación de sincronización de ficheros

- Permite comparar dos directorios, detectar las diferencias y propagarlas, de forma que ambos directorios acaben teniendo el mismo contenido
- Los directorios pueden estar en la máquina local o en una máquina remota accesible a través de FTP o SFTP. También soporta Google Drive
- Libre y gratuita, disponible para Microsoft Windows, Linux y macOS
- Software maduro, aparece en 2008 y se actualiza con frecuencia

Con FreeFileSync haremos principalmente dos tipos de sincronización

- Sincronización bidireccional
- Espejo

En ambos casos

- Antes de sincronizar tendremos dos directorios que normalmente serán parecidos pero con algunas diferencias
- Después de sincronizar, ambos directorios serán idénticos

Ambos directorios podrán estar en cualquier lugar

- En el mismo disco de la misma máquina
- En otro disco de la misma máquina
- En otro disco de otra máquina
- Incluso en el mismo disco pero otra máquina (como en nuestro laboratorio)

Sincronización bidireccional

En la sincronización bidireccional:

- Todas las novedades del primer directorio se propagarán al segundo (ficheros nuevos, modificados o borrados, directorios nuevos, modificados o borrados)
- Todas las novedades del segundo directorio se propagarán al primero

Tras la sincronización, ambos directorios serán idénticos
Ejemplo típico:

- El primer directorio es mi directorio de trabajo en mi pc de casa
- El segundo directorio es mi directorio de trabajo en el laboratorio

Figura 5: Pantalla principal de FreeFileSync

Ventana principal. El panel de la izquierda muestra el directorio local. El de la derecha, el remoto

Espejo

En la sincronización *espejo*, tenemos dos directorios

- Uno es el principal

- Otro será una copia

Después de la sincronización, ambos serán idénticos
Para ello:

- Todas las novedades del principal se llevarán a la copia
- En el directorio copia no debería haber novedades, y si las hay, serán ignoradas y destruidas

Ejemplo típico:

- El primer directorio es mi directorio de trabajo
- El segundo es una copia de seguridad en un disco externo

Cómo sincronizar
Para sincronizar los directorios

- Indicamos el primer directorio en el panel de la izquierda
- Indicamos el segundo directorio en el panel de la derecha
- Pulsamos el icono de la rueda dentada verde para especificar el tipo de sincronización (bidireccional o espejo)
- Pulsamos *comparar*
- Pulsamos *sincronizar*

Pulsando el icono del filtro, tenemos la opción de excluir algunos ficheros. Por ejemplo `*.o`, `*.exe`

En la ventana de la rueda dentada verde indicamos el tipo de sincronización

Desde el icono de la nube accedemos a la ventana de *almacenamiento en línea*, donde podemos indicar que el directorio estará en una máquina remota. Por ejemplo en el laboratorio de la EIF, accesible mediante el protocolo SFTP

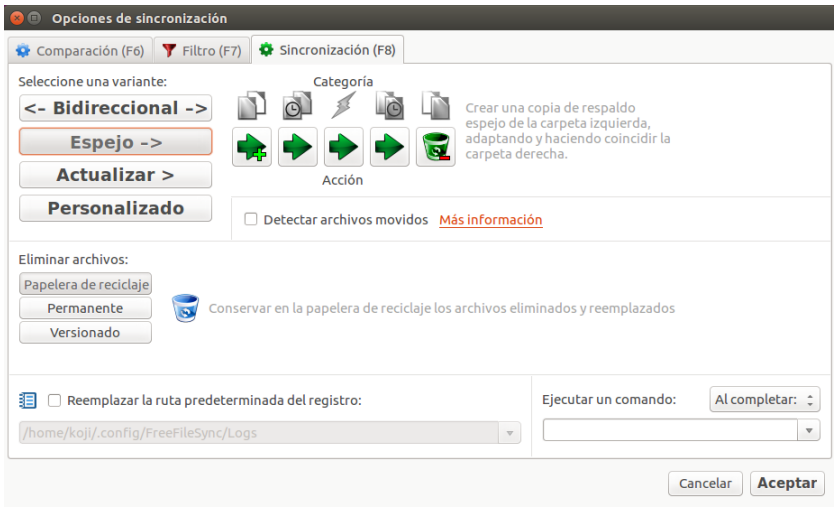
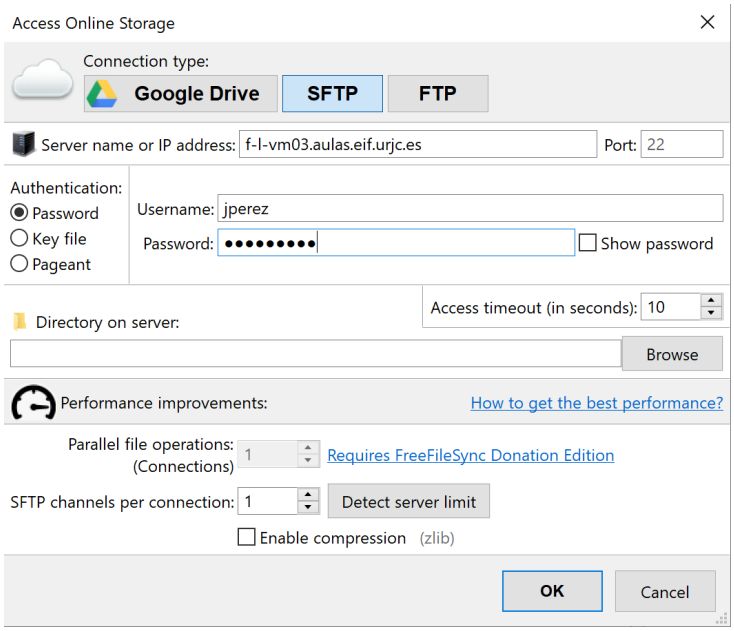


Figura 6: Opciones de sincronización



Conflictos

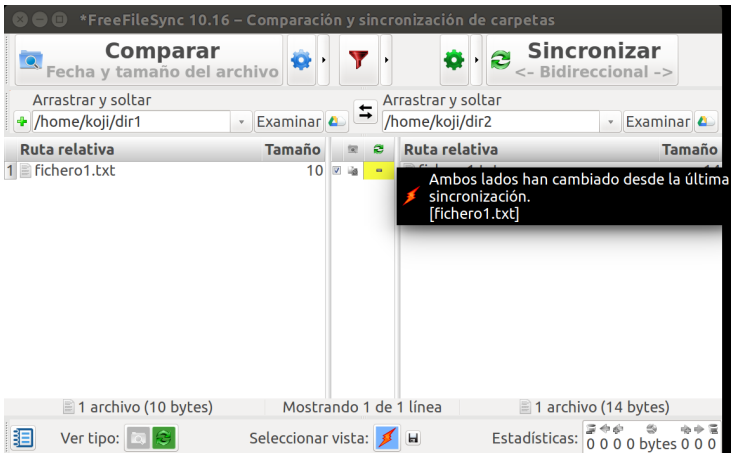
En la sincronización bidireccional, es importante sincronizar siempre que cambiemos de máquina, de lo contrario se producirá un *conflicto*

Ejemplo:

- El lunes trabajo en casa y sincronizo el directorio contra el laboratorio
- El martes trabajo en el laboratorio
- El miércoles vuelvo a casa, pero olvido sincronizar. Añado mis cambios sobre la versión del lunes, no sobre la del martes

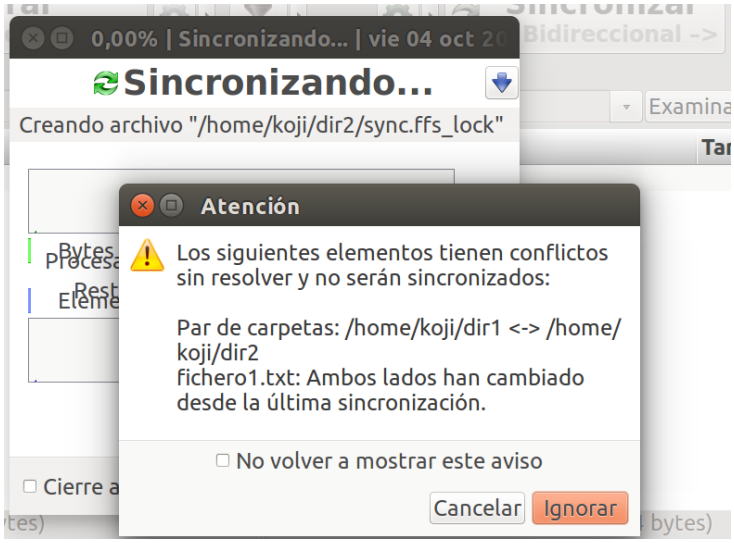
En este caso, tendré cambios que solo estarán en casa, y otros que solo estarán en el laboratorio

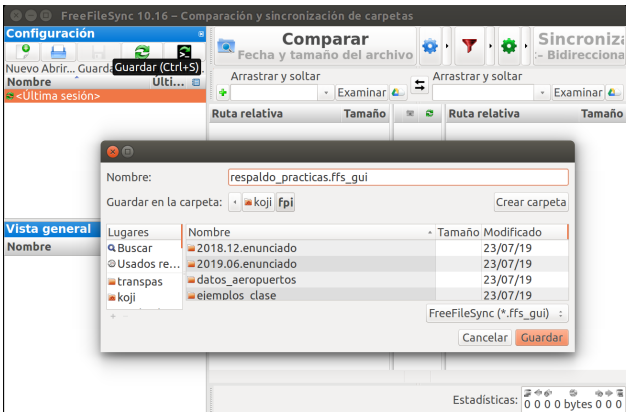
FreeFileSync nos avisará del problema, que tendremos que corregir a mano



El conflicto se señala con un icono en forma de chispa naranja

El problema tendremos que corregirlo nosotros manualmente: hacemos lo necesario para que uno de los directorios esté *correcto* y luego indicar en qué sentido debe hacerse la actualización


0,00% | Sincronizando... | vie 04 oct 20
Sincronizando...
Creando archivo "/home/koji/dir2/sync.ffs_lock"
Atención
Los siguientes elementos tienen conflictos sin resolver y no serán sincronizados:
Par de carpetas: /home/koji/dir1 <-> /home/koji/dir2
fichero1.txt: Ambos lados han cambiado desde la última sincronización.
No volver a mostrar este aviso
Cancelar
Ignorar


FreeFileSync 10.16 – Comparación y sincronización de carpetas
Configuración
Comparar
Fecha y tamaño del archivo
Guardar (Ctrl+S)
Nombre:
respaldo_practicas.ffs_gui
Guardar en la carpeta:
Crear carpeta
Vista general
Nombre
Tamaño
Modificado
2018.12.enunciado
2019.06.enunciado
datos_aeropuertos
ejemplos_clase
FreeFileSync (*.ffs_gui)
Cancelar
Guardar
Estadísticas:

Guardar la configuración

Finalmente, guardaremos toda la configuración en un fichero de extensión `.ffs_gui`, para poder repetir la sincronización en la siguiente ocasión, sin volver a especificarlo todo.

- Aquí puedes ver una sesión de ejemplo: https://youtu.be/gJGp6liznE0
- Atención, en este vídeo se usan direcciones obsoletas (aulas.etsit.urjc.es), no las actuales: aulas.eif.urjc.es

11. El Lenguaje Python

11.1. Introducción

El Lenguaje Python

- Lenguaje *de autor* creado por Guido van Rossum en 1989
- Muy relacionado originalmente con el S.O. *Amoeba*
- Disponible en Unix, Linux, macOS, Windows,
- Libre
- Lenguaje de Script Orientado a Objetos (no muy puro)
- Muy alto nivel
- Librería muy completa
- Verdadero lenguaje de propósito general
- Sencillo, compacto
- Sintaxis clara
- Interpretado => Lento
- Ofrece persistencia
- Recolector de basuras
- Muy maduro y muy popular
- Aplicable para software de uso general

Programa python

```
for x in xrange(1000000):
          print x
```

Su equivalente Java

```
public class ConsoleTest {
       public static void main(String[] args) {
              for (int i = 0; i < 1000000; i++) {
                                System.out.println(i);
                                                }
                        }
}
```

Programa python

```
for i in xrange(1000):
         x={}
         for j in xrange(1000):
             x[j]=i
             x[j]
```

Su equivalente Java

```
import java.util.Hashtable;
public class HashTest {
    public static void main(String[] args) {
       for (int i = 0; i < 1000; i++) {
           Hashtable x = new Hashtable();
           for (int j = 0; j < 1000; j++) {
               x.put(new Integer(i), new Integer(j));
               x.get(new Integer(i));
           }
       }
    } }
```

Librerías

Python dispone de librerías *Nativas* y *Normalizadas* para

- Cadenas, listas, tablas hash, pilas, colas
- Números Complejos
- Serialización, Copia profunda y Persistencia de Objetos
- Regexp
- Unicode, Internacionalización del Software
- Programación Concurrente
- Acceso a BD, Ficheros Comprimidos, Control de Cambios...

Librerías relacionadas con Internet:

- CGIs, URLs, HTTP, FTP,
- pop3, IMAP, telnet

- Cookies, Mime, XML, XDR
- Diversos formatos multimedia
- Criptografía

La referencia sobre todas las funciones de librería podemos encontrarlas en la documentación oficial, disponible en el web en muchos formatos. Basta con localizar en cualquier buscador la *python standard library*

Inconvenientes de Python

Además de su velocidad limitada y necesidad de intérprete
(Como todo lenguaje interpretado)

- No siempre compatible hacia atrás
- Uniformidad.
  Ej: `función len(), método  items()`
- Algunos aspectos de la OO

  Python is a hybrid language. It has functions for procedural programming and objects for OO programming. Python bridges the two worlds by allowing functions and methods to interconvert using the explicit "self" parameter of every method def. When a function is inserted into an object, the first argument automagically becomes a reference to the receiver.
- ...

Versiones de python

- Python 1 (año 1991)
- Python 2 (año 2001)

  Incompatible con Python 1
- Python 3 (año 2009)

  Incompatible con Python 2. Versión problemática, durante la década de 2010 fue muy habitual seguir usando Python 2

El soporte para Python 2 finalizó oficialmente el 1 de enero de 2020. En la actualidad:

- Sigue habiendo mucho código antiguo en Python 2

- Sigue siendo recomendable que en cualquier sistema *python a secas* sea Python 2
- Deberíamos programar siempre en Python 3, indicando explícitamente que el intérprete es *python3*

11.2. El intérprete de python

El intérprete de python se puede usar

- En modo interactivo

```
koji@mazinger:~$ python3
Python 3.9.0 (default, Oct 27 2020, 14:13:35)
[Clang 11.0.0 (clang-1100.0.33.17)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hola, mundo")
Hola, mundo
>>> 3/2
1.5
```

- Mediante scripts

```
#!/usr/bin/env python3
# holamundo.py
print("hola mundo")    # esto es un comentario
euros=415
pesetas=euros*166.386
print("{} euros son {} pesetas".format(euros, pesetas))
```

La línea `#!/usr/bin/env python3` indica al S.O. dónde está la utilidad *env*, que buscará en el *path* el ejecutable *python3* y le pasaré el fuente para que lo ejecute

- Debe ser exactamente la primera línea
- No puede haber espacios entre la admiración y la barra

```
#Este ejemplo es doblemente incorrecto
#! /usr/bin/env python3
# ¡MAL!
```

En linux se puede especificar directamente la dirección del intérprete

```
#!/usr/bin/python3
print("Hola mundo")
```

Pero `env` es más general, permite que tengamos varios intérpretes en el sistema (algo muy normal en macOS)

11.3. Operadores

Operadores

En orden de precedencia decreciente:

```
+x, -x, ~x    Unary operators
x ** y    Power
x * y, x / y, x % y    Multiplication, division, modulo
x + y, x - y    Addition, subtraction
x << y, x >> y    Bit shifting
x & y    Bitwise and
x | y    Bitwise or
x < y, x <= y, x > y, x >= y, x == y, x != y,
x <> y, x is y, x is not y, x in s, x not in s
                        Comparison, identity,
                        sequence membership tests
not x     Logical negation
x and y    Logical and
lambda args: expr            Anonymous function
```

11.4. Identificadores

Identificadores (nombre de objetos, de funciones...):

- Letras inglesas de 'a' a 'z', en mayúsculas o minúsculas. Barra baja '_' y números
- Sensible a mayúsculas/minúsculas

Naturalmente, en las cadenas de texto y en los comentarios podemos escribir cualquier carácter en la codificación habitual en la actualidad (utf-8)

11.5. Tipado

Python es

- Dinámicamente tipado, (*dynamically typed*). No es estáticamente tipado (*statically typed*)

  Una variable puede cambiar su tipo, dinámicamente

- Fuertemente tipado, (*strongly typed*). No es débilmente tipado (*weakly typed*)

  Este concepto no es absoluto, decimos que ciertos lenguajes tienen tipado más fuerte o más débil que otros

  Si algún objeto, variable, método, función... espera cierto tipo de objeto/de dato:

- Un lenguaje fuertemente tipado ha de recibir o bien exactamente ese tipo o bien uno muy parecido, de forma que pueda hacerse una conversión automática sin pérdida de información

  Obliga al programador a conversiones explícitas. Esto resulta rígido, tal vez farragoso, pero facilita la seguridad
- Un lenguaje débilemente tipado, admite casi cualquier cosa.

  Esto resulta cómodo, flexible, potencialmente peligroso

11.6. Declaración de objetos

En Python la declaración de objetos (variables) es implícita
(no hay declaración explícita)

- Las variables "nacen" cuando se les asigna un valor
- Las variables "desaparecen" cuando se sale de su ámbito
- La declaración implícita de variables como en perl puede provocar resultados desastrosos

```
#!/usr/bin/perl
$sum_elementos= 3 + 4 + 17;
$media=suma_elementos / 3;    # deletreamos mal la variable
print $media;   # y provocamos resultado incorrecto
```

- Pero Python no permite referenciar variables a las que nunca se ha asignado un valor.

```
#!/usr/bin/env python3
sum_elementos= 3 + 4 + 17
media=suma_elementos / 3    # deletreamos mal la variable
print(media)   # y el intéprete nos avisa con un error
```

11.7. Funciones predefinidas

Funciones predefinidas

- `abs()` valor absoluto
- `float()` convierte a float
- `int()` convierte a int
- `str()` convierte a string
- `round()` redondea
- `input()` acepta un valor desde teclado

11.8. Tabulación

Sangrado y separadores de sentencias

- ¡En Python NO hay llaves ni `begin-end` para encerrar bloques de código! Un mayor nivel de sangrado indica que comienza un bloque, y un menor nivel indica que termina un bloque.
- Las sentencias se terminan al acabarse la línea (salvo casos especiales donde la sentencia queda "abierta": en mitad de expresiones entre paréntesis, corchetes o llaves).
- El carácter \ se utiliza para extender una sentencia más allá de una linea, en los casos en que no queda "abierta".
- El carácter : se utiliza como separador en sentencias compuestas. Ej.: para separar la definición de una función de su código.
- El carácter ; se utiliza como separador de sentencias escritas en la misma línea.

- La recomendación oficial es emplear 4 espacios para cada nivel de sangrado
  - *PEP-8 Style Guide for Python Code*
  - David Goodger, *Code Like a Pythonista: Idiomatic Python*
    Traducción al español:
    Programa como un Pythonista: Python Idiomático
- Emplear 8 espacios o emplear tabuladores es legal, con tal de que no lo mezclemos [9]

11.9. Tipos de objeto

Tipos de objeto

En python todo son objetos: cadenas, listas, diccionarios, funciones, módulos...

[9] En python 2 era necesario añadir la opción -tt para que el intérprete detectase este problema

- En los lenguajes de scripting más antiguos como bash o tcl, el único tipo de datos es la cadena
- Los lenguajes imperativos más habituales (C, C++, pascal...) suelen tener (con variantes) los tipos: booleano, carácter, cadena, entero, real y matriz
- Python tiene booleanos, enteros, reales y cadenas. Y además, cadenas unicode, listas, tuplas, números complejos, diccionarios, conjuntos...
    - En terminología python se denominan *tipos de objeto*
    - Estos tipos de objeto de alto nivel facilitan mucho el trabajo del programador

En python es muy importante distinguir entre

- Objetos inmutables: Números, cadenas y tuplas
    - Se pasan a las funciones por valor
    - Si están declarados fuera de una función son globales y para modificarlos dentro de la función, es necesaria la sentencia *global*
- Objetos mutables: Todos los demás
    - Se pasan a las funciones por referencia
    - Si están declarados fuera de una función son globales, pero no hace falta la sentencia *global* para modificarlos dentro de la función, puesto que pueden ser modificados a través de sus métodos

Comprobación de tipos

```
#!/usr/bin/env python3
# comprueba_tipos.py

if isinstance("Bla blá", str):
    print("ok, es una cadena")
else:
    print("no es una cadena")
```

Tipos de objeto habituales:

```
bool
int
float
list
str
dict
tuple
```

11.9.1. Cadenas

Cadenas

- No existe tipo `char`
- Podemos emplear indistintamente la comilla simple o la doble (con tal de que el cierre coincida con la apertura)

  ```
  print("hola")
  print('hola')
  print('me dijo "hola"')
  ```

  Lo que resulta más legible que escapar los caracteres especiales

  ```
  print('me dijo \'hola\'')
  ```

- Se permiten caracteres especiales, p.e. nueva línea

  ```
  print("hola\nque tal")
  ```

- Cadenas crudas: esto imprime la barra y la n, literalmente

  ```
  print(r"""hola\nque tal""")
  ```

- El operador `+` concatena cadenas, y el `*` las repite un número entero de veces
- Para concatenar una cadena con un objeto de tipo diferente, podemos convertir el objeto en cadena mediante la función `str()`

```
>>> gamma=0.12
>>> print "gamma vale "+str(gamma)
gamma vale 0.12
```

- Se puede acceder a los caracteres de cadenas mediante índices y rodajas como en las listas
- Las cadenas son inmutables. Sería erróneo `a[1]=...`

11.9.2. Listas

Listas

- Tipo de datos predefinido en Python, va mucho más allá de los arrays
- Es un conjunto *indexado* de elementos, no necesariamente homogéneos
- Sintaxis:Identificador de lista, mas índice entre corchetes
- Cada elemento se separa del anterior por un carácter `,`

```
#!/usr/bin/env python3
# listas01.py
a=['rojo','amarillo']
a.append('verde')
print(a)      # ['rojo', 'amarillo', 'verde']
print(a[2])   # verde
print(len(a)) # 3

b=['uno',2, 3.0]  # Lista heterogénea
```

- El primer elemento tiene índice 0.
- Un índice negativo accede a los elementos empezando por el final de la lista. El último elemento tiene índice -1.
- Pueden referirse *rodajas* (*slices*) de listas escribiendo dos índices entre el carácter `:`
- La rodaja va desde el *primero, incluido*, al *último, excluido*.
- Si no aparece el primero, se entiende que empieza en el primer elemento (0)
- Si no aparece el segundo, se entiende que termina en el último elemento (incluido).

```
#!/usr/bin/env python3
# listas02.py
a=[0,1,2,3,4]
print(a)      # [0, 1, 2, 3, 4]
print(a[1])   # 1
print(a[0:2]) # [0,1]
print(a[3:])  # [3,4]
print(a[-1])  # 4
print(a[:-1]) # [0, 1, 2, 3]
print(a[:-2]) # [0, 1, 2]
```

La misma sintaxis se aplica a las cadenas

```
a="niño"
print(a[-1])  # o
```

- `append()` añade un elemento al final de la lista
- `insert()` inserta un elemento en la posición indicada

```
#!/usr/bin/env python3
# listas03.py
a=['cero', 'uno']
a.append('dos')
print(a)      # ['cero', 'uno', 'dos']
a.insert(1, 'cero cinco')
print(a)      # ['cero', 'cero cinco', 'uno', 'dos']
```

- `index()` busca en la lista un elemento y devuelve el índice de la primera aparición del elemento en la lista. Si no aparece se eleva una excepción.
- El operador `in` devuelve *true* si un elemento aparece en la lista, y *false* en caso contrario.

```
>>> lista=['cero', 'uno', 'dos']
>>> print(lista)
['cero', 'uno', 'dos']
>>> lista.index('uno')
1
>>> 'doce' in lista
False
>>> print(lista.index('doce'))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 'doce' is not in list
```

- `remove()` elimina la primera aparición de un elemento en la lista. Si no aparece, eleva una excepción.
- `pop()` devuelve el último elemento de la lista, y lo elimina. (Pila)
- `pop(0)` devuelve el primer elemento de la lista, y lo elimina. (Cola)

```
#!/usr/bin/env python3
# listas04.py
a = ['rojo', 'verde', 'azul', 'negro']
a.remove('negro')
print(a)    # ['rojo', 'verde', 'azul']
print(a.pop())  # azul
print(a)    # ['rojo', 'verde']
print(a.pop(0))  # rojo
print(a)    # ['verde']
```

- El operador `+` concatena dos listas, devolviendo una nueva lista
- El operador `*` concatena repetitivamente una lista a sí misma

```
#!/usr/bin/env python3
# listas05.py
a = ['rojo', 'verde' ]
a = a + ['azul']
print(a)      # ['rojo', 'verde', 'azul']
a += ['negro']
print(a)      # ['rojo', 'verde', 'azul', 'negro']
a = ['x', 'y']
a = a * 3
print(a)      # ['x', 'y', 'x', 'y', 'x', 'y']
```

Funciones, métodos y operadores

El lenguaje python:

- Emplea el modelo de programación imperativa convencional

  Por tanto usa funciones, cuya sintaxis es

  `funcion(objeto)`

- Emplea el modelo de programación orientada a objetos

  Por tanto usa métodos, cuya sintaxis es

  `objeto.metodo()`

- Es de muy alto nivel, cuenta con operadores con funcionalidad avanzada

  La sintaxis de un operador es

  `elemento1 operador elemento2`

Esto puede provocar confusión, es fácil equivocarse e intentar usar una función como un método o viceversa

Este script emplea la función `len()`, el método `pop()` y el operador `in`

```
#!/usr/bin/env python3
# funcion_operador_metodo.py

lista=["rojo","amarillo","verde"]
print(len(lista))         # 3
print("blanco" in lista)  # False
print(lista.pop())        # verde
print(lista)              # ['rojo', 'amarillo']
```

Inversión de una lista

- El método `reverse()` invierte las posiciones de los elementos en una lista.

No devuelve nada, simplemente altera la lista sobre la que se aplican.

```
>>> a=['sota', 'caballo', 'rey']
>>> a.reverse()
>>> print(a)
['rey', 'caballo', 'sota']
```

Errores típicos:

```
a = reverse(a)   # ¡Mal! Esta función no existe
a = a.reverse()  # Ahora valdría None
```

El primer error lo indica el intérprete. El segundo es más peligroso

Ordenar una lista

- La función `sorted()` devuelve una lista ordenada (no la modifica)
- El método `sort()` ordena una lista (Modifica la lista, devuelve *None*)

Ambas admiten personalizar la ordenación de elementos complejos, pasando como argumento una función que devuelva la parte del elemento a usar como clave. Esta función argumento debe llamarse *key*

```python
#!/usr/bin/env python3
# ordenar01.py
mi_lista=[ "gamma", "alfa", "beta"]

print( sorted(mi_lista) )  # alfa, beta, gamma
print( mi_lista  )         # gamma, alfa, beta. No ha cambiado.

print( mi_lista.sort() )   # Devuelve 'None'
print( mi_lista )          # alfa, beta, gamma. La ha ordenado
```

```python
#!/usr/bin/env python3
# ordenar02.py
mi_lista=[  ['IV',4] , ['XX',20], ['III',3]  ]

def clave_lista(lista):
    return lista[1]

mi_lista.sort(key = clave_lista)
print( mi_lista )
```

Split, join

Es muy frecuente trocear una cadena para formar en un lista (split) y concatenar los elementos de una lista para formar una cadena (join)

```python
#!/usr/bin/env python3
# split_join.py
mi_cadena="esto es una prueba"
print(mi_cadena.split()) # ['esto', 'es', 'una', 'prueba']
print("esto-tambien".split("-"))    # ['esto', 'tambien']

mi_lista=["as","dos","tres"]
#print(mi_lista.join()) # ¡ERROR! Parecería lógico que join()
                        # fuera un método del tipo lista. Pero no
                        # lo es.
print("".join(mi_lista)) # Es un método del tipo string, hay
                         # que invocarlo desde una cadena cualquiera,
                         # que será el separador.
                         # Devuelve  "asdostres"

print(",".join(mi_lista)) # Devuelve "as,dos,tres"
```

Otros métodos de los objetos string

```python
#!/usr/bin/env python3
# cadenas.py
print("hola mundo".upper())  # HOLA MUNDO
print("HOLA MUNDO".lower())  # hola mundo

# Estos métodos devuelven una cadena,
# sin modificar la cadena original
a="prueba"
print(a.upper())            # PRUEBA
```

```
print(a)                    # prueba

# find() indica la posición de una subcadena
print("buscando una subcadena".find("una")) # 9
print("buscando una subcadena".find("nohay")) # -1

# strip() devuelve una copia de la cadena quitando
# espacios a derecha e izda, retornos de carro, etc
print("    hola  \n".strip())  # 'hola'

print("te digo que no".replace("digo","diego"))
       # imprime "te diego que no"
```

Nombres de objeto

Con frecuencia hablamos de *variables*, porque es el término tradicional en programación. Pero Python no tiene *variables*, sino *nombres*. Son referencias a objetos

```
#!/usr/bin/env python3
# nombres.py
x=['uno']
y=x      # y apunta al mismo objeto
print(x)  # ['uno']
print(y)  # ['uno']

x=['dos']  # x apunta a un nuevo objeto

print(x)  # ['dos']  # El objeto nuevo
print(y)  # ['uno']  # El objeto antiguo

x=['uno']
y=x      # y apunta al mismo objeto
x.append('dos')   # modificamos el objeto
print(x)  # ['uno','dos']  # el objeto modificado
print(y)  # ['uno','dos']  # el mismo objeto, modificado
```

11.9.3. Diccionarios

Diccionarios

- Es un conjunto *desordenado* de elementos
- Cada elemento del diccionario es un par clave-valor.
- Se pueden obtener valores a partir de la clave, pero no al revés.
- Longitud variable
- Hace las veces de los *registros* en otros lenguajes
- Atención: Se declaran con `{}`, se refieren con `[]`

- Asignar valor a una clave existente reemplaza el antiguo
- Una clave de tipo cadena es sensible a mayúsculas/minúsculas
- Pueden añadirse entradas nuevas al diccionario
- Los diccionarios se mantienen desordenados
- Los valores de un diccionario pueden ser de cualquier tipo
- Las claves pueden ser enteros, cadenas y algún otro tipo
- Pueden borrarse un elemento del diccionario con `del`
- Pueden borrarse todos los elementos del diccionario con `clear()`

Otras operaciones con diccionarios:

- `len(d)` devuelve el número de elementos de `d`
- `d.has_key(k)` devuelve 1 si existe la clave `k` en `d`, 0 en caso contrario
- `k in d` equivale a: `d.has_key(k)`
- `d.items()` devuelve la lista de elementos de `d` (pares clave:valor)
- `d.keys()` devuelve la lista de claves de `d`

```
#!/usr/bin/env python3
# diccionarios.py
pais={'de': 'Alemania', 'fr': 'Francia', 'es': 'España'}
print(pais["fr"])

extension={}
extension['py']='python'
extension['txt']='texto plano'
extension['mp3']='MPEG layer 3'

for x in pais.keys():
   print(x, pais[x])

del pais['fr']   # Borramos francia
print(len(pais))  # Quedan 2 paises
print('es' in pais)   # True
pais['es']="Spain"  # modificamos un elemento
pais.clear()  # Borramos todas las claves
```

```
#!/usr/bin/env python3
# diccionarios02.py

diccionario={"juan": ["empanada"] ,
             "maria": ["refrescos","vino"]}

diccionario["luis" ] = ["patatas fritas","platos plastico"]
```

```
diccionario["luis"].append("vasos plastico")

claves = diccionario.keys() # Devuelve un tipo dict_keys
claves = list(claves)     # Lo convertimos en lista para ordenarlo
claves.sort()
for clave in claves:
     print(clave, diccionario[clave])
```

Resultado de la ejecución:

```
juan ['empanada']
luis ['patatas fritas', 'platos plastico', 'vasos plastico']
maria ['refrescos', 'vino']
```

Acceso a las claves mediante el operador in

Una forma alternativa de obtener las claves de un diccionario:

```
for clave in d:
    print(clave)
```

- Esto es más eficiente que emplear el método `keys()`
- Es aplicable a listas y tuplas
- Aunque en ocasiones seguiremos necesitando el método `keys()`

  ```
  claves=list(diccionario.keys())
  claves.sort()
  ```

11.9.4. Tuplas

Tuplas

- Tipo predefinido de Python para una lista inmutable
- Se define de la misma manera que las listas, pero con los elementos entre paréntesis
- Las tuplas no tienen métodos: no se pueden añadir elementos, ni cambiarlos, ni buscar con `index()`
- Sí puede comprobarse la existencia con el operador `in`.

```
>>> t = ("a", "b", "blablabla", "z", "example")
>>> t[0]
'a'
>>> 'a' in t
True
>>> t[0] =  "b"
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
```

Utilidad de las tuplas:

- Son más rápidas que las listas
- Pueden ser una clave de un diccionario (no así las listas)
- Se usan en el formateo de cadenas

`tuple(li)` devuelve una tupla con los elementos de la lista `li`
`list(t)` devuelve una lista con los elementos de la tupla `t`

Asignaciones múltiples

- Pueden hacerse también tuplas de nombres de objetos

```
>>> v = ('a', 'b', 'e')
>>> (x, y, z) = v
>>> x
'a'
```

11.10. Sentencias de control

If

```
#!/usr/bin/env python3
# if01.py
x = True
if x :
     print('verdadero')
else:
     print('falso')
```

Nótese como el carácter `:` introduce cada bloque de sentencias.
Si en una rama queremos poner la sentencia nula: `pass`
No confundir con el valor nulo: `None`

```python
#!/usr/bin/env python3
# if02.py

x = int(input("Escribe un entero: "))
if x < 0:
     x = 0
     print('Valor negativo, modificado a cero')
elif x == 0:
     print('Cero')
elif x == 1:
     print('Uno')
else:
     print('Más de uno')
```

La sentencia `case` aparece en la versión 3.10 (Octubre 2021)

For

En los lenguajes *convencionales*, la cláusula *for* sirve para que un entero recorra una serie de valores.

En python es diferente: recorre un objeto iterable, como una lista o una tupla. Por cada elemento del iterable, ejecuta el bloque de código

```python
lista = ["sota","caballo","rey"]
for x in lista:
    print(x) # Imprime el elemento y nueva línea
```

Resultado:

```
sota
caballo
rey
```

Si necesitamos un bucle *convencional* podemos emplear la función *range()*

```python
#!/usr/bin/env python3
# rangos01.py
rango = range(3)
for x in rango:
    print(x, end='')
```

Resultado:

```
012
```

Observa que la función *print*

- Por omisión añade un carácter *nueva línea* tras cada argumento
- Para evitarlo, podemos añadir el parámetro *end* indicando qué añadir. Por ejemplo una cadena vacía, como en este caso

A range() le podemos pasar

- Un elemento: el final del rango
- Dos elementos: principio y final
- Tres elementos: principio, final e incremento

Por omisión, el principio es 0 y el incremento es +1

Esta función devuelve un objeto de tipo *range*. Si es necesario, podemos convertirlo en lista con la función *list*

```
#!/usr/bin/env python3
# rangos02.py
print(range(2,5))            # range(2,5)
print(list(range(2,5)))      # 2, 3, 4
print(list(range(2,-2,-1)))  # 2, 1, 0, -1
```

No deberíamos usar range para los bucles a menos que sea imprescindible. No es idiomático en python, añade complejidad innecesaria.

No hagas bucles *al estilo Pascal*

```
#!/usr/bin/env python3
# for.py

lista=["sota","caballo","rey"]
# ¡¡NO HAGAS ESTO!!
for i in range(len(lista)):
    print(lista[i])

# Lo idiomático en python es
for x in lista:
    print(x)
```

While

```
>>> a=0
>>> while a<10:
...     print(a, end='')
...     a+=1
...
0 1 2 3 4 5 6 7 8 9
```

`break` sale de un bucle. (Aunque según los principios de la *programación estructurada*, `break` no debería usarse nunca. Empléalo solo si estás muy seguro de lo que haces)

```
#!/usr/bin/env python3
# while01.py
a=10
while a > 0:
    print(a, end='')
    a-=1
```

equivale a

```
#!/usr/bin/env python3
# while02.py
a=10
while True:
    print(a, end='')
    if a==1:
        break
    a-=1
```

11.11. format

format

Las cadenas cuentan con el método `format()`

- Dentro de una cadena, podemos indicar, entre llaves, qué campos se mostrarán y con qué formato.

  Format tiene un microlenguaje para esto

- Los argumentos de `format()` serán los campos

Ejemplo: Indicar qué campo mostrar, a partir del ordinal

```
#!/usr/bin/env python3

name="Juan"
surname="García"
print("Se llama {0} y se apellida {1}".format(name,surname))
print("Se llama {} y se apellida {}".format(name,surname))

persona=["Juan","García"]
print("Se llama {0[0]} y se apellida {0[1]}".format(persona))

persona={"name":"Juan", "surname":"García"}
print("Se llama {0[name]} y se apellida {0[surname]}".format(persona))
```

Resultado:

```
Se llama Juan y se apellida García
Se llama Juan y se apellida García
Se llama Juan y se apellida García
Se llama Juan y se apellida García
```

Después de indicar qué campo mostrar, separado por el carácter dos puntos, podemos especificar cuántos caracteres debe ocupar la salida, y si estará alineada a la derecha (signo de mayor), a la izquierda (signo de menor o ningún signo) o al centro (acento circunflejo)

Ejemplo: mostrar una palabra, ocupando siempre 12 caracteres

```
#!/usr/bin/env python3

print("{0:>12}{1:>12}".format("sota","caballo"))
print("{0:<12}{1:<12}".format("sota","caballo"))
print("{0:12}{1:12}".format("sota","caballo"))
print("{0:^12}{1:^12}".format("sota","caballo"))
```

Resultado:

```
        sota     caballo
sota        caballo
sota        caballo
    sota      caballo
```

- Si solo hay un campo, podemos omitir el 0 a la izquierda del carácter dos puntos
- Con el carácter `d` podemos indicar que el campo contiene un número entero. En este caso, la alineación por omisión es a la derecha
- Con el carácter `f` indicamos que el campo es un número real

  Podemos especificar cuántos decimales representar. Por ejemplo 4: `.4f`

```
print("{:<6d} metros".format(592))
print("{:>6d} metros".format(592))
print("{0:6d} metros".format(592))
print("Pi vale {:.4f}".format(3.14159265358979))
```

Resultado:

```
592    metros
   592 metros
   592 metros
Pi vale 3.1416
```

Naturalmente, la cadena no tiene por qué ser una constante, puede ser una variable

```
#!/usr/bin/env python3

x=12.3
y=0.345
z=34000
template="{:8},{:8},{:8}"
msg=template.format(x,y,z)
print(msg) #    12.3,   0.345,   34000
```

11.12. Funciones

Funciones

```
#!/usr/bin/env python3
# grados.py
def a_centigrado(x):
    """Convierte grados farenheit en grados centígrados."""
    return (x-32)*(5/9.0)


def a_farenheit(x):
    """Convierte grados centígrados en grados farenheit."""
    return (x*1.8)+32
```

Los nombres de objeto declarados fuera de una función son globales, y los declarados dentro, locales

```
#!/usr/bin/env python3
# ambito01.py
a=3
def f():
    b=4
    print(a) # 3
    print(b) # 4
    return
f()
print(a)     # 3
print(b)     # ¡Error! B es un objeto local
```

- Algunas metodologías establecen que los objetos globales deben usarse lo mínimo posible. Otras los prohiben por completo
- Los objetos globales pueden leerse dentro (y fuera) de la función.
- Los objetos locales, declarados dentro de una función, son invisibles fuera de ella

Supongamos que intentamos modificar el objeto global de esta forma

```
#!/usr/bin/env python3
# ambito02.py
a=3
def f():
    a=0
    print(a) # 0
    return
f()
print(a)     # 3 . No se ha modificado
```

No podemos modificar el objeto global sin más, lo que sucede es que python crea un nuevo objeto local, con el mismo nombre que el global. El objeto local hace que el objeto global sea invisible, el local *tapa* al global

Las modificaciones similares a esta siempre generarán un error

```
#!/usr/bin/env python3
# ambito03.py
c=3
def f():
    c=c-1   # ERROR: la variable global ya no es visible y la
            # local aún no está definida
    return

f()
```

En cuanto el intérprete procesa el nombre del objeto a la izquierda de un signo igual, crea un objeto local que aún no está definido, pero que hace invisible al objeto global

Para poder modificar un objeto global, es necesario declararlo con la sentencia `global`

```
#!/usr/bin/env python3
# ambito04.py
c=3
def f():
    global c
    c=0       # Esto modifica el objeto global
    print(c)  # 0
    return
f()
print(c)       #0
```

La sentencia global evita que al declarar un objeto en una función, se cree un nuevo objeto con el mismo nombre pero de ámbito local. Por tanto permite modificar el objeto global

En muchos lenguajes, para hacer que una variable sea global, la declararíamos `global` en la la *zona global* del código, haríamos un código similar a este, pero que en python es incorrecto

```
#!/usr/bin/env python3
# ambito05
global c   #ERROR, esto no sirve de nada
c=3
def f():
    c=0      # Esto es un objeto local
    print(c) # 0
    return
f()
print(c)      #3  el global no ha cambiado
```

- Observa que en python se usa la sentencia `global` en la función local que vaya a modificar el objeto

- Dicho de otro modo: la sentencia `global` no significa *haz que este objeto sea global*, sino *haz que este objeto global pueda ser modificado aquí*
- Seguramente resultaría más intutivo si la sentencia `global` tuviera un nombre distinto. Tal vez `global-write` o `GlobalModify`

Los objetos mutables (listas, diccionarios...) declarados dentro de una función también son locales, en este aspecto se comportan igual que los objetos inmutables

```
#!/usr/bin/env python3
# ambito06.py
l= ["uno","dos"]
def f():
    l=["cuatro"]  # nuevo objeto mutable, local

print(l) #  ["uno","dos"]
f()
print(l) #  ["uno","dos"]
```

Hay una diferencia entre los objetos mutables y los inmutables.
Como hemos visto

- Los objetos inmutables globales se pueden leer localmente

- Para poder modificar un objeto inmutable global, es necesario usar la sentencia `global`

  Por tanto, un objeto global sin la sentencia `global` queda *protegido contra escritura*

Los objetos mutables globales no se pueden *proteger contra escritura* de esta manera

Un objeto mutable sí puede ser modificado en una función local, a pesar de no estar declarado `global`

```
#!/usr/bin/env python3
# ambito07.py
l= ["uno","dos"]
def f():
    l.pop()

print(l) #  ["uno","dos"]
f()
print(l) #  ["uno"]  . El objeto mutable fue modificado por la función
```

El objeto mutable puede ser modificado a través de sus métodos. (No debo pensar que la ausencia de la sentencia `global` hace que el objeto esté en modo *solo lectura*)

```
#!/usr/bin/env python3
# ambito08.py
l= ["uno","dos"]
def f():
    l=["uno"]
print(l) #  ["uno","dos"]
f()
print(l) #  ["uno","dos"]  No cambia .
```

En el caso de que la modificación se haga redefiniendo el objeto (no mediante métodos), como ya sabemos, implica la declaración implícita de un objeto nuevo, local, que oculta al objeto global. Por tanto, el objeto global no es modificado

Si al ejemplo anterior le añadimos `global` de esta manera, como cabría esperar, permite modificar el objeto global

```python
#!/usr/bin/env python3
# ambito09.py
l= ["uno","dos"]
def f():
    global l
    l=["uno"]
print(l) #  ["uno","dos"]
f()
print(l) #  ["uno"]  Ha cambiado.
```

Resumen:

- Los objetos declarados fuera de una función son globales
- Los objetos declarados dentro de una función son locales
- Los objetos globales siempre se pueden leer dentro de una función
- Para modificar un objeto global dentro de una función
  - Si es inmutable, hay que usar `global` dentro de la función
  - Si es mutable
    - Para modificarlo mediante una asignación, hay que usar `global`
    - Para modificarlo mediante sus métodos, no es necesario usar `global`

En las llamadas a funciones

- Los objetos inmutables se pasan por valor. La función recibe una copia del valor, por lo que una posible modificación de la copia no altera el original
- Los objetos mutables se pasan por referencia. La función recibe una referencia al objeto original, una modificación del objeto en la función modifica el objeto original

```python
#!/usr/bin/env python3
# valor_referencia.py
def f(x,y):
    x=x-1
    y.pop()

def g():
```

```
    v=3
    l=["uno","dos"]
    f(v,l)
    print(v)  # 3  . La función creó copia local, no tocó el parámetro
    print(l)  # ['uno'] La función recibió el parámetro por referencia

g()
```

11.13. Cadenas de documentación

Cadenas de documentación

- No son obligatorias pero sí muy recomendables (varias herramientas hacen uso de ellas).
- La cadena de documentación de un objeto es su atributo `__doc__`
- En una sola línea para objetos sencillos, en varias para el resto de los casos.
- Entre triples comillas-dobles (incluso si ocupan una línea).
- Si hay varias líneas:
    - La primera línea debe ser una resumen breve del propósito del objeto. Debe empezar con mayúscula y acabar con un punto
    - Una línea en blanco debe separar la primera línea del resto
    - Las siguientes líneas deberían empezar justo debajo de la primera comilla doble de la primera línea

De una sola línea:

```
def kos_root():
    """Return the pathname of the KOS root directory."""
    global _kos_root
    ...
```

De varias:

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)

    """
    if imag == 0.0 and real == 0.0: return complex_zero
```

Documentando el código (tipo Javadoc)

- Permite documentar el código -generalmente las funciones- dentro del propio código
- Genera la documentación del código en formatos legibles y navegables (HTML, PDF...)
- Se basa en un lenguaje de marcado simple
- PERO... hay que mantener la documentación al día cuando se cambia el código

Ejemplo

```
def interseccion(m, b):
  """
  Devuelve la interseccion de la curva M{y=m*x+b} con el eje X.
  Se trata del punto en el que la curva cruza el eje X (M{y=0}).

  @type  m: número
  @param m: La pendiente de la curva
  @type  b: número
  @param b: La intersección con el eje Y

  @rtype:   número
  @return:  la interseccíoin con el eje X de la curva M{y=m*x+b}.
  """
  return -b/m
```

11.14. Ficheros

Ficheros

- `open(nombre_fichero,modo)` devuelve un objeto fichero.

  modo:

  - `w`: Escritura. Destruye contenido anterior
  - `r`: Lectura. Modo por defecto
  - `r+`: Lectura y escritura
  - `a`: Append

- `write(cadena)` escribe la cadena en el fichero. Solo escribe cadenas, para otros tipos, es necesario pasar a texto o usar librerías como *json* o *pickle*
- `read()` devuelve una cadena con todo el contenido del fichero

- `readlines()` devuelve una lista donde cada elemento es una línea del fichero
- `close()` cierra el fichero

```
#!/usr/bin/env python3
# ficheros01.py
lista=['sota','caballo','rey']
fichero=open('prueba.txt','w')
for x in lista:
    fichero.write(x+"\n")
fichero.close()

fichero=open('prueba.txt','r')
mi_cadena=fichero.read()          # Una sola cadena de 3 líneas
fichero.seek(0)                   # vuelvo al princio del fichero

lista_de_cadenas=fichero.readlines()  # ahora cada elemento incluye \n
fichero.seek(0)

for linea in fichero.readlines():
    print(linea, end='')

fichero.close()
```

Los métodos *read()* y *readlines()* crean una copia completa del fichero en memoria.

Para ficheros muy grandes es más eficiente trabajar línea a línea

```
fichero=open('prueba.txt','r')
for linea in fichero:
    print(linea, end='')
fichero.close()
```

No se deben mezclar estas dos maneras de acceder a un fichero

11.15. Excepciones

Excepciones

- Un programa sintácticamente correcto puede dar errores de ejecución

```
#!/usr/bin/env python3
# excepcion01.py
while 1:
    x=int(input("Introduce un nº" ))
    print(x)
```

Si el usuario no escribe un número sino por ejemplo *a*

```
  File "./excepcion01.py", line 4, in <module>
    x=int(input("Introduce un nº"))
ValueError: invalid literal for int() with base 10: 'a'
```

- Definimos una acción para determinada excepción

```python
#!/usr/bin/env python3
# excepcion02.py
while 1:
    try:
        x=int(input("Introduce un nº:"))
        print(x)
    except ValueError:
        print("Número incorrecto")
```

- Se puede indicar una acción que se ejecute sea cual sea la excepción pero es *muy* desaconsejable (enmascara otros errores)
- El programador puede levantar excepciones

```python
#!/usr/bin/env python3
# excepcion03.py
try:
    x=int(input("Introduce un nº:"))
    print(x)
except :     # para cualquier excepción
    raise Exception("Número incorrecto")
```

11.16. Fechas

Fechas

- En Unix, y por tanto en internet, es habitual expresar la fecha y la hora como el número de segundos transcurridos desde el *epoch* (1 de enero de 1970 a las 00:00)
- En python, la librería *time* ofrece el método *time* que acepta una fecha en segundos desde el *epoch*, y la devuelve en un *struct_time*
    - Si invocamos a este método sin argumentos, nos devuelve la fecha actual
- Para conocer la hora Unix de modificación de un fichero

  `os.path.getmtime("/nombre/de/fichero")`

```python
#!/usr/bin/env python3
# fechas.py
import time,os

t_unix = time.time()       # Segundos desde el epoch
t = time.gmtime(t_unix)    #  Fecha en un struct_time
print(t)
print(t.tm_year)
print(t.tm_mon)
print(t.tm_mday)

print(os.path.getmtime("/etc/hosts"))
```

Resultado:

```
time.struct_time(tm_year=2021, tm_mon=11, tm_mday=24, tm_hour=16, tm_min=48,
tm_sec=3, tm_wday=2, tm_yday=328, tm_isdst=0)
2021
11
24
1637418845.3172204
```

11.17. Cadenas binarias

Cadenas binarias

- Hasta la década de 1990, en prácticamente cualquier ámbito de la informática, un carácter equivalía a un byte. La codificación de caracteres no ingleses era compleja, con distintas normas incompatibles
- En las décadas de 1990, 2000 se emplea la norma ISO 8859. Para los lenguajes de Europa Occidental, ISO 8859-1, también llamada *latin 1*
- En la década de 2000 se empieza a usar la norma Unicode, concretamente con la codificación UTF-8
    - Es la codificación prácticamente universal en la actualidad, aunque no es raro encontrarse sistemas antiguos con normas anteriores
    - Permite representar cualquier lenguaje humano, incluyendo lenguajes extintos y lenguajes artificiales
    - UTF-8 es un superconjunto de ASCII, para cada caracter emplea entre 1 y 4 bytes

En python 3, todas las cadenas son unicode UTF-8 por omisión

- Si es necesario, también se pueden usar cadenas *al estilo antiguo*, llamadas tipo *bytes*[10]
- Podemos declarar una cadena de tipo *bytes* anteponiendo *b*

```
b"Esto es una cadena de tipo byte"
```

- Para detectar el tipo de una cadena usamos la función *type*

```
print(type('holamundo'))    #   <class 'str'>
print(type(b'holamundo'))   #   <class 'bytes'>
```

Conversión entre cadenas de 8 bits y cadenas unicode en python 3

- De string (unicode) a byte (8 bits)

```
>>> "españa".encode('utf-8')
b'espa\xc3\xb1a'
```

- De bytes (8 bits) a string (unicode)

```
>>> b=b'123'
>>> print(b)
b'123'
>>> s=b.decode()
>>> print(s)
123
```

12. Librerías de Python

12.1. Librería sys

libreria sys

- Argumentos de linea de órdenes

`sys.argv` devuelve una lista con los argumentos pasados al script python desde la shell

[10] Al contrario que en python 2, donde por omisión era de tipo byte aunque también existía el tipo unicode

```python
#!/usr/bin/env python3
# argumentos.py
import sys
print(sys.argv[:])
```

```
koji@mazinger:~$ ./argumentos.py un_argumento otro_argumento
['./argumentos.py', 'un_argumento', 'otro_argumento']
```

(El argumento cero es el nombre del programa)
Escribir en stderr

```python
#!/usr/bin/env python3
# error_estandar.py
import sys
sys.stderr.write('Error: \n')
```

Leer desde stdin, escribir en stdout

```python
#!/usr/bin/env python3
# cat.py
import sys
for linea in sys.stdin.readlines():
    sys.stdout.write(linea)
```

12.2. Librería subprocess

subprocess

- `subprocess.check_output()` permite ejecutar una orden de la shell en un subproceso externo
- Aunque puede ser muy útil, el script deja de ser portable entre sistemas operativos diferentes
- Su primer argumento es una lista con los argumentos de la orden a ejecutar
- Devuelve la salida estándar del subproceso pero como *bytes* que habrá que convertir a cadena (unicode) con *decode*
  - En caso contrario obtendremos un error

    `TypeError: a bytes-like object is required, not 'str'`
- Si se produce algún error, eleva la excepción `subprocess.CalledProcessError`

Los comandos de la shell normalmente adaptan su salida al número de filas y columnas que tenga nuestro terminal

- Estas dimensiones se obtienen desde las variables de entorno `COLUMNS` y `LINES`
- Por si acaso estamos usando un terminal muy pequeño, es conveniente indicarle a la shell que ejecutamos con *subprocess* que considere un terminal de tamaño *normal*. Para ello, damos valor a estas variables

```
os.environ["COLUMNS"]="80"
os.environ["LINES"]="150"
```

```python
#!/usr/bin/env python3
# subproceso01.py
import subprocess,sys,os
os.environ["COLUMNS"]="80"
os.environ["LINES"]="150"

mandato="ps -ef"
mandato_troceado=mandato.split()
try:
    salida=subprocess.check_output(mandato_troceado)
except subprocess.CalledProcessError:
    sys.stderr.write("La orden ha producido un error\n")
    raise SystemExit
salida=salida.decode("utf-8")   # De bytes a string
lineas=salida.split("\n")  # troceamos la salida línea a línea
lineas.pop(0)      # quitamos la primera línea, la cabecera del ps
for linea in lineas:
    if len(linea) != 0:
        campos_linea=linea.split()
        usuario = campos_linea[0]
        proceso =  campos_linea[7]
        print("Usuario:{} Proceso:{}".format(usuario,proceso))
```

Si necesitamos más control sobre los posibles errores

- Para redirigir la salida de error del subproceso a la salida estándar, pasamos el parámetro `stderr=subprocess.STDOUT`
- El atributo `returncode` de `CalledProcessError` contiene el código del error

```
#!/usr/bin/env python3
# subproceso02.py
import subprocess,sys,os

os.environ["COLUMNS"]="80"
os.environ["LINES"]="150"

mandato="ls inexistente"
mandato_troceado=mandato.split()
try:
    salida=subprocess.check_output(mandato_troceado,
                  stderr=subprocess.STDOUT)
except subprocess.CalledProcessError as e:
    plantilla = "{}Código:{}\n"
    msj_subproceso = e.output
    msj_subproceso = msj_subproceso.decode() # De bytes a string
    codigo_subproceso = e.returncode
    msj = plantilla.format(msj_subproceso, codigo_subproceso)
    sys.stderr.write(msj)
```

12.3. Librerías os, shutil

os.path

- Las funciones os.path.join() y os.path.split() unen y separan nombres de fichero con directorios
  - Son compatibles con cualquier S.O.
  - No importa si el path acaba en barra o no
- `os.path.exists()` devuelve un boolean indicando si un fichero existe

```
#!/usr/bin/env python3
# path01.py
import os
ejemplo=os.path.join("/etc/apt","sources.list")
print(ejemplo)   # /etc/apt/sources.list
print(os.path.split(ejemplo))  # ('/etc/apt', 'sources.list')

print(os.path.exists(ejemplo))  # True
print(os.path.exists("/usr/local/noexiste"))  # False
```

Enlazar, borrar

```
#!/usr/bin/env python
# enlazar_borrar.py
import os
if not os.path.exists("/tmp/test"):
    os.mkdir("/tmp/test")
```

```
os.chdir("/tmp/test")                    # cd /tmp/aa
os.symlink("/etc/hosts","enlace_hosts") # crea enlace simbólico
os.remove("enlace_hosts")                # borra el fichero
os.rmdir("/tmp/test")                    # borra directorio (vacio)
```

copiar, copiar y borrar recursivamente

```
#!/usr/bin/env python3
# recursivo.py
import shutil,os
shutil.copytree("/home/koji/.gnome","/tmp/probando")
    # copia recursivamente. El destino no debe existir

shutil.copy("/etc/hosts","/tmp/probando")
    # copia 1 fichero (como el cp de bash)

shutil.move("/tmp/probando/hosts","/tmp/probando/mi_hosts")

shutil.rmtree("/tmp/probando")   # borra directorio lleno
```

os.walk

- Recorre recursivamente un directorio
- Por cada directorio devuelve una 3-tupla
  - Directorio
  - Subdirectorios
  - Ficheros

```
#!/usr/bin/env python3
# walk.py
import os
directorio_inicial=os.getcwd() # current working directory
os.chdir("/tmp/musica")        # cd

for x in os.walk("."):
    print(x)

os.chdir(directorio_inicial)
```

```
/tmp/musica
|-- listado.txt
|-- jazz
`-- pop
    |-- sabina
    |   |-- pirata_cojo.mp3
    |   `-- princesa.mp3
```

```
    `-- serrat
        |-- curro_el_palmo.mp3
        `-- penelope.mp3


('.', ['jazz', 'pop'], ['listado.txt'])
('./jazz', [], [])
('./pop', ['serrat', 'sabina'], [])
('./pop/serrat', [], ['curro_el_palmo.mp3', 'penelope.mp3'])
('./pop/sabina', [], ['princesa.mp3', 'pirata_cojo.mp3'])
```

12.4. Librerías pickle: Persistencia

Persistencia

Persistencia en Python:

Una librería de persistencia permite *serializar objetos*, esto es, convertirlos en una secuencia binaria o de texto que se pueda transmitir fuera de python, almacenar en disco, base de datos, etc

- Librería *Pickle*
  - Formato propio de Python, binario, muy eficiente
  - Soporta cualquire clase, predefinida en Python o por el usuario
- Librería *json*
  - Formato estándar de internet, modo texto, legible
  - Solo soporta cadenas, números, booleanos, diccionarios y listas

Persistencia con pickle

```python
#!/usr/bin/env python3
# conserva.py
import pickle

cp={28:'madrid',8:'barcelona',33:'asturias'}
fich=open('prueba.pick','wb')  # Apertura modo binario
pickle.dump(cp,fich)
fich.close()

fich=open('prueba.pick','rb')  # Lectura modo binario
codigos_postales=pickle.load(fich)
fich.close()

for x in codigos_postales.keys():
    print(x,codigos_postales[x])
```

Persistencia con json

```
#!/usr/bin/env python3
# ejemplo_json.py
import json

# Objeto python ordinario
diccionario = { 'MAD':'Madrid Barajas', 'MCV':'Madrid Cuatro Vientos'}
print(type(diccionario))   # <class 'dict'>

# Lo convertimos en cadena json
cadena_json = json.dumps(diccionario)
print(type(cadena_json))   # <class 'str'>
print(cadena_json)
# {"MAD": "Madrid Barajas", "MCV": "Madrid Cuatro Vientos"}

# Volvemos a convertir la cadena en objeto python
diccionario = json.loads(cadena_json)
print(type(diccionario))  # <class 'dict'>
```

12.5. Acceso a las variables de entorno

Variables de entorno

```
#!/usr/bin/env python3
# entorno.py
import os, sys
mi_variable=os.getenv("MI_VARIABLE")
if mi_variable==None:
        msg="ERROR: variable de entorno MI_VARIABLE no definida"
        sys.stderr.write(msg+'\n')
        raise SystemExit
```

Atención: Cuando la shell crea un proceso (p.e. el intérprete de python), puede no pasarle todas las variables de entorno. Por tanto, las variables visibles desde la shell serán distintas a las visibles desde python

12.6. Módulos

Módulos

Un módulo es un fichero que contiene definiciones y sentencias, que pueden ser usados desde otro fichero

mi_modulo.py

```
#!/usr/bin/env python3
a=3
def f(x):
    return x+1
```

test.py

```
#!/usr/bin/env python3
import mi_modulo

print(mi_modulo.a)     # 3
print(mi_modulo.f(0))  # 1
```

También se pueden importar los objetos por separado, de forma que luego se puede usar sin indicar explícitamente el módulo

```
#!/usr/bin/env python3
from mi_modulo import f
from mi_modulo import a

print(f(0))  # 1
print(a)     # 3
```

Es posible importar todos los objetos de un módulo

```
#!/usr/bin/env python3
from mi_modulo import *

print(f(0))  # 1
print(a)     # 3
```

Pero esto es una mala práctica, porque cuando el número de módulos aumenta, es difícil saber en qué módulo está cada objeto

Búsqueda de los módulos

El intérprete busca los módulos en el siguiente orden

1. En el directorio del script
2. En cada directorio indicado en la variable de entorno PYTHONPATH
3. En el directorio por omisión
   - En Unix y Linux suele estar en `/usr/lib`
     Por ejemplo
     `/usr/lib/python3.9`

Ficheros .pyc

Cuando se importa un módulo, si el intérprete tiene permisos, guarda en el mismo directorio un fichero con extensión .pyc que contiene el script compilado en bytecodes

- Este fichero ahorra tiempo la segunda vez que se ejecuta el módulo
- No es dependiente de la arquitectura pero sí de la versión exacta del intérprete. Si no existe o no es adecuado, se genera uno nuevo automáticamente
- Permite borrar el fuente .py si no queremos distribuirlo

Objetos en módulos

- Usar objetos globales es peligroso, muchas metodologías lo prohíben
- Pero usar algún objeto global, en un módulo compartido por otros módulos, en algunas ocasiones puede ser una práctica aceptable y conveniente

mis_globales.py

```
#!/usr/bin/env python3
a=3
```

modulo1.py

```
#!/usr/bin/env python3
import mis_globales
def f():
    return mis_globales.a
```

test.py

```
#!/usr/bin/env python3
import mis_globales, modulo1

print(modulo1.f()) #3
mis_globales.a=5
print(modulo1.f()) #5
```

Un fichero puede ser un script y un módulo simultáneamente, si añadimos una función main() y la sentencia

```
if __name__== "__main__":
    main()
```

De esta manera,

- Si el fichero se ejecuta como un script, el intérprete ejecutará la función main()
- Si el fichero se usa como módulo, importando sus funciones desde otro script, la función main() no será ejecutada

modulo1.py

```python
#!/usr/bin/env python3
def f(x):
    return x+1

def main():
    print("probando f", f(2))

if __name__== "__main__":
    main()
```

test.py

```python
#!/usr/bin/env python3
import modulo1
print(modulo1.f(0)) #1  No se ejecuta main()
```

12.7. optparse

optparse

`optparse` es una librería de python para procesar las opciones y argumentos con los que se llama a un script

```
  orden  opciones          argumentos
-------------------------------------------------
   cp     -r  -v      directorio1    directorio2
```

En un buen interfaz

- Las opciones deben ser opcionales. (El programa debe hacer algo útil sin ninguna opción)
- Las opciones proporcionan flexibilidad, pero demasiadas introducen complejidad
- Los parámetros fundamentales e imprescindibles deben ser argumentos

- Creamos una instancia de la clase `OptionParser`, pasando como argumento la cadena `usage` (que se mostrará al usuario cuando use mal el script, o cuando lo llame con `-h` o `--help`

```
usage = "Uso: %prog [opciones] origen destino"
parser = OptionParser(usage)
```

- Para añadir opciones invocamos al método `add_option`

```
    parser.add_option("-v", "--verbose",
                      action="store_true", dest="verbose",
                      help="Informe detallado")
```

- Invocamos a `parse_args()`, que devuelve las opciones ya procesadas y los argumentos

```
opciones, argumentos = parser.parse_args()
```

```
#!/usr/bin/env python3
# opciones01.py
import sys

from optparse import OptionParser
def main():
    usage = "%prog [opciones] origen destino"
    parser = OptionParser(usage)
    parser.add_option("-e", "--energy",
                      action="store", dest="energy",
                      help="Tipo de energia a usar en la copia ",
                      default='eolic')
    parser.add_option("-v", "--verbose",
                      action="store_true", dest="verbose",
                      help="Informe detallado")
    parser.add_option("-q", "--quiet",
                      action="store_false", dest="verbose",
                      help="Informe silencioso")
    opciones, argumentos = parser.parse_args()
```

```
    if len(argumentos) != 2:
        parser.error("Número incorrecto de argumentos")
    print("Tipo de energia:"+opciones.energy)
    print("Origen:",argumentos[0])
    print("Destino:",argumentos[1])
    if opciones.verbose:
        print("mucho blablabla ")

if __name__ == "__main__":
    main()
```

add_option

```
parser.add_option("-e", "--energy",
    action="store", dest="energy",
    help="Tipo de energia a usar en la copia ", default='eolic')
```

- Cada opción puede invocarse con una única letra (p.e. `-v`) o con una palabra (p.e. `--verbose`)
- Con el atributo *help* se construye el mensaje que se mostrará al usuario cuando invoque el programa con `-h` o `--help`
- La opción puede
  - Limitarse a activar o desactivar un flag.
    `action="store_true"` `action="store_false"`
  - Indicar un valor
    `action="store"`

  En ambos casos, la información se almacena en un atributo que se llama como indique el parámetro `dest`

```
parser.add_option("-d", "--discount",
                  action="store", dest="discount", type="float",
                  help="Coeficiente de descuento")
```

- Por omisión el tipo de la opción es un string, pero también acepta `string, int, long, choice, float` y `complex`

```
koji@mazinger:~/python$ ./cp_ecologico.py
Usage: cp_ecologico.py [opciones] origen destino

cp_ecologico.py: error: Número incorrecto de argumentos

koji@mazinger:~/python$ ./cp_ecologico.py -h
Usage: cp_ecologico.py [opciones] origen destino

Options:
  -h, --help            show this help message and exit
  -e ENERGY, --energy=ENERGY
                        Tipo de energia a usar en la copia
  -v, --verbose         Informe detallado
  -q, --quiet           Informe silencioso
  -d DISCOUNT, --discount=DISCOUNT
                        Coeficiente de descuento

koji@mazinger:~/python$ ./cp_ecologico.py -v  -d 0.15  mi_origen  mi_destino
Tipo de energia:eolic
Coeficiente de descuento:0.15
Origen: mi_origen
Destino: mi_destino
mucho blablabla
```

12.8. Bots de telegram

Bots de telegram

Telegram es una aplicación de mensajería instantánea muy similar a WhatsApp, con la que tiene las siguientes diferencias:

- Telegram es muy popular pero no tiene tantos usuarios como WhatsApp
- Telegram no solo tiene clientes para iOS y Android (como WhatsApp) , también tiene clientes independientes del telefono para Windows, Linux y macOS
- El cliente es software libre (el servidor,no)
- Fácilmente accesible mediante API

En python hay muchas librerías para manejar telegram, aquí veremos *telepot*

12.8.1. Creación de un bot de telegram

Para poder enviar y recibir mensajes, hay que crear un tipo de usuario especial llamado *bot*

Para crear un bot, tenemos que usar otro bot, llamado *BotFather*

1. Desde nuestro cliente de telegram, enviamos un mensaje con el texto `/newbot` al usuario `@BotFather`
2. Un diálogo nos irá preguntando todo lo necesario
3. Recibiremos un *token* que nos permitirá maneja el bot via API

De la misma forma, `@BotFather` nos permite cambiar el nombre de nuestro bot (comando `/newbot`), su foto (`/setuserpic`), eliminarlo (`deletebot`), etc

Para que un bot de telegram pueda enviar un mensaje a un usuario, hace falta:

1. Que el usuario le envíe al bot un mensaje cualquiera (un mensaje en la vida es suficiente)
2. Que el usuario facilite al programador del bot su propio `id` de usuario

- Es un número de unos 9 dígitos, no confundir con el nombre de usuario (p.e. `@JuanPerez`)
- El usuario puede averiguar su propio `id` enviando un mensaje cualquiera al bot `@userinfobot`

12.8.2. Uso de la librería telepot

Uso de la librería telepot

Para instalar telepot

- Si tenemos privilegios de root y queremos instalarlo para todos los usuarios del sistema, ejecutamos desde la shell

  ```
  pip3 install telepot
  ```

- En otro caso, podemos instalarlo solo para nuestro usuario:

  ```
  pip3 install --user telepot
  ```

Enviar un mensaje a un usuario

```python
#!/usr/bin/env python3

import telepot

TOKEN = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
bot = telepot.Bot(TOKEN)

def main():
    id_usuario = "999999999"
    bot.sendMessage(id_usuario, "Hola")

    return

if __name__ == "__main__":
    main()
```

El tamaño máximo del mensaje es 4096 caracteres

Contestar a los mensajes de un usuario

```python
#!/usr/bin/env python3
import telepot
import telepot.namedtuple
import time
from telepot.loop import MessageLoop

TOKEN = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
bot = telepot.Bot(TOKEN)

def handle(msg):
```

```
    chat_id = msg["chat"]["id"]
    texto = msg["text"]

    print("Recibiendo mensaje:")
    print(msg)

    respuesta = "Me has dicho " + texto
    bot.sendMessage(chat_id, respuesta)
    return
```

```
def main():
    MessageLoop(bot, handle).run_as_thread()
    print "Escuchando..."

    while 1:
        time.sleep(10)
    return

if __name__ == "__main__":
    main()
```

Ejemplo:

```
Escuchando...
Recibiendo mensaje:
{u'date': 1542706429, u'text': u'Hola', u'from': {u'username': u'Juan_perez',
u'first_name': u'Juan', u'last_name': u'Perez', u'is_bot': False,
u'language_code': u'es', u'id': 154195197}, u'message_id': 2704, u'chat':
{u'username': u'Juan_perez', u'first_name': u'Juan', u'last_name': u'Perez',
u'type': u'private', u'id': 154196127}}
```

- En los ejemplos anteriores, para preparar un ejemplo mínimo hemos escrito el token en el código fuente
- Por motivos de seguridad, en cualquier programa que no sea una prueba de concepto, el token (o cualquier otra contraseña similar) debería ir en un fichero aparte
  - Al leer el token desde un fichero, será necesario que suprimas el salto de línea final
  - Puedes usar el método `strip()`, que devuelve una copia de la cadena eliminando tabuladores, espacios y saltos de línea al princio y al final de una cadena

```
>>> a = '    hola    '
>>> a.strip()
'hola'
```

12.9. Expresiones Regulares en python

12.9.1. Introducción

Expresiones regulares. Introducción

- Las *expresione regulares* son expresiones que definen un conjunto de cadenas de texto
- Pertenecen a la disciplinas de teoría de autómatas y lenguajes formales. Las bases las sienta Stephen Cole Kleene en la década de 1950. Se desarrollan en los años 60 y se popularizan en los años 80
- Se denominan abreviadamente *re*, *regex* o *regexp*

  También *patrón*
- Las regex son una herramienta muy potente para procesar texto automáticamente. Especialmente texto plano, no son muy apropiadas para HTML o XML
- Las regex pueden manejarse desde
  - Herramientas clásicas como grep, sed, awk
  - Editores de texto
  - Librerías para lenguajes de programación clásicos como C o Pascal
  - Librerías nativas en cualquier lenguaje moderno: perl, python, java, ruby, c#, etc
- Entre las distintinas versiones hay similitudes y diferencias
  - Las regex *tradicionales* (grep, sed, awk) se parecen bastante entre sí.
  - Las regex *modernas* se parecen entre sí. Son una derivación de las tradicionales. Su uso resulta más sencillo
- Es una materia que puede llegar a resultar bastante compleja, conocerlas a fondo es difícil. Pero manejar sus fundamentos resulta de gran utilidad para prácticamente cualquier programador en cualquier entorno

Algunas definiciones

Decimos que una regex y una cadena de texto *encajan* o *no encajan*. [11]
Ejemplo. Patrón/regex

`[Aa]na [Pp].rez`

- La cadena `Ana Pérez` encaja
- También encajan las cadenas `ana perez`, `ana pérez`, `ana porez`, `Ana pÑrez`, etc
- La cadena `ANA PEREZ` no encaja

- Decimos que un carácter [12]
  - Se usa como literal si representa a ese carácter.
  - Se usa como metacarácter (o *comodín*) si tiene un significado especial, si representa algo distinto al propio carácter

  Ejemplo: el punto usado como literal, representa un punto. Usado como metacarácter, representa cualquier carácter
- Normalmente, cuando un carácter puede tomarse como metacarácter o como literal
  - Por omisión se toma como metacarácter
  - Para interpretarlo como literal, hay que escaparlo. Típicamente anteponiendo una barra invertida o incluyendolo entre comillas, rectas o dobles. Ejemplo: `\.`

12.9.2. Metacaracteres

Metacaracteres clásicos

```
^     Principio de cadena (principio de línea)
$     Fin de cadena (fin de línea)
.     Cualquier carácter
*     La regex precedente puede aparecer 0 o más veces
```

[11] O también *se corresponde*, *se ajusta a*. En inglés, *match*

[12] la palabra *carácter* es llana y lleva tilde, no es aguda. El plural es caracteres, también es llana

```
?     La regex precedente puede aparecer o no aparecer
[]    Clase de caracteres: uno cualquiera de los caracteres entre
      corchetes
[^]   Complementario de la clase de caracteres: cualquiera menos
      los incluidos entre corchetes
[a-f] Caracteres de la 'a' hasta la 'f'
{2,3} La regex precedente se puede repetir entre 2 y 3 veces
{2,}  La regex precedente se repite 2 o más veces
{,3}  La regex precedente se repite entre 0 y 3 veces
{4}   La regex precedente se repite 4 veces
()    Permite agrupar una regex
\2    El segundo grupo de regex
r1|r2 Una regex u otra

\<    Inicio de palabra
\>    Fin de palabra
```

Ejemplos

```
[a-z][a-z0-9_]*     letra minúscula seguida de cero o
                    más letras minúsculas, números o barras bajas
Señora?             Señor o Señora
Serg[eé][iy]? Ra(j|ch|h|kh)m[aá]n[ij]no(v|ff|w)
                    Sergéi / Sergei / Sergey / Serge
                    Rajmáninov / Rachmaninoff / Rahmanjnov ...
```

Dentro una clase de caracteres, cada carácter siempre se toma literalmente, no se escapa ningún posible metacarácter (excepto el cierre de corchetes)

```
[0-9.]        # Un dígito o un punto. (Aquí el punto representa
                un punto, no "cualquier carácter")
```

Atención: algunos metacaracteres de bash coinciden, otros tienen un significado distinto

```
?     En bash, cualquier carácter
*     En bash, cualquier carácter 0 o más veces
```

Fin de línea

El fin de línea se representa de diferentas maneras

- En MS-DOS/Windows y otros, el fin de línea se representa con `CRLF`

- En Unix, se representa con `LF`

Esto es una fuente tradicional de problemas

- En Windows, un fichero para Unix se verá como una única línea
- En Unix, un fichero para Windows tendrá un `^M` al final de cada línea

Algunos editores son lo bastante *listos* como para mostrar correctamente un fichero con un formato distinto

- Pero ocultar el problema a veces es contraproducente: puede suceder que la apariencia sea correcta, pero el compilador no lo acepte y muestre un error muy confuso

Nombre ASCII	**Abreviatura**	**Decimal**	**Hexa**	***Caret Notation***	**Notación C**
Carriage Return	CR	13	OD	`^M`	`\r`
Line Feed	LF	10	0A	`^J`	`\n`

- *Caret notation* es una método empleado en ASCII para representar caracteres no imprimibles. (Caret: acento circunflejo). Normalmente, se puede usar la tecla `control` para generar estos caracteres
- *Notación C*: Notación del lenguaje C, que después han seguido muchos otros como python

Obsérvese que nada de esto se refiere directamente a las expresiones regulares: Cuando en una cadena escribimos `\n`, se entiende que es un avance de línea (excepto si lo escapamos con otra barra adicional, o con una cadena cruda de python)

```
\n suele representar LF, excepto en macOS, donde suele representar CR.
En java o en .net sobre cualquier SO, siempre representa LF
```

Python emplea *universal newlines*:
En la E/S de ficheros, por omision:

- Sea cual sea el criterio de la entrada, lo convierte a `\n`
- A la salida, escribe el formato propio de la plataforma

Este comportamieto puede cambiarse si es necesario (consultar PEP 278 y PEP 3116)

Para cadenas que no provengan de un fichero, se puede emplear el método *splitlines()* de las cadenas, que:

- Trocea una cadena con el mismo enfoque (soporta todos los criterios), y elimina el fin de linea (sea el que sea)
- A menos que se invoque *splitlines(true)*, entonces conserve el fin de linea, inalterado

Otra fuente típica de problemas: ¿El fin de línea es un terminador o un separador?

- Algunas herramientas/aplicaciones/sistemas operativos entienden que es un separador, y por tanto la última línea no acaba en `\n` sino en fin de fichero
- Otras consideran que es un terminador, por tanto la última línea sí acaba en `\n` (P.e. Unix)

Todo esto son cuestiones que puede ser necesario considerar procesando texto. Pero si lo único que queremos es convertir ficheros entre Windows y Unix, no hace falta usar regex

```
sed -e 's/$/\r/' inputfile > outputfile     # Unix a Windows
sed -e 's/\r$//' inputfile > outputfile     # Windows a Unix
```

El metacarácter `$` de las regex no se corresponde exactamente con CR ni con LF. Su significado exacto depende de la plataforma. Normalmente encaja tanto con el fin de cadena como con la posición inmediatamente antes de LF/CR/CRLF

Metacaracteres modernos

El lenguaje perl es el *padre* de las regex modernas. Incluye los metacaracteres clásicos y añade otros nuevos. Lenguajes como python copian las regex de perl

```
Metac.                                          Clase equivalente
-------------------------------------------------------------------
\d    Dígito                                                [0-9]
\s    Espacio en blanco, tab...                      [\ \t\r\n\f] (*)
\w    Carácter de palabra (alfanumético o barra baja) [0-9a-zA-Z_]
\D    Cualquiera menos \d                                  [^0-9]
\S    Cualquiera menos \s                                   [^\s]
\W    Cualquiera menos  \w;                                 [^\w]
\b    Limite de palabra. (Secuencia de alfanuméricos o barra baja)

                                        (*)  \t: Tab
                                             \f: Form Feed, salto de página
```

Observaciones

- El único metacarácter que cambia entre regex clásicas y modernas es el límite de palabra, se usa `\b` y no `\< \>`
- Las locales no siembre están bien definidas, en tal caso para definir una palabra tal vez haya que incluir explicitamente las letras españolas (si procede)

12.9.3. Regexp en python

Regexp en python

- Para operaciones sencillas con cadenas, como búsquedas y sustituciones sin metacaracteres, es más eficiente emplear los métodos de las cadenas, como `find` y `replace`
- El módulo `re` tiene funciones a la que se puede pasar directamente una cadena regexp

```
>>> import re
>>> m=re.search('[0-9]+' , 'abc98521zzz')
>>> m.group(0)
'98521'
```

Pero aquí usaremos objetos regex, más potentes

Regexp en python

Para usar regexp, importamos el módulo `re`
`import re`

- Una regex es un objeto que construimos con la función `compile`

```
regex=re.compile("a+")
```

- Para buscar el patrón en una cadena tenemos los métodos
  - `match()`, que comprueba si el principio de la cadena encaja en la regex
  - `search()`, que comprueba si alguna parte de la cadena encaja en la regex

Ambos métodos devuelven

- Un objeto `SRE_Match` si han tenido éxito (que se evalúa como *cierto*)

```
#!/usr/bin/env python3
# regex01.py
import re
regex=re.compile("aa+")

m=regex.match("taartamudo")
print(m)   # None

m=regex.search("taartamudo")
print(m)   # Cierto

m=regex.match("aaahora")
print(m)    # Cierto
```

Casi siempre hay más de una regex posible. Ejemplo: Capturar una dirección IP

Estas sentencias son equivalentes

```
direccion_ip=re.compile(r"""\d\d?\d?\.\d\d?\d?\.\d\d?\d?\.\d\d?\d?""")
direccion_ip=re.compile(r"""(\d\d?\d?\.){3}\d\d?\d?""")
direccion_ip=re.compile(r"""\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}""")
direccion_ip=re.compile(r"""(\d{1,3}\.){3}\d{1,3}""")
```

- Es necesario *escapar* el punto
- Obsérvese que esta regex no se corresponde exactamente con una dirección IP. Por ejemplo admitiría `315.15.256.715`
- Suele ser conveniente definir la regex con *cadenas crudas* de python (`r"""cadena"""`)

  Esto evita tener que escapar las barras invertidas para que se tomen como literales.

  También permite, por ejemplo, que la secuencia `\n` se tome como como una barra invertida y una ene. (Y no como un salto de línea carro)

Comentarios en las regex

El flag `re.VERBOSE` es muy útil. Al activarlo se ignoran

- Los espacios (no *escapados*)
- Las almohadillas y todo el texto posterior, hasta fin de línea

```
ip = re.compile(r"""
        (\d{1,3}\.){3}  # de 1 a 3 digitos y punto, repetido 3 veces
        \d{1,3}         # de 1 a 3 digitos
        """, re.VERBOSE)
```

Otros flags

- `re.VERBOSE`

  `re.X`

  Permite comentarios dentro de la regex

- `re.IGNORECASE`

  `re.I`

  No distingue entre mayúsculas y minúsculas

- `re.LOCALE`

  `re.L`

  Hace que `\w, \W, \b, \B, \s` y `\S` tengan en cuenta las *locales*

Para combinar más de un flag, se usa la barra vertical (`'|'`), que es el operador *or* a nivel de bit.

Grupos

Un objeto `SRE_Match` devuelve en el atributo `group` las partes de la cadena que han encajado en la regex

`group[0]` es el texto que ha encajado en la regex completa

```
#!/usr/bin/env python3
# regex02.py
import re
ip = re.compile(r"""
        (\d{1,3}\.){3}  # 1,2 o 3 digitos y punto, repetido 3 veces
        \d{1,3}         # 1,2 o 3 dígitos
        """, re.VERBOSE)
texto=r"""Mi correo es j.perez@alumnos.urjc.es
          y mi dirección IP, 192.168.1.27"""
```

```
for linea in texto.split('\n'):
    m=ip.search(linea)
    if m:
        print(m.group(0))
```

Ejecución:

```
192.168.1.27
```

Los paréntesis

- Como hemos visto, definen el ámbito y precedencia de los demás operadores
- Además, definen grupos. El resultado de cada búsqueda devuelve en `group[n]` el grupo n-ésimo

```
#!/usr/bin/env python3
# regex03.py
import re
correo_alumno = re.compile(r"""
(
\b                  # Límite de palabra
[\w.]+              # 1 o más caracteres de palabra o punto
\b                  # límite de palabra
)                   # Hasta aquí el grupo 1
@
(alumnos\.urjc\.es) # Grupo 2
""", re.VERBOSE)

texto=r"""Llegó un correo de j.perez@alumnos.urjc.es preguntando
          si hay clase mañana"""

for linea in texto.split('\n'):
    m=correo_alumno.search(linea)
    if m:
        print("Alumno: {}".format(m.group(1)))    # j.perez
        print("Dominio: {}".format(m.group(2)))   # alumnos.urjc.es
```

Dentro de una regex, podemos hacer referencia a un grupo

```
#!/usr/bin/env python3
# regex04.py
import re

regex=re.compile(r"""(\b\w+\b)  # Una palabra
                 \s+            # Espacios
                 \1             # Grupo 1: la misma palabra
                 """, re.VERBOSE)

texto=r"""Buscando palabras repetidas repetidas"""
```

```
for linea in texto.split('\n'):
    m=regex.search(linea)
    if m:
        print(m.group(1))  # Devuelve "repetidas"
```

Ejemplo de definición explícita de palabra española

```
#!/usr/bin/env python3
# regex05.py
import re

regex=re.compile(r"""
      (\b                 # Límite de palabra
      [\wáéíóúÁÉÍÓÚñÑüÜ]+ # Palabra, incluyendo letras españolas
      \b)
      \s*                 # Espacios, opcionalmente
      $                   # Fin de línea
      """, re.VERBOSE)

texto=r"""Buscando la última palabra de la línea  """

for linea in texto.split('\n'):
    m=regex.search(linea)
    if m:
        print(m.group(1)) # Devuelve "línea"
```

Sustituciones

Además de `search` y `match`, los objetos regex tienen el método `sub(reemplazo,cadena)` que

- Busca el patrón en la cadena
- Si lo encuentra, reempleza el texto que ha encajado por `reemplazo`

  Dentro de `reemplazo` se pueden usar referencias a grupos
- Devuelve el texto resultante

```
#!/usr/bin/env python3
# regex06.py
import re
# reemplazamos los correos  login@urjc.es por
# [Correo de login en la URJC ]

correo_urjc = re.compile(r"""
(
\b                # Límite de palabra
[\w.]+            # 1 o más caracteres de palabra o punto
\b                # límite de palabra
```

```
)
@urjc\.es
""", re.VERBOSE)

texto="Si es necesario, escribe a j.perez@urjc.es"
for linea in texto.split('\n'):
    print(correo_urjc.sub(r"""[Correo de \1 en la URJC]""",linea))
```

Resultado de la ejecución

```
koji@mazinger:~/python$ ./test.py
Si es necesario, escribe a [Correo de j.perez en la URJC]
```

Regex multilínea

Hasta ahora hemos procesado cada línea de forma independiente de las demás, lo cual es bastante frecuente

En este caso

- El metacarácter `'^'` representa el principio de cadena, lo que equivale al principio de línea
- El metacarácter `'$'` representa el fin de cadena, lo que equivale al fin de línea
- El metacarácter `'.'` no encaja en el fin de línea

Pero en otras ocasiones querremos aplicar la regex a más de una línea. Esto generalmente requiere de algunos *flags* adicionales

- `re.DOTALL`

  `re.S`

  Hace que el metacarácter `'.'` encaje en el fin de línea

- `re.MULTILINE`

  `re.M`

  Hace que el metacarácter `'^'` represente el principio de línea

  El metacarácter `'$'` representa el fin de línea

```
#!/usr/bin/env python3
# regex07.py
import re
regex=re.compile(r"""
         ^                  # Principio de línea
         (\b
         [\wáéíóúÁÉÍÓÚñÑ]+  # Palabra
```

```
        \b)                 #
        .*                  # Resto de la línea
        ^                   # Comienzo de línea
        \1                  # La misma palabra
        """, re.VERBOSE|re.MULTILINE|re.DOTALL)

texto=r"""En este ejemplo estamos
buscando dos líneas que comiencen igual
buscando líneas con primera palabra
coincidente
"""
m=regex.search(texto)
if m:
    print(m.group(1))  # Devuelve "buscando"
```

Split con regex

Se puede trocear una cadena indicando con una regex cuál es el separador

Ejemplo: queremos una lista con todos los unos consecutivos, separados por ceros

```
>>> import re
>>> miregex=re.compile(r'0+')
>>> miregex.split('10011100011110001')
['1', '111', '1111', '1']
```

Atención: el separador, por definición, está entre dos elementos. No antes del primero ni después del último.

En el siguiente ejemplo los ceros no se comportan como separadores, por lo que el resultado no es exactamente el deseado (aunque se acerca mucho)

```
>>> miregex.split('00100111000111100010')
['', '1', '111', '1111', '1', '']
```

13. Sistemas de ficheros

13.1. Estructura del sistemas de fichero

Introducción

- Un sistema de ficheros es una forma de almacenar y organizar ficheros para permitir su uso
- Pueden usar un dispositivo de almacenamiento (disco, cdrom), la red o ser sólo un interfaz para acceder a datos
- Para poder empezar a almacenar información en un sistema de ficheros, éste tiene que ser *inicializado*
- En Unix, para poder usarlo, hay que *montarlo* en alguna parte de la jerarquía de directorios, un árbol cuya raíz es el directorio llamado `/`.

On a UNIX system, everything is a file; if something is not a file, it is a process

Los ficheros pueden ser

- Ficheros normales
- Directorios
- Ficheros especiales (Entrada y salida. Están en */dev*)
- Enlaces
- Fifos. (Pipes con nombre). Para comunicación entre procesos
- Sockets de dominio. Similares a los sockets TCP/IP

El primer caracter de `ls -l` representa:

```
-       Regular file
d       Directory
c       Special file
l       Link
p       Named pipe
s       Socket
b       Block device
```

Jerarquía del Sistema de Ficheros

Para quien se acerca a Linux resulta confuso un `ls -l /`

```
drwxr-xr-x    2 root     root         4096 ene 30 20:34 bin
drwxr-xr-x    2 root     root         4096 mar 12 19:46 boot
drwxr-xr-x    5 root     root        24576 may 22 06:27 dev
drwxr-xr-x   66 root     root         4096 may 19 00:26 etc
drwxrwsr-x    7 root     staff        4096 abr 16 17:36 home
drwxr-xr-x    6 root     root         4096 feb  1 18:02 lib
drwxr-xr-x    2 root     root        16384 nov  7  2000 lost+found
dr-xr-xr-x    2 root     root         4096 nov 10  2000 mix
dr-xr-xr-x   67 root     root            0 may 19 02:25 proc
drwxr-xr-x   14 root     root         4096 feb 12 19:28 root
drwxr-xr-x    2 root     root         4096 ene 30 20:30 sbin
drwxrwxrwt    9 root     root         4096 may 22 10:19 tmp
drwxr-xr-x   15 root     root         4096 nov  8  2000 usr
drwxr-xr-x   16 root     root         4096 nov  9  2000 var
```

- La estructura de todos los Unix se *parece*
- La estructura de todas las distribuciones Linux se *parece mucho*

Jerarquía clásica

La jerarquía actual puede resultar algo ilógica, pero hay motivos históricos
En los primeros Unix los discos eran más pequeños y más caros,
en uno estaba lo *imprescindible* para que el sistema funcionase:

```
/
/etc
/lib
/tmp
/bin
/root
```

y en un segundo disco, se montaba `/usr`

```
/usr/spool
/usr/bin
/usr/include
/usr/tmp
/usr/adrn
/usr/lib
```

13.2. FHS Filesystem Hierarchy Standard

FHS Filesystem Hierarchy Standard

Estándar propuesto para todos los Linux y para los UNIX que quieran unirse. Año 1994. Versión actual: 3.0 (junio 2015)

Dos criterios

¿Un fichero puede almacenarse en una máquina y usarse en otra?

- Sí: Compartibles. (*shareable*)
- No: No compartibles. (*unshareable*)

¿Un fichero puede cambiar sin intervención del administrador?

- Sí: Dinámicos.
- No: Estáticos. Pueden almacenarse el modo sólo-lectura. Copias de seguridad menos frecuentes

1. Directorios de usuarios
2. Programas (incluyendo mandatos y librerías)
3. Configuración del sistema
4. El Hardware
5. Documentación
6. Ficheros Temporales
7. Otros directorios relacionados con el S.O.
8. Puntos de montaje

13.2.1. Directorios de usuarios

Directorios de usuarios

- Directorio del administrador

  `/root`

- Usuarios locales

  `/home/jperez`

  o bien

  `/home/profesores`

  `/home/alumnos`

13.2.2. Programas y mandatos

Programas y mandatos

- Mandatos útiles para todos los usuarios
  /bin
  /usr/bin
- Mandatos útiles para el root
  /sbin
  /usr/sbin

(Todo lo que haya bajo `/usr` debería ser sólo lectura)

- Programas
  - Software no incluido en la distribución Linux
    /usr/local
  - Grandes aplicaciones en la distribución
    /opt
- Librerías estáticas y dinámicas
  /lib
  /usr/lib
  /usr/local/lib
- Ficheros de cabecera (para compilar)
  /usr/include
- Ficheros independientes de la arquitectura
  /usr/share

13.2.3. Configuración del sistema

Configuración del sistema

Directorio `/etc`

- Información sobre el sistema de ficheros (puntos de montaje, opciones)
  /etc/fstab

- cuentas de usuarios

  /etc/passwd

- Passwords de los usuarios

  /etc/shadow

- Scripts para arranque del sistema

  /etc/init.d

- ...

13.2.4. El Hardware

El Hardware

Los dispositivos del sistema `/dev`

```
/dev/hda       IDE primario master
/dev/hdb       IDE primario slave
/dev/hdc       IDE secundario master
/dev/hdd       IDE secundario slave
/dev/hda1      Primera partición primaria del hda
/dev/hda2      ...

/dev/sda       Primer disco SCSI
/dev/sdb       Segundo disco SCSI
/dev/sda1      ...

/dev/nvme0n1   Primer disco nvme
/dev/nvme0n2   Segundo disco nvme
/dev/nvme0n1p1 Primera partición del primer disco nvme


/dev/cdrom
/dev/fd0     disquete
/dev/audio   tarjeta sonido
/dev/modem
/dev/mouse
/dev/input/mouse0
/dev/ttyN    donde N es el nº de consola (no gráfica)
/dev/pts/N   Consola (X Window)
```

El estándar no dice mucho sobre `/dev`, es bastante variable

- Ficheros *virtuales* que representan las estructuras del Kernel en ejecución, dan información sobre la cpu...

```
/proc/cpuinfo     CPU
/proc/pci         Tarjetas PCI
/proc/ioports     Puertos I/O
/proc/meminfo     Información sobre la memoria
/proc/NN          Información sobre el proceso de pid NN
```

Los directorios `/proc` y `/sys` no se corresponden con discos físicos, sino que son un medio de enviar y recibir información directamente del *kernel*.

Cuando se lee o se escribe algún fichero del `/proc`, se está pidiendo o recibiendo información del kernel

13.2.5. Documentación

Documentación

- /usr/share/doc

  Documentación sobre el software del sistema

- /usr/man

  Ficheros del mandato *man*

13.2.6. Ficheros Temporales

Ficheros Temporales

- Ficheros temporales

  (se borran cuando la máquina arranca)

  `/tmp`

- Fragmentos de ficheros recuperados

  `/lost+found`

- Ficheros que cambian con frecuencia, de aplicaciones

  `/var`

```
/var/log/syslog        bitácora principal del sistema
/var/log/messages      otra bitácora con diversos mensajes
/var/log/dmesg         mensajes del sistema al arrancar
/var/spool/lpd/lp      spool de la impresora
/var/tmp               Ficheros temporales
/var/mail              Correo de los usuarios
```

- Ficheros del sistema que cambian con frecuencia

  `/run`

  Esta es la principal novedad respecto a la versión anterior, (FHS 2.3, año 2004). El directorio equivalente a este era `/var/run`. Resultaba problemático porque `/var/run` normalmente no estaba disponible durante el arranque (`/var` no es especialmente importante, podía estar en una partición distinta)

13.2.7. Otros directorios relacionados con el S.O.

Otros directorios relacionados con el S.O.

- `/boot`

  Todo lo requerido para el arranque, antes de que el sistema ejecute programas de usuario

- Código fuente

  - Código fuente del software de sistema
    `/usr/src`
  - Código fuente del kernel linux
    `/usr/src/linux`

13.2.8. Puntos de Montaje

Puntos de Montaje

Unidades extraibles: Disquetes, cdrom, *pendrives*
Solían colocarse en el raiz p.e. `/cdrom`. Pero esto llena el raiz de directorios
En FHS 2.3 (año 2004) aparece `/media`
`/media/cdrom` `/media/cdrecorder` `/media/zip` `/media/floppy`

- Si solo hay uno de un tipo:

  `/media/cdrom`

- Si hay más de uno del mismo tipo

  `/media/cdrom0`

  `/media/cdrom1`

  `/media/cdrom -> /media/cdrom1`

`/mnt`

Directorio vacío para que el administrador monte un sistema de ficheros que necesita temporalmente. Los programas no deberían usarlo

- `/mnt/cdrom` ¡No es estándar!

  Es una costumbre reciente, va contra el estándar. Dentro de `/mnt` debe estar directamente el sistema de ficheros temporal, sin subdirectorios

13.3. Montaje de sistemas de ficheros

Montaje de sistemas de ficheros

- Normalmente, no todos los ficheros del árbol de directorios se encuentran en el mismo disco.
- *Punto de montaje*: directorio que pertenece a un disco (o *partición*) distinto, junto con todo su contenido (excluyendo otros puntos de montaje).
- Se pueden consultar los puntos de montaje junto con los discos o particiones que están *montadas* en ellos con las órdenes *mount* y *df*

13.3.1. mount, df, lsblk

mount, df, lsblk

- `mount`: Muestra las particiones, puntos de montaje, tipo de partición y opciones de cada una de ellas:

```
/dev/hda2 on / type ext3 (rw,noatime)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/hda5 on /scratch type ext3 (ro,noatime)
tmpfs on /tmp type tmpfs (rw)
```

- `df`: Muestra cada una de las particiones *con ficheros reales* montadas en el sistema, el punto en el que está montada, su capacidad y su uso:

```
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/hda2             28842780   6957692  20419960  26% /
/dev/hda5             38448276  32838556   3656620  90% /scratch
tmpfs                   517960      1196    516764   1% /tmp
```

- `lsblk -f`: Muestra cada disco (dispositivo de bloques)
- `blkid`: Muestra cada disco, aunque no esté montado

Para montar un sistema de ficheros

- Crear el directorio si no existe:

  `mkdir /var`

- Hacer visible el sistema de ficheros bajo ese directorio:

  `mount -t ext2 -o rw /dev/hda3 /var`

  (es más habitual indicar las opciones en `/etc/fstab`)

- Si queremos desmontar (o hacer invisible) un sistema de ficheros que esté montado en el directorio `/var`:

  `umount /var`

13.3.2. El fichero /etc/fstab

```
# <filesystem> <mount point>   <type>  <options>         <dump><pass>
proc           /proc           proc   defaults             0    0
/dev/hda2      /               ext3   noatime              0    1
/dev/hda5      /scratch        ext3   noatime,ro           0    1
/dev/hda6      none            swap   sw                   0    0
tmpfs          /tmp            tmpfs  defaults             0    0
/dev/sda1      /media/pendrive vfat   defaults,user,noauto 0    0
```

- `mount -a` monta todo lo indicado en este fichero

- En el arranque se ejecuta `mount -a`
- `mount /media/pendrive`

  monta el pendrive con todas las opciones indicadas en fstab

`<dump>` ¿Incluir en las copias de seguridad hechas con *dump*? (Normalmente no)

`<pass>` Orden para el `fsck` del arranque (0: desactivado).

`<options>`

- `rw`: Permisos de lectura y escritura.
- `ro`: Sólo lectura.
- `auto/noauto`: ¿Montar automáticamente con `mount -a`?
- `user/nouser`: ¿Los usuarios normales pueden montar y desmontar? (o hace falta ser `root`)
- `exec/noexec`: ¿Se pueden ejecutar binarios?
- `sync`: Al modificar un fichero, se escribe físicamente de inmediato
- `async`: Se usan buffers
- `defaults`: `rw, suid, dev, exec, auto, nouser, async`
- ...

13.3.3. Tipos de sistemas de ficheros

Tipos de sistemas de ficheros

- Tradicionales
  - `msdos`: El usado por MS-DOS y Windows pre-95, sin permisos ni dueños, nombres de fichero de 8 caracteres con extensiones de 3 caracteres
  - `vfat`: Usado a partir de Windows-95, compatible con MS-DOS pero con posibilidad de nombres de fichero largos
  - `ntfs`: Desde Windows NT hasta Windows XP. Añade características de seguridad (permisos, dueños, etc). Los primeros *drivers* para Linux tenían limitaciones, en la actualidad se puede leer y escribir con normalidad

- `iso9660`: Sistema de fichero utilizado en los CDs de datos
- `minix`: usado por MINIX y por los primeros Linux
- `ext2`: Sistema de ficheros tradicional en Linux

- Con *journal*
  - `ext4`: sucesor de `ext2` y `ext3`. El más utilizado actualmente
  - `reiserfs`, `jfs`, `xfs`
    Mejores prestaciones, pero incompatibles con `ext2`

- Con características especiales :

  `romfs, cramfs, autofs, umsdos`

- No asociados a dispositivo

  `proc, sysfs, devfs, devpts, tmpfs, ramfs, usbfs`

- Remotos:
  - `nfs`: *Network File System*, desarrollado por SUN, el más usado entre los sistemas ficheros remotos en UNIX
  - `smb/cifs`: Sistema de ficheros remotos usado por Microsoft
  - `ncp`: *Netwate Core Protocol*, protocolo sobre IPX para montar sistemas de ficheros de Novell Netware
  - `sshfs`: *Secure SHell FileSystem*, protocolo basado en ssh

- Soporte de otras plataformas:

  `hfs` (Apple Macintosh), `bfs` (*Boot File System*, SCO), `efs` (SGI, IRIX), `jffs` (*Journaling Flash File System*), `hpfs` (OS/2), `qnx4`, `sysv` (System V), `ufs` (SunOS, FreeBSD, NetBSD, OpenBSD)...

13.3.4. Sistemas de Ficheros en Espacio de usuario

Sistemas de Ficheros en Espacio de usuario

- Los sistemas de ficheros tradicionales están implementados en el núcleo. Añadir uno sistema de ficheros es complicado, y puede comprometer la integridad del sistema.

- Los sistemas de ficheros en espacio de usuario son aplicaciones *normales*

- Para Linux, FreeBSD, NetBSD, OpenSolaris y Mac OS X exite FUSE *Filesystem in Userspace*. Es un módulo del núcleo que actúa de puente entre el núcleo y el código del sistema de ficheros

Ejemplos de sistemas de ficheros FUSE

- sshfs
- GmailFS. Almacena los datos sobre correos de gmail. No es fiable porque no está aprobado por google. (Tampoco prohibido, al menos explícitamente)
- Acceso a ficheros empaquetados (tgz, zip, etc)
- Almacenamiento en Bases de Datos
- Encriptación
- Hardware poco común
- Sistemas de versiones de ficheros (CVS, SVN...)
- Monitorización de sistemas de ficheros

13.3.5. frametitle

13.3.6. sshfs

Secure SHell FileSystem. Basado en FUSE. Sistema de ficheros de red

- Menos eficiente pero más seguro que NFS
- En el servidor basta disponer del demonio ssh convencional
- En el cliente basta instalar el paquete `sshfs`

Montar el *home* remoto:
`sshfs -C usuario@maquina: /punto/de/montaje`
Montar un directorio remoto
`sshfs -C usuario@maquina:/un/directorio  /punto/de/montaje`
Desmontar:
`fusermount -u /punto/de/montaje`

upstart

- El sistema de arranque tradicional de Linux (System V) no es adecuado para las máquinas actuales
  - Son externos: aparecen y desaparecen
  - Están en red
  - Ahorran energía
  - ...
- *Upstart* es un sistema de arranque basado en eventos, desarrollado por Ubuntu, con el propósito de extenderlo a todos los Linux

  Aparece en Ubuntu 6.10 *edgy* (Octubre de 2006)
- Alternativas: *launchd* (macOS X), *initng*, SMF
- Está previsto que reemplace a *cron* y tal vez a *inetd*, manteniendo siempre la compatibilidad

En *upstart* se modifica la columna `<filesystem>` de `/etc/fstab`, incorporando un *Universally Unique Identifier*

```
# <file system> <mount point>   <type>                      <options><dump><pass>
proc            /proc           proc                              defaults 0 0
UUID=e8a76033-f833-490d-8a55-ceca132c2ba7 / ext3 defaults,errors=remount-ro 0 1
UUID=e38c8abf-1af7-49be-bba5-bcf45dab8dc2 /home              ext3 defaults 0 2
UUID=967cf88c-7b0b-42a9-bf93-deb7b710aad2 /media/sda6        ext3 defaults 0 2
UUID=f5c3bc51-7795-4bc9-b18e-4a16b7496e93 none                     swap sw 0 0
/dev/hda        /media/cdrom0                       udf,iso9660 user,noauto 0 0
```

13.4. Codificación de caracteres

Codificación de caracteres

Correspondencia entre un carácter de lenguaje natural y un símbolo en otro sistema de representación. En informática, uno o más octetos

A veces se llama *code pages* (IBM, Microsoft)

13.4.1. Codificaciones clásicas

- EBCDIC: Extended Binary Coded Decimal Interchange Code. IBM, año 1963. 8 bits. Se usa en algunos equipos IBM. Diferentes versiones incompatibles entre sí
- ASCII: American Standard Code for Information Interchange. ANSI, American National Standards Institute, año 1963). 7 bits. Solo inglés

13.4.2. ASCII extendido

ASCII extendido

8 bits. Cada conjunto de idiomas necesita su propia variante. Compatible con ASCII

- Code Pages 437. Inglés. Primeros IBM PC, MS-DOS

  Code Pages 850. Europa occidental. Primeros IBM PC, MS-DOS

- ISO-8859 (Organización Internacional para la Estandarización), año 1992. Habitual en linux hasta mediados de los años *cerenta*

  ISO-8859-1, informalmente conocido como Latin-1

  ISO-8859-2 europa central, ISO-8859-5 cirílico , ISO-8859-6 árabe, ...

  ISO-8859-15 o Latin-9. Año 1998. Muy parecido a Latin-1, incluye el símbolo del euro

- windows-1252. Parecido a ISO-8859-1. Se confunden con frecuencia. Se empleaba en los primeros Windows

13.4.3. Unicode

Unicode

Estándar industrial. *Unicode Consortium*, año 1991. Compatible con ISO 10646.

Asocia un número a cada carácter empleado por algún lenguaje escrito del mundo. Más de 100.000 caracteres

Se puede codificar de diferentes maneras

- UTF-8 es la forma en Unix de codificar unicode.

  Compatible con ASCII. Cada carácter ocupa entre 1 y 4 octetos

- UTF-16. Cada carácter ocupa entre 2 y 4 octetos.

  Nativo en Windows desde Windows 2000, aunque se seguía usando windows-1252.

- Punycode. RFC 3492. Empleado en la Internacionalización de Nombres de Dominio en Aplicaciones (IDNA). Años 2003-2005. Permite nombres de dominio en unicode.

  ```
  españa.es   ->   xn--espaa-rta.es
  ortuño.es   ->   xn--ortuo-rta.es
  ```

- UCS-2, UCS-4, SCSU, ...

13.4.4. recode

recode

Orden que convierte ficheros entre diferentes codificaciones

- `recode utf-8`

  Lee *stdin*, convierte desde utf-8 hasta las locales actuales y escribe en *stdout*

- `recode latin-1..utf-8`

  Lee *stdin*, convierte desde latin-1 hasta utf-8 y escribe en *stdout*

- `recode  utf-8..windows-1252 fichero`

  Modifica el fichero, convirtiendo desde utf-8 hasta windows-1252

14. DevOps

14.1. DevOps

Definición de DevOps

DevOps es un término acuñado por Andrew Shafer y Patrick Debois en la conferencia de desarrollo *Agile* del año 2008

Se origina con la composición de dos palabras:

- *Development*

  Desarrollo, programación de software en sentido amplio, esto es, análisis, diseño, codificación y prueba

- *Operations*

  Operaciones, explotación del software, puesta en producción, uso real

Definición de Bass, Weber y Zhu [4]:

DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality

Definición de Davis y Daniels [2]:

Devops is a cultural movement that changes how individuals think about their work, values the diversity of work done, supports intentional processes that accelerate the rate by which businesses realize value, and measures the effect of social and technical change. It is a way of thinking and a way of working that enables individuals and organizations to develop and maintain sustainable work practices. It is a cultural framework for sharing stories and developing empathy, enabling people and teams to practice their crafts in effective and lasting ways

Definición de Huttermann [1]:

DevOps is a mix of patterns intended to improve collaboration between development and operations. DevOps addresses shared goals and incentives as well as shared processes and tools. Because of the natural conflicts among different groups, shared goals and incentives may not always be achievable. However, they should at least be aligned with one another

Aquí lo definiremos como

```
Grupo de técnicas que buscan optimizar el trabajo
conjunto de desarrolladores de software y administradores
de sistemas
```

- Optimizar: que sea rápido, sencillo, barato, de calidad y sin conflictos

Estas *técnicas* se agrupan en dos grandes categorías

- Culturales, políticas, organizativas. Referidas a la interacción entre personas
- Herramientas software

Las segundas son importantes, pero las primeras, más

Hay algunas cosas relativamente claras:

- Qué es *DevOps*
- Qué no es *DevOps*
- Qué problemas se quieren solucionar
- Qué objetivos se buscan
- Lo relativo a herramientas software

Lo que no está tan claro es *cómo*. *DevOps* es una disciplina compleja

- Organizar el trabajo de personas es difícil. No se aprende en un libro. Lo que puede valer en un entorno, puede ser inaplicable en otro

¿Qué NO es DevOps? (1)

- *DevOps* no significa que desaparezca la frontera entre desarrollo y producción
- No significa que los desarrolladores controlen el software en producción
- No significa que los administradores editen el código fuente
- Según la bibliografía especializada, *DevOps* nunca debería ser un departamento en una empresa, ni un cargo. No tiene sentido hablar de *Ingeniero DevOps*

    - Aunque la industria sí usa este término

- No significa que una sola persona desarrolle y opere

    - Excepto tal vez en empresas muy pequeñas

- No significa que una persona trabaje por dos (y cobre por una)

¿Qué NO es DevOps? (2)

- *DevOps* no es una herramienta software. Tienen su utilidad, pero ni son suficientes ni son imprescindibles

- *DevOps* no es una certificación. No es una metodología concreta y única que se pueda enseñar, que se pueda seguir y de la que uno se pueda examinar

Con frecuencia, en una ofertas de empleo donde se solicita un *ingeniero devops*, realmente lo que se está buscando es un desarrollador con conocimientos de integración continua/entrega continua/despliegue continuo.

Word Cloud para DevOps en ofertas de empleo

Fuente: The DevOps Job Market, Scalyr blog

14.2. Desarrollo Ágil

Desarrollo Ágil

DevOps está muy ligado con el desarrollo de software *agile* (*ágil*), proviene de la misma comunidad, tiene objetivos muy similares

- Podemos considerar *DevOps* una extensión del moviento *agil* a la explotación del software, no solo a su desarrollo

Modelo de desarrollo de software en cascada

El desarrollo en cascada (*waterfall*) es el tradicional, generalmente aceptado y prácticamente único hasta los años 1990.

Formado por pasos que se siguen secuencialmente, de forma rígida, uno tras otro, sin vuelta atrás

1. Análisis de requerimientos
2. Diseño
3. Desarrollo (programación)
4. Prueba
5. Despliegue
6. Mantenimiento

Desarrollo Ágil

Ante los problema del modelo en cascada, en los años 1990 empiezan a aparecer diversas metodologías de desarrollo de software con un enfoque muy diferente

- *rapid application development, the unified process, dynamic systems development method (DSDM), scrum, extreme programming (XP), feature-driven development*

En 2001 se publica el *Manifesto for Agile Software Development* que resume y condensa todas estas metodologías

Manifiesto por el desarrollo ágil de software:

```
Estamos descubriendo formas mejores de desarrollar
software tanto por nuestra propia experiencia como
ayudando a terceros. A través de este trabajo hemos
aprendido a valorar:

Individuos e interacciones sobre procesos y herramientas.
```

```
Software funcionando sobre documentación extensiva.
Colaboración con el cliente sobre negociación contractual.
Respuesta ante el cambio sobre seguir un plan.

Aunque valoramos los elementos de la derecha,
valoramos más los de la izquierda.
```

12 principios del manifiesto ágil
`http://agilemanifesto.org/iso/es/principles.html`

Scrum

Scrum es una de las metodologías de desarrollo de software ágil más populares. Una idea dentro de la filosofia *DevOps* es incluir las operaciones en los *sprints* de *Scrum*. Esto se puede hacer de varias formas, es una materia abierta

- Es posible integrar personal de operaciones en los equipos *Scrum*, aunque no es una idea muy habitual, va contra el principio de separación desarrollo-operaciones
- Es más natural una integración más débil: p.e. asistencia de personal de operaciones a las reuniones de *Scrum*, como miembro externo
- También se pueden adaptar las técnicas de *Scrum* dentro del equipo de operaciones

Scrum es una metodología de desarrollo ágil de software elaborada por Ken Schwaber y Jeff Sutherland, publicada en 1995

- Se forman equipos de desarrolladores, típicamente entre 5 y 9 (más un *product owner* más un *scrum master*)
- El trabajo se descompone en ciclos denominados *sprints*, que duran entre 1 y 4 semanas. Típicamente 2
- Al final de cada *sprint* se entrega una versión del software

Equipos de Scrum

En los equipos de *scrum* hay tres roles

- *Dueño del producto, Product owner*

  Es una persona, que representa al cliente. Tiene la visión del producto final y poder de decisión sobre cómo debe ser el producto.

- *Scrum master*

  Es el responsable de que se siga la metodología *scrum* . Modera las reuniones, dirije al equipo en lo necesario para que el equipo se auto-dirija

- Miembro del equipo

  Son los desarrolladores. Los equipos son multifuncionales, sin distinción de roles entre analista/programador/tester. Todos puedes hacer cualquier función y son responsables de todo (aunque cada uno tenga una especialidad propia)

Los valores en *scrum* son

- Respeto entre las personas
- Responsabilidad y disciplina auto-impuesta
- Compromiso
- Trabajo enfocado en aportar valor al cliente

Las unidades básicas de construcción del producto son las *historias de usuario*

- El usuario de tipo xxxx quiere hacer yyyy. Esto le aporta el valor zzzz
- Las *historias de usuario* las aporta el *product owner*

Reuniones de trabajo en Scrum

Planificación del *sprint*

- Reunión de todo el equipo, típicamente de unas 4 horas, antes de cada *sprint*
- A partir de las *historias de usuario* que propone el *product owner*, el equipo decide cuáles implementar y cómo

Scrum diario

- Reunión de 15 minutos, del equipo al completo, siempre en el mismo sitio, a la misma hora, de pie, con horario inflexible, falte quien falte

- Cada miembro explica qué hizo ayer, qué hará hoy, qué obstáculos cree que pueden impedir el sprint

Evaluación de *sprint*

- Reunión de unas dos horas al final del sprint
- Se presenta lo realizado (solo lo concluido)
- Se evalúa el trabajo.

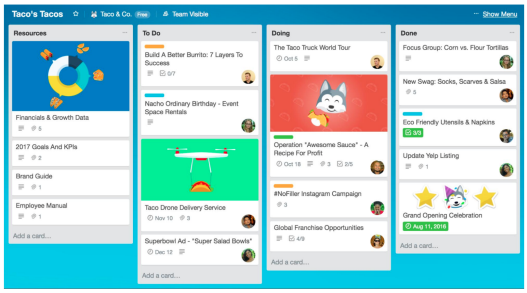
Kanban

Kanban es una metodología de gestión de procesos

- Tiene su origen en la industria del automóvil: *Toyota Prodution System* y *Lean Manufacturing*
- Se usa, entre otras cosas, para desarrollo de software, como metodología ágil y especialmente ligera
- Es muy adecuada para *DevOps*. Se puede usar de diversas formas, por ejemplo que desarrollo y operaciones compartan la misma pizarra Kanban, aunque cada equipo gestione sus tareas

El elemento principal es el tablero Kanban, también llamado pizarra Kanbam. Es un diagrama que representa el flujo de trabajo

- Tradicionalmente se usaba una pizarra con tarjetas adhesivas o imanes,
- Hay versiones software, típicamente como aplicación web. P.e. `https://trello.com`



Tablero Kanban con Trello

- Cada tarea, característica o historia de usuario se anota en una tarjeta o *post-it*, que se va desplazando desde la columna de la izquierda hasta la columna de la derecha
- WIP: *Work in Progress*. Tarjetas que circulan por el tablero. Es importante minimizar el WIP
- El numero de columnas es variable, entre 4 y 7 son valores típicos. La denominación de cada columna se adapta para cada empresa

  Ejemplo:

  ```
  Pendiente | Analizando | En desarrollo | Probando | Aceptado | Producción
  ```

Tarjetas Kanban

Cada tarjeta tiene

- Descripción de la tarea

Puede tener:

- Quién la está haciendo
- Su fecha límite
- Distintos colores
  - Tal vez según la urgencia
  - Más habitualmente, por el tipo de trabajo

    P.e. Verde: mantenimiento. Amarillo: historia de usuario. Rojo: Bug
- Indicador de progreso

14.3. Problemas habituales

Diferencias desarrollo-operaciones

La mayoría de problemas que se procura resolver con *DevOps* parten de que normalmente hay diferencias marcadas entre desarrollo y operaciones. Son equipos muy separados, con diferente lenguaje, culturas, habilidades, objetivos...

Una de las mayores diferencias es que

- Los desarrolladores (analistas, programadores, *testers* y responsables de calidad) buscan el cambio continuamente, para corregir errores y añadir funcionalidad
- Los administradores (administradores de sistemas, de bases de datos y de redes) buscan estabilidad. Ven cualquier cambio como un riesgo potencial. Nadie les agradecerá la nueva funcionalidad. Pero sí les culparán de los problemas de explotación provocados por los cambios

Problemas típicos (1)

Enumeramos a continuación algunos problemas habituales entre el equipo de desarrollo y el equipo de operaciones, que las técnicas *DevOps* buscan solucionar

- El software que funcionaba en los equipos de desarrollo, da errores en producción
  - Desarrollo echa la culpa a operaciones
  - Operaciones echa la culpa a desarrollo
- Aparece algún problema menor en la funcionalidad o el rendimiento. El equipo de desarrollo hace un parche rápido sin pasar todos los controles de calidad
  - Operaciones instala el parche, arregla una cosa pero rompe otra
  - Operaciones no instala el parche, porque sabe que los parches son peligrosos

Problemas típicos (2)

- Desarrollo hace un producto de baja calidad, lo mínimo para ser aceptado

- El trabajo ya está *hecho*. El diagrama de Gantt está cumplido. Los problemas posteriores no importan, son cosa de *mantenimiento*, de otro contrato, de otra subcontrata, de otro presupuesto...
- Consumir más recursos (tiempo, esfuerzo) para entregar un producto de más calidad, no reportará beneficios al equipo de desarrollo

Problemas típicos (3)

- Problema contrario al anterior: Desarrollo *anancástico* (demasiado perfeccionista)
  - El equipo de desarrollo prepara un software con cambios radicales (nuevo lenguajes, nuevas librerías). O preparado para eventualidades poco probables.
  - Todo lo contrario al *pequeño cambio incremental*. No tiene en cuenta las implicaciones para operaciones. Dispara los costes y/o los plazos, peligrando la viabilidad de la empresa. A operaciones o al mercado no llegan las soluciones adecuadas porque la solución *óptima* no está disponible

Problemas típicos (4)

- Para intentar evitar los problemas anteriores, se *mejora* la especificación de los *deliverables* (entregables) que unos proporcionan a otros
  - Negociaciones duras, criterios muy rígidos, contratos complicados, especificaciones a la defensiva. Lo fundamental es, en caso de problema, dejar claro *quién tiene la culpa*
  - Esto aumenta los trámites, la preparación y la frecuencia de las entregas
  - Aumenta la distancia entre los equipos

Problemas típicos (5)

- Cultura del héroe (*rock star, ninja, crack*)
  - Programador individualista. Con frecuencia hace software con errores y no documentado. Aparentemente es muy valioso porque solo él sabe arreglar esos errores

- Planificación rígida
    - Inicialmente se prepara un diagrama de Gantt. Luego todo se fuerza para que encaje en el diagrama

14.4. Aspectos humanos

Valores a promover

- Equipos motivados y productivos
- Compromiso con objetivos y valores comunes
- Respeto al otro
- Colaboración bienintencionada entre las partes/las empresas
- Aceptación de un cierto ratio de errores como inevitables, sin buscar culpables

Todo esto

- Tiene ventajas evidentes
- Es difícil de conseguir

Esta caricatura de *Pantomima Full* es muy ilustrativa
`https://www.youtube.com/watch?v=FMgC_IROlSA`

Motivación de equipos

Según Poppendieck, la motivación se puede conseguir con:

- Sensación de pertenencia
- Confianza en una cierta tolerancia a los errores
- Confianza en la capacidad propia y del resto del equipo
- Celebración conjunta de los progresos
    - Teniendo en cuenta que no todo el mundo sale de copas o juega al Paintball

Trabajo en equipo productivo

- Definir objetivos, métodos, pasos, plazos temporales... y reajustarlo cuando sea necesario
- Evitar la microgestión. Que los gestores digan qué hacer, pero sin demasiados detalles del cómo. El equipo se auto-organiza
  - Esto suele ser más eficiente
  - Hace al equipo sentirse más valorado
- Hacer pequeños experimentos. Fallar a menudo pero pronto y con pequeñas cosas
- De vez en cuando (una vez al día, a la semana...) reservar un rato para alguien de operaciones se siente con alguien de desarrollo
- Evitar el *presentismo laboral*. Respetar el equilibrio entre el trabajo y la vida personal. No esperar que los empleados hagan jornadas maratonianas en la oficina y que luego contesten al correo a cualquier hora

Comunicación efectiva

Que los individuos y equipos:

- Comprendan las circustancias y dificultades de los demás
- Busquen influenciar en otros de forma positiva.

  No porque *te lo mando* o *me debes una*, sino porque esto es lo mejor para todos
- Reconozcan el trabajo ajeno. Hacerlo en público es mucho más efectivo. Y si hay que hacer algún reproche, con mucha mano izquierda y en privado

Reuniones de calidad

- Todos los convocados llegan puntuales. La reunión acaba puntualmente
- Meta-decisiones claras. Ya sea por jerarquía, por votación, o idealmente, por consenso
- Los participantes hablan de uno en uno
- Lo que solo afecta a unos pocos, no se trata en el tiempo de todos

- ...

Sin olvidar las

- Discusiones retrospectivas. Reuniones con periodicidad predeterminada para tratar las etapas superadas, para analizarlas y extraer conclusiones.
- Reuniones *post morten*. Similares a las retrospectivas, pero provocadas por un problema concreto. Siempre es necesario trabajar de forma constructiva sin echar la culpa a nadie, pero en estos casos, mas que nunca.

14.5. Aspectos técnicos

Automatización

La automatización es una técnica fundamental en *DevOps*
Tareas que se pueden automatizar:

- Construcción (compilación)
- Pruebas
- Despliegue (puesta en producción)
- Configuración en los distintos entornos
- Monitorización
- Control de incidencias

Ultimamente se ha introducido el término *orquestar*:

- Automatizar

  Usar herramientas (scripts o similares) que permitan realizar una tarea sin intervención de una persona

- Orquestar

  Coordinar diversas automatizaciones de tareas individuales, para que formen procesos / flujos de trabajo

Automatizar (incluyendo orquestar) es, en general, positivo. Con algunas salvedades

- Debemos asegurarnos de que merezca la pena. Que el esfuerzo necesario para preparar y mantener la automatización, sea menor que el esfuerzo de realizar las tareas a mano.
- Paradoja del exceso de automatización

  Inevitablemente, habrá ocasiones en que el sistema requiera intervención humana (errores, cambios no previstos...). Cuánto más automatizado esté el sistema:

  - Más complejos serán estos cambios, más especializado tendrá que ser el personal
  - Menos especializado será el personal del día a día.

    Como esto lo puede llevar cualquiera, el resultado es que lo acaba llevando cualquiera

Técnicas de despliegue (1)

- Despliegue frecuente

  Un cambio grande puede ser muy drástico. Por el contrario, el despliegue frecuente pone en producción pequeños cambios, de forma continua

  - Esto familiariza a todo el equipo con el proceso de introducir novedades
  - Los cambios menores implican problemas potenciales menores
  - Hay técnicas más avanzadas (integración continua, entrega continua, despliegue continuo). Pero realizar al menos *despliegue frecuente* es prácticamente imprescindible dentro de la filosofía *DevOps*

Técnicas de despliegue (2)

- Conmutación de funciones

  Ejemplo: Ponemos en producción funcionalidad nueva. Pero si falla y decidimos desactivarla, se puede hacer con un conmutador sencillo desde el código. Sin necesidad de volver a desplegar el código *viejo*

    - Los contenedores pueden hacer innecesaria esta técnica

- *Dark Launching*

  Las nuevas versiones se aplican solo a unos pocos usuarios

    - Esto facilita la corrección de problemas y limita los problemas potenciales
    - Pueden ser los empleados, pueden ser voluntarios, pueden ser usuarios que hemos detectado como *avanzados* o pueden ser aleatorios

Técnicas de despliegue (3)

- *Blue Green Deployment*

  La versión nueva y la versión anterior se preparan para que funcionen en paralelo

    - Para conmutar de la *versión azul* a la *versión verde* no hay que cambiar el código, solo el router/el *balanceador* de carga o algún fichero de configuración

Integración, entrega y despliegue continuo

Las siguiente técnicas son habituales en *DevOps*, aunque

- No son imprescindibles para hacer *DevOps*
- Implementarlas no significa estar haciendo *DevOps*

En cualquier entorno de desarrollo moderno de cierto tamaño, hay varios desarrolladores, utilizando un sistema de control de versiones. Cada uno tiene su copia de trabajo del software, que con cierta periodicidad, integra en el repositorio principal

- *Continuous Integration* (CI)

  Realizar esta integración muy a menudo. Típicamente varias veces al día

- *Continuous Delivery* (CD)

  No solamente hacer *Continuous Integration*, sino dar un paso más allá. Además de integrar el código, asegurarse de que está listo para ponerse en producción muy a menudo. Esto es, pasar los controles de calidad y automatizar la puesta en producción. Tal vez no tan a menudo como la CI, pero sí muy a menudo. P.e. una vez al día.

- *Continuous Deployment*

  No solo hacer *Continuous Delivery*, sino dar un paso más allá. Además de asegurarse de que el código tiene calidad como para ponerse en producción, ponerlo realmente en producción

14.6. Herramientas

Herramientas

Las siguientes herramientas son habituales cuando se siguen los principios *DevOps*

- Contenedores Docker

  https://gsyc.urjc.es/~mortuno/lagrs/02-virtualizacion_I.pdf

  https://gsyc.urjc.es/~mortuno/lagrs/02-virtualizacion_III.pdf

- Ansible
- Jenkins
- Vagrant

  https://gsyc.urjc.es/~mortuno/lagrs/vagrant.pdf

Jenkins

`https://jenkins.io`

Jenkins es una herramienta para implementar Integración Continua (C.I.)

- Aparece en el año 2011. Es software libre, muy popular
- Aplicación basada en web
- Va un paso más allá de herramientas como Maven, que construyen el fuente pero no hacen C.I.
- Su entorno nativo es Java, pero tiene *plugins* para distintos lenguajes, herramientas de control de versiones, de virtualización, de testing, de comunicación con el personal, etc

- La unidad principal es el *build*: el conjunto de pasos para desplegar una aplicación software
    - Un *build* se pueden disparar manualmente, por un commit, o con planificación periódica similar a cron
    - Un *build* se organiza en *pipelines*, cada una compuesta de *steps*

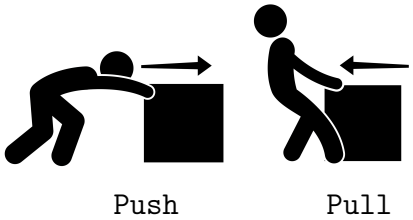
```
Jenkinsfile (Declarative Pipeline)
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying'
            }
        }
    }
}
```

Ansible

`https://www.ansible.com`
Ansible es un software de gestión de configuraciones

- Creado por Michael DeHaan en 2012. En la actualidad pertenece a RedHat
- software libre, muy popular
- Arquitectura cliente servidor
- Los clientes se denominan *nodos*. Son las máquinas controladas. Solo necesitan un servidor de ssh, por tanto soporta cualquier Linux, Unix, macOS. También funciona sobre la PowerShell de Microsoft Windows

- El servidor se denomina *controlling machine*. Es la máquina que administra y controla los nodos. El soporte nativo es para Linux. Hay versiones para macOS y otras plataformas, prácticamente cualquiera donde funcione python y pip



Iconos: www.vecteezy.com

- Ansible sigue un paradigma *push*: el controlador envía las órdenes
- Otras herramientas similares como *puppet* tienen un enfoque *pull*, que resulta más complejo: la máquina administrada corre un demonio que, periódicamente, se *trae* las órdenes

Conceptos principales en Ansible

- *Inventory*

  Fichero que contiene el listado de las máquinas administradas, con su nombre y/o dirección IP, puerto, claves ssh, etc

- *Playbook*

  Es una especie de *howto* automatizable, un script de propósito específica (configurar una máquina), de alto nivel y fácilmente legible por humanos

  - Palabra inglesa que significa libro de juego, libro de tácticas o libro de reglas
  - Escrito en formato YAML, similar a JSON pero con sintaxis pensada para que sea cómodo para las personas. Filosofía análoga al formato *markdown*, pero para datos, no para texto

- *Role*

  Estructura de nivel superior al *playbook*. Permite hacer plantillas de *playbooks*. Está formado por *playbooks*, ficheros, dependencias entre *playbooks*...

Ansible Galaxy

Repositorio centralizado de roles. Facilita la instalación de software. Equivalente a tener un repositorio de libros de instrucciones, pero que se autoejecutan

`https://galaxy.ansible.com`

```
# Configurar un servidor web básico con  nginx
- name: Configurar servidor web con nginx
  hosts: miServidor01
  sudo: yes
  tasks:
    - name: install nginx
      apt: name=nginx update_cache=yes

    - name: copy nginx conf file
      copy: src=files/nginx.conf dest=/etc/nginx/nginx.conf

    - name: copy nginx server file
      copy: src=files/server.conf dest=/etc/nginx/sites-available/default

    - name: enable config
      file: >
        dest=/etc/nginx/conf.d/default
        src=/etc/nginx/sites-available/default
        state=link
    - name: copy index.html
      template: src=templates/index.html.j2 dest=/usr/share/nginx/html/index.html mode=0644

    - name: restart nginx
      service: name=nginx state=restarted enabled=yes
```

- El guión denota un elemento de una lista (una tarea, (*task*)
- Cada tarea suele estar compuesta por varias líneas.

  La primera suele ser el nombretiene un nombre (opcional)

  A continuación, la orden

  p.e.

  `apt: name=nginx update_cache=yes`

  ejecutará en cada nodo

  `apt update; apt install -y nginx`

Referencias

[1] *DevOps for Developers*

Michael Huttermann. Ed. Apress, 2012

http://proquest.safaribooksonline.com/book/software-engineering-and-development/9781430245698

[2] *Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale*

Jennifer Davis, Katherine Daniels. Ed. O'Reilly, 2016

http://proquest.safaribooksonline.com/book/software-engineering-and-development/9781491926291

[3] *DevOps for Web Development*

Mitesh Soni. Ed. Pack, 2016.

http://proquest.safaribooksonline.com/book/web-design-and-development/9781786465702

[4] *DevOps: A Software Architect's Perspective*

Len Bass, Ingo Weber, Liming Zhu. Ed. Pearson, 2015

http://proquest.safaribooksonline.com/book/software-engineering-and-development/9780134049885

[5] *Kanban in Action*

Marcus Hammarberg, Joakim Sunden. Ed. Manning, 2014

http://proquest.safaribooksonline.com/book/software-engineering-and-development/agile-development/9781617291050

[6] *The Elements of Scrum*

Chris Sims, Hillary Louise Johnson. Ed. Dymaxicon, 2011

15. Administración de servicios

15.1. Empaquetado de ficheros

Empaquetado de ficheros

Almacenar varios ficheros en uno solo, no necesariamente con compresión
Utilidad:

- Más cómodo de manejar (copiar, enviar por correo, etc)
- Conservar metainformación (permisos) o incluso mayúsculas/minúsculas, tildes, etc si los ficheros van a pasar por un sistema de ficheros diferente
  - ISO9660 (cdrom)
  - vfat (Windows, discos externos, pendrives)
  - ntfs (Windows)

gzip

Comprime o descomprime 1 fichero
Extensión: `fichero.z` `fichero.gz`

- Comprimir y descomprimir (borrando el original):

  ```
  gzip fichero
  gunzip fichero.gz
  ```

- Comprimir y descomprimir (manteniendo el original):

  ```
  gzip -c fichero > fichero.gz
  zcat fichero.gz > fichero
  zcat fichero.gz  | less
  ```

Empaquetado sin compresión. Varios ficheros en un fichero `.tar`

- Empaquetar:
  `tar -cvf fichero.tar fichero1 fichero2`
- Desempaquetar:
  `tar -xvf fichero.tar`
- Mostrar contenido:
  `tar -tf fichero.tar`

Empaquetado con compresión. Varios ficheros comprimidos en un fichero `.tar.gz`. O lo que es lo mismo, un fichero `.tgz`

- Comprimir:
  `tar -cvzf fichero.tgz fichero1 fichero2`
- Descomprimir:
  `tar -xvzf fichero.tgz`
- Mostrar contenido:
  `tar -tzf fichero.tgz`

(Son las mismas opciones, pero añadiendo una z)

WinZip

- Por motivos de licencias, originalmente no había compresores para Linux. (Pero las aplicaciones Windows saben descomprimir descomprimir `.tgz`)
- Descomprimir: `unzip fichero.zip`

bz2

Formato que ofrece compresión más alta que .gz, (empleando más CPU y memoria)

- Comprimir y descomprimir 1 fichero, borrando el original

  `bzip2 fichero`

  `bunzip2 fichero.bz2`
- Comprimir y descomprimir 1 fichero, manteniendo el original

  `bzip2 -c fichero > fichero.bz2`

  `bunzip2 -c fichero.bz2 >  fichero`
- Comprimir y descomprimir varios ficheros, manteniendo el original

  `tar -c fichero1 fichero2  |  bzip2 > fichero.bz2`

  `tar -xjf fichero.bz2`

Fragmentación de ficheros

Si necesitas trocear una imagen de gran tamaño en ficheros que quepan en un *pendrive* o cdrom

- Empaquetar y comprimir un directorio:
  `tar -cvzf mi_imagen.tgz mi_directorio`
- Mostrar contenido:
  `tar -tzf mi_imagen.tgz`
- Trocear:
  `# tamaño fichero prefijo split -b 500MB mi_imagen.tgz mi_imagen.tgz.`
  (Observa que el segundo parámetro es igual al primero, pero añadiendo un punto)
- Habremos generado

  ```
  mi_imagen.tgz.aa  mi_imagen.tgz.ab  mi_imagen.tgz.ac
  ```

En la máquina destino (no importa si en el *host* el S.O. es distinto)

- Unir los fragmentos `cat mi_imagen.tgz.* > mi_imagen.tgz`

  (En MS Windows para este paso podemos emplear HjSplit, Free File Splitter o cualquier otro programa similar)
- Descomprimir y desempaquetar:
  `tar -xvzf mi_imagen.tgz`

  (En MS Windows podemos usar 7-Zip o similares)

15.2. Instalación de paquetes

Instalación de paquetes

- Método clásico para instalar programas:

  Formato .tgz

  Descomprimir y seguir las instrucciones del fichero README

  Suele ser del estilo de

  ```
  ./configure
  make compile
  make install
  ```

- Sistema de gestión de paquetes

  Colección de herramientas que automatizan la instalación, actualización y eliminación de programas.

- Gestión de paquetes, Debian y derivados

  Paquetes en formato `.deb`

  Se pueden manejar directamente con `dpkg`, o con `apt-get, apt, aptitude, dselect`, o `synaptic`

- Gestión de paquetes, RedHat y derivados

  Paquetes en formato `.rpm`

  Se pueden manejar directamente con `rpm`, o con `up2date` o `yum`

15.2.1. El sistema de paquetes de Debian

El sistema de paquetes de Debian

Los paquetes mantienen *dependencias* entre sí, de forma que la instalación de un paquete puede:

- *depender* de que se instale también otro
- *recomendar* que se instale también otro
- *sugerir* que se instale también otro
- *entrar en conflicto* con otro actualmente instalado

15.2.2. dpkg

dpkg

- Es la herramienta básica de gestión de paquetes, que es usada por las otras (dselect, apt-get, apt, aptitude, synaptic).
- Usos principales:
  - `dpkg -i paquete_VVV-RRR.deb`
    Instala un paquete
  - `dpkg -r paquete`
    Desinstala (*remove*) un paquete, elimina todo excepto los ficheros de configuración

- `dpkg -P paquete`
  Purga un paquete, eliminando incluso los ficheros de configuración

- Tiene muchas opciones. Puede esquivarse el esquema de dependencias (peligroso) con las opciones que empiezan por `--force-...`

Versiones de Ubuntu:

```
nombre año.mes

Warty Warthog 4.10      Hoary Hedgehog 5.04
...                     ...
Saucy Salamander 13.10  Trusty Tahr 14.04 LTS
Utopic Unicorn 14.10    Vivid Vervet 15.04
Wily Werewolf 15.10     Xenial Xerus 16.04 LTS
Yakketiy Yak 16.10      Zesty Zapus 17.04
Artful Aardvark 17.10   Bionic Beaver 18.04 LTS
Cosmic Cuttlefish 18.10 Disco Dingo 19.04
Eoan Ermine 19.10       Focal Fossa 20.04 LTS
Groovy Gorilla 20.10    Hirsute Hippo 21.04
Impish Indri 21.10      Jammy Jellyfish 22.04 LTS
Kinetic Kudu 22.10
```

Versión estándar: Desde 13.04, soportada durante 9 meses (18 meses en las versiones anteriores)

LTS: Long Term Support: soportada durante 3 años en escritorio y 5 en servidor

Ubuntu Desktop / Ubuntu Server Edition / Ubuntu Server Edition JeOS

Variantes de Ubuntu: Kubuntu, Xubuntu, Gobuntu, Ubuntu Studio

15.2.3. apt

apt

- La herramienta más sencilla de usar y más potente.
- Usa *repositorios*: sitios centralizados donde se almacenan paquetes
- Las direcciones de los repositorios se indican en el fichero
  `/etc/apt/sources.list`
- Los repositorios de ubuntu se dividen en 4 componentes
  1. *Main*. Soportado oficialmente por ubuntu. Libre
  2. *Restricted*. Soportado oficialmente. No libre
  3. *Universe*. No soportado oficialmente. Libre

4. *Multiverse*. No soportado oficialmente. No libre

Además, se pueden añadir componentes de terceros

```
# deb cdrom:[Ubuntu 6.06 _Dapper Drake_ - Release i386 (20060531)]/ dapper main restricted
deb http://archive.ubuntu.com/ubuntu edgy main restricted
deb http://security.ubuntu.com/ubuntu edgy-security main restricted
deb http://archive.ubuntu.com/ubuntu edgy-updates main restricted

## All community supported packages, including security- and other updates
deb http://archive.ubuntu.com/ubuntu edgy universe multiverse
deb http://security.ubuntu.com/ubuntu edgy-security universe multiverse
deb http://archive.ubuntu.com/ubuntu edgy-updates universe multiverse

# Google Picasa for Linux repository
deb http://dl.google.com/linux/deb/ stable non-free
```

Uso básico de apt

Desde línea de comandos se puede usar `apt-get`

- `apt-get update`

  Actualizar lista de paquetes:

- `apt-get upgrade`

  Actualizar todos los paquetes instalados a la última versión disponible (sin cambiar de distribución)

- `apt-get  install paquete`

  Instalar un paquete (resolviendo conflictos)

En 2014 aparece la herramienta `apt`, con la misma finalidad e interfaz de usuario, pero que resulta un poco más fácil de manejar porque unifica `apt-get` y `apt-cache`

```
apt update
apt upgrade
apt install paquete
```

Aunque indiquemos a nuestro sistema de paquetería que instale la última versión de un paquete, tal vez no sea posible. Se dice que el paquete está *retenido* (*hold*)

- El paquete depende de otro no incluido en la distribución actual
- El administrador lo ha retenido *a mano* (no le gusta, da problemas...)

```
sudo install feta
sudo feta hold nombre_del paquete
sudo feta unhold nombre_del paquete
```

- `apt remove paquete`

  Desinstala un paquete

- `apt purge paquete`

  Desinstala un paquete y borra su configuración

- `apt full-upgrade`

  Actualiza *agresivamente* todos los paquetes instalados, lo que puede incluir el paso a la versión más reciente de la distribución

Otros mandatos interesantes

En los repositorios hay muchos paquetes ¿Cómo saber cuál necesito?

- `apt search cadena`

  Buscar una cadena en el nombre o descripción de un paquete. Indica el estado del paquete (instalado, no instalado, borrado...)

- `apt show paquete`

  Muestra descripción del paquete

- `dpkg-reconfigure paquete`

  Reconfigurar un paquete

15.2.4. Sistemas de paquetes en macOS

Sistemas de paquetes en macOS

Apple no tiene previsto el uso de estos sistemas para usuarios *normales*. Pero sí son muy útiles para usuarios con perfil de desarrollador o administrador. Se usan prácticamente igual que apt-get

Actualmente podemos optar por tres sistemas

1. fink

   El más antiguo. Basado en apt-get. Poco usado hoy

2. macports

   Basado en los ports de FreeBSD. Muy completo. Muy independiente de Apple

3. homebrew

   El más moderno y mejor integrado con Apple. Tal vez el más popular hoy

15.2.5. Paquetes Snap

Recapitulación: ficheros tar

Como hemos visto, la forma clásica para distribuir aplicaciones, desde finales de los años 70 es el uso de ficheros .tar o tar.gz con el código fuente y tal vez un *Makefile*

- Exigen mucha intervención manual por parte del administrador de la máquina
- El número de dependencias es muy alto, incluyendo ¡todas las herramientas de compilación!

Recapitulación: paquetes tradicionales

Desde finales de los años 90, sistemas de gestión de paquetes (deb, rpm, brew, etc)

- Los responsables de cada distribución (Debian, Ubuntu, Fedora, Suse, etc) toman la versión original de cada aplicación (denominada *upstream*) y la preparan para que funcione en una versión concreta de una distribución concreta, con todas las dependencias necesarios.
- Ya no requiere trabajo por pate del administrador local, pero sí de los responsables de la distribución
- Para instalar un paquete, es necesario
  - Disponer de la versión exacta para la distribución exacta
  - Que ese paquete esté correctamente preparado
  - Que el estado de mis paquetes sea consistente (a veces hay conflictos, paquetes mal instalados, etc)

Paquetes Snap

En 2014 Canonical desarrolla los paquetes Snap, que están ganando popularidad recientemente

- Un único paquete Snap incluye todo lo necesario para que funcione la aplicación, con todas sus dependencias en cualquier distribución Linux, sin necesidad de adaptar nada
- Un paquete Snap incluye un fichero que se monta en el sistema de ficheros, que se descomprime sobre la marcha
- Cada aplicación se actualiza automáticamente
- La aplicación se ejecuta de forma aislada dentro de un *sandbox*, con acceso limitado al *host*

Inconvenientes de snap:

- Consume más recursos que un paquete nativo
- La integración con el resto del sistema puede ser peor
- El tiempo necesario para comenzar a ejecutar la aplicación suele ser relativamente alto
- Aunque se pueden usar en la mayoría de distribuciones Linux, están muy vinculados con Ubuntu

Hay sistemas similares, alternativos, como *Flatpak* y *AppImage*

Uso de Snap

Podemos consultar todos los *snaps* disponibles en la *Snap Store*
https://snapcraft.io
El manejo es muy sencillo

```
snap find <palabras clave>
sudo snap install <paquete>
snap list
sudo snap remove <paquete>
```

Más información:
https://snapcraft.io/docs/getting-started

15.3. Búsqueda de ficheros

Localizar ficheros

- `find` Busca un fichero
  `find . | grep fichero` Filtra la búsqueda
- `locate` Busca un fichero (en una base de datos)
- `updatedb` Actualiza la base de datos

15.4. Hora. Parada del sistema

Hora. Parada del sistema

- `shutdown -P now` ≡ `poweroff`

  Apaga el sistema
- `shutdown -r now` ≡ `reboot`

  Reinicia el sistema
- `sleep n`

  Duerme la shell segundos
- `sleep 28800 ; halt`
  Detiene la máquina al cabo de 8 horas
- Poner fecha y hora:
  - Automáticamente: Demonio *ntpd*, cliente de *Network Time Protocol*
  - Manualmente
    `date -s AAAA-MM-DD`
    `date -s HH:MM`

Casi siempre hay varias soluciones para una tarea. Generales o particulares

- `find . |grep cadena`

  `find . -name cadena\*`
- `sleep 60 | shutdown -h now`

  `shutdown -h 1`

- etc, etc

Todas sirven. ¿No? ¿Cuál es *mejor*?

- Cuando somos novatos en un sistema, con una solución general sabremos resolver ese problema y otros parecidos
- Cuando conocemos mejor un sistema y dominamos las soluciones generales, las soluciones particulares suelen ser más eficientes

15.5. Copias de seguridad

Copias de seguridad

- `tar` o similares

  Problema: Siempre se duplican los directorios enteros

- `rsync`

  Mirror unidireccional. Permite mantener una réplica de un directorio. Solo se actualizan las novedades. No permite modificar la réplica

- `FreeFileSync, Synkron`

  Herramientas libres para sincronización bidireccional (Windows, Linux, OS X).

  Sincronizan dos (o más) directorios: Cualquiera de los dos directorios puede modificarse

- Sistemas de almacenamiento permanente

  `Time Machine` (OS X)

  `dumpfs` (bsd)

  `pdumpfs` (Linux, Windows)

  `TimeVault, FlyBack` (Linux)

  `venti` (Plan 9)

  - Se registran los cambios en los ficheros, sin borrar nunca nada
  - Mantienen una *foto* del estado diario del sistema de ficheros, en un directorio con formato yyyy/mm/dd
  - Parece mucho, pero hoy el almacenamiento es muy barato. P.e. si generamos 10 Mb diarios, necesitamos unos 4Gb anuales

15.6. Administración de los demonios

Administración de los demonios

Los demonios son programas relativamente *normales*, con algunas particularidades

- Ofrecen servicios (impresión, red, ejecución periódica de tareas, logs, etc)
- Suelen estar creados por el proceso de arranque *init* (ppid=1)
- Sus nombres suelen acaban en *d*
- Se ejecutan en *background*
- No están asociados a un usuario en una terminal
- El grueso de su configuración suele hacerse desde un único fichero

  En el caso de debian, `/etc/midemonio.conf`
- Se inician y se detienen de manera uniforme

Unix System V

Versión de Unix comercializada en 1983 por AT&T

- La mayoría de los Unix, incluyendo Linux, son derivados de System V
- Otros Unix son derivados del Unix BSD de aquella época: esto incluye los BSD actuales y OS X (Apple)

System V introduce una forma de organizar los demonios basada en

- *Niveles de ejecución*
- Scripts en `/etc/init.d`
- Ordenación lineal de sucesos:

  Las tareas se ordenan secuencialmente en orden preestablecido, solo cuando una está completamente acabada empieza la siguiente

Systemd

15.6.1. Systemd

- El sistema de arranque tradicional de Linux (System V) no es adecuado para las máquinas actuales
  - Son externos: aparecen y desaparecen
  - Están en red
  - Ahorran energía
  - ...
- *Systemd* es un sistema de arranque basado en eventos (pueden suceder en cualquier orden, puede haber tareas en paralelo)
- Proporciona los comandos *systemctl* para iniciar y detener los demonios
- Mantiene una capa adicional de software para que las órdenes al estilo System V sigan funcionando
- El desarrollo de Systemd lo compienza Red Hat en el año 2010
- En Ubuntu solamente se utiliza desde la versión 15.04 (año 2015). Anteriormente empleaba *upstart*, un sistema similar, también compatible con System V

Ficheros de configuración

Los ficheros donde se configura un demonio son:

- Fichero principal de configuración

  `/etc/midemonio.conf`
- Configuración de puesta en marcha y parada
  - System V

    `/etc/init.d/midemonio`
  - Systemd

    `/etc/init/midemonio.conf`
- Configuración del administrador local

  `/etc/default/midemonio`

  Solo existe en Debian y derivados (también en Ubuntu con Upstart). No lo usan todos los paquetes

Directorio /etc/default

- La inmensa mayoría de los parámetros de `/etc/midemonio.conf` y de `/etc/init.d/midemonio` o de `/etc/init/midemonio.conf` los ha escrito el desarrollador del demonio o el empaquetador de la distribución, es normal que el administrador local de cada máquina concreta solo modifique unos pocos
- En algún momento habrá que actualizar el demonio a una versión nueva, que frecuentemente incluirá cambios en sus ficheros de configuración, escritos por el desarrollador o el empaquetador
  - ¿Instalamos los ficheros nuevos y *machacamos* los viejos?

    Problema: se pierde la configuración que ha personalizado el administrador local
  - ¿Mantenemos los viejos y descartamos los nuevos?

    Problema: se pierden los cambios de la versión actual, el fichero de configuración (antiguo) podría incluso se incompatible con el demonio (actual)

Solución: fichero `/etc/default/midemonio`

- Es un fichero muy corto, con muy pocos parámetros, muy importantes, que se sabe que serán modificados por el administrador local
- Cuando se instalan versiones nuevas del demonio, este fichero se mantiene
- Los cambios introducidos por las nuevas versiones de los demonios estarán en

  `/etc/midemonio.conf` o en `/etc/init.d/midemonio` o `/etc/init/midemonio.conf`

Administración estilo System V

Método clásico, obsoleto aunque sigue funcionando. El código de un demonio puede estar en cualquier lugar del sistema de ficheros. Pero siempre se coloca en `/etc/init.d/midemonio` un script para manejarlo

- `/etc/init.d/midemonio start`

  Inicia el servicio

- `/etc/init.d/midemonio stop`

  Detiene el servicio

- `/etc/init.d/midemonio restart`

  Detiene e inicia el servicio. Suele ser **necesario para releer los ficheros de configuración** si se han modificado (`/etc/midemonio.conf`)

Con frecuencia también está disponible

- `/etc/init.d/midemonio reload`

  Lee el fichero de configuración sin detener el servicio

Administración con systemctl

Método moderno, preferible

- `systemctl start midemonio`
- `systemctl stop midemonio`
- `systemctl restart midemonio`

  Detiene e inicia

- `systemctl reload midemonio`

  Lee fichero de configuración sin detener el servicio

Naturalmente, todas estas órdenes necesitan privilegios de root

15.6.2. Niveles de ejecución

Niveles de ejecución

¿Qué demonios se ponen en marcha cuando se inicia el sistema?

Un Nivel de ejecución (*runlevel*) es una configuración de arranque. Para cada nivel, se define un conjunto de demonios que deben ejecutarse

- Este es un concepto de System V, en *Systemd* se usan *targets*, que son equivalentes

Supongamos una fábrica. Diferentes niveles (estados), no secuenciales. Al entrar en un nivel se apagan ciertos sistemas y se encienden otros

```
Nivel 1 - Noche
    Al entrar en este nivel apagar
        01 motores
        02 luces principales
    Al entrar en este nivel encender
        01 alarma
        02 luces_auxiliares
Nivel 2 - Producción normal
    Al entrar en este nivel apagar
        01 alarma
        02 luces auxiliares
    Al entrar en este nivel encender
        ....
Nivel 3-  Mantenimiento
    Al entrar en este nivel apagar
        01 motores
        ....
```

El responsable de conectar y desconectarlo todo será el vigilante de seguridad, así que hay que dejarle unas instrucciones muy claras:

`ordinal` `[encender|apagar]` `nombre_del_sistema`

Código	**Significado**	**Mandato**
S10motor-ppal	$1^{\underline{o}}$ encender motor principal	`/etc/init.d/motor-ppal start`
S20motor-aux	$2^{\underline{o}}$ encender motor auxiliar	`/etc/init.d/motor-aux start`
K10alarma	$1^{\underline{o}}$ apagar alarma	`/etc/init.d/alarma stop`

Dentro de cada nivel, las tareas se ordenan desde 00 hasta 99 (con un cero a la izquierda para los valores del 0 al 9)

El *vigilante de seguridad* es el proceso `init`.

Hay un directorio por nivel:
Debian y derivados

```
/etc/rc0.d
/etc/rc1.d
...
```

Red Hat y derivados

```
/etc/rc.d/rc0.d
/etc/rc.d/rc1.d
...
```

Hay otro directorio cuyos servicios se activan siempre, en cualquier nivel

```
/etc/rcS.d
```

Dentro de los directorios hay enlaces simbólicos

- Apuntan al script en `/etc/init.d` que controla el demonio
- Cada nombre del enlace indica conexión/desconexión, ordinal y script a manejar

Cuando entra en el nivel N, el proceso init se encarga de

- Ejecutar por orden todos los scripts en `/etc/rcN.d` que empiezen por K (de Kill). Les pasa el parámetro *stop*
- A continuación, ejecuta por orden todos los scripts en `/etc/rcN.d` que empiezen por S (de Start). Les pasa el parámetro *start*

`who -r`

Indica el nivel de ejecución actual

```
0   Halt (Parada del sistema)
1   Modo monousuario, usuario root, sin red
2-4 Diversos modos multiusuario, sin gráficos
5   Modo multiusuario completo, con X Window
6   Reboot (Reiniciar el sistema)
```

Ejemplo del contenido de `/etc/rc2.d/`

```
S10acpid            S18hplip         S20postfix        S89atd
S10powernowd.early  S19cupsys        S20powernowd      S89cron
S10sysklogd         S20apmd          S20rsync          S90binfmt-support
S10wacom-tools      S20festival      S20ssh            S98usplash
```

Ejemplo del contenido de `/etc/rc6.d/`

```
K19cupsys       K25mdadm            S15wpa-ifupdown     S50lvm
K19setserial    K25nfs-user-server  S20sendsigs         S50mdadm-raid
K20dbus         K30etc-setserial    S30urandom          S60umountroot
K20laptop-mode  K50alsa-utils       S31umountnfs.sh     S90halt
```

Resumiendo, para ejecutar automáticamente un demonio manejamos 3 ficheros

- El fichero con el ejecutable del demonio

  p.e.

  `/usr/sbin/sshd`

  - Si se trata de un servicio que no es estándar en la distribución, su sitio es el directorio `/usr/local`

- El script que maneja el demonio

  - Acepta los parámetros `start, stop, reload, ...` y llama el demonio en consecuencia

p.e.

`/etc/init.d/ssh`

- El enlace, dentro del directorio correspondiente al nivel, que apunta al script

  p.e.

  `/etc/rc5.d/S02ssh`

  apuntando a

  `/etc/init.d/ssh`

15.7. Consulta de logs

Los demonios no muestran información ni en la consola ni en ninguna aplicación gráfica

- Los demonios usan el demonio *syslogd* o *sysklogd* para notificar y almacenar información relevante: inicio, parada, estado, peticiones, respuestas, errores, etc

  A partir del año 2009 es más habitual emplear *rsyslogd*, muy similar a *syslogd*

- Todo esto se escribe en diversos ficheros de texto, siendo los más interesantes

  - `/var/log/syslog`
    Información general del sistema
  - `/var/log/auth.log`
    Información sobre autenticación de usuarios

- Podemos ver un fichero cualquiera, p.e. un log, con *cat*

  ```
  cat /var/log/syslog
  ```

  (Muestra un fichero entero)

- Es más práctico usar *tail*

  ```
  tail -20 /var/log/syslog
  ```

(Muestra las últimas 20 línea)

`tail /var/log/syslog`

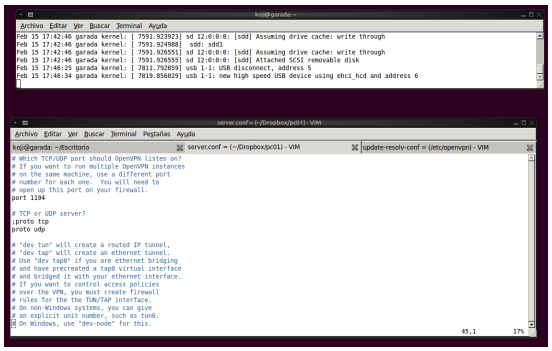
(Muestra las últimas 10 línea)

O mejor aún, si queremos monitorizar continuamente un log, abrimos un terminal exclusivamente para esto y ejecutamos

- `tail -f /var/log/syslog`

(Muestra las últimas líneas, se queda esperando a que haya novedades en el fichero, y cuando las hay, las muestra también)

Si estamos depurando un servicio y tenemos una sesión gráfica, puede ser útil esta disposición del escritorio: dejar un terminal siempre visible, ejecutando `tail -f`, y trabajar en otro terminal con varias pestañas



Si no tenemos gráficos, podemos pulsar Alt F2, Alt F3, etc, abrir una sesión y dedicarla a los logs (también dentro de VirtualBox)

15.8. Tareas periódicas: cron

Tareas periódicas

15.8.1. Tareas periódicas

- Automatizan la gestión del sistema
- Fiabilidad. Protegen frente a olvidos
- Se ejecutan en el momento preciso (día y hora)

- Ayudan o detectan situaciones de error
- Facilitan el control del sistema
- Programas:
  - `cron`
  - `anacron`. Permite ejecutar algo programado para un momento en que el sistema estaba apagado
  - `at`. Ejecuta una tarea a la hora indicada por *stdin*

Usos de las tareas periódicas

- Generación de informes periódicos (fin de mes, etc.)
- Estado de las comunicaciones
- Borrado de ficheros temporales (`/tmp`, `/var/tmp`)
- Tareas de respaldo de información
- Control de los procesos presentes en el sistema
- Parada del sistema según horarios de trabajo
- Recordatorios
- Descarga de *software* en horarios de poco tráfico

cron

- Es uno de los demonios esenciales de un sistema, siempre está arrancado (`/usr/sbin/cron`)
- Se encarga de ejecutar tareas programadas para un determinado momento, bajo la identidad del usuario que lo programó y con precisión de 1 minuto
- Se controla a través del uso de determinados ficheros de configuración (solo para el superusuario) y mediante el uso de la orden "`crontab`" (para todos los usuarios).

15.8.2. Tabla de cron

```
SHELL=/bin/bash
PATH=/usr/local/bin:/usr/bin:/bin
#   m    h   dayofmonth  month  dow   command
   16    *       *        *     *    ping 193.147.71.119  -c 1
   0     9       4        8     *    comando_ejemplo_1
   0   15,18     *        *    1-5   comando_ejemplo_2
```

m: Minuto. De 0 a 59
h: Hora. De 0 a 23
dayofmonth: de 0 a 31
month: de 1 a 12
dayofweek: de 0 a 7. 0=7=domingo, 1=lunes, 2=martes...

Cada línea es una tarea

- Se pueden poner comentarios con `#` pero no en cualquier posición, solo siguiendo el patrón *principio de línea, 0 o más espacios, almohadilla*
- En las asignaciones `variable=valor`, el valor no se expande. Todo lo que hay a la derecha del igual se toma literalmente. Por tanto, no pueden hacerse cosas como p.e. `PATH=$HOME/bin:$PATH`
- Se puede usar la variable de entorno `$PATH` en un comando. Ejemplo:

```
0   18        *        *     *   $HOME/blabla/bla.py
```

- Es necesario dejar una linea en blanco al final de la tabla

```
*          -> todos
1-4        -> 1,2,3 y 4
1,4        -> 1 y 4
*/3        -> cada 3
1-15/3     -> los primeros 15, cada 3
```

Ejemplos y contraejemplos:

```
#   m    h   dayofmonth  month  dow   command
   *   14-15     *        *     *    comando_ejemplo_3
   *   23-7      *        *     *    comando_ejemplo_4
```

Atención:

- `comando_ejemplo_3` no se se ejecutará de 14:00 a 15:00 sino de 14:00 a 15:59
- `comando_ejemplo_4` tiene un rango incorrecto ($23 > 7$)

- `crontab -e`
  Edita la tabla de cron del usuario. Usa el editor por omisión (normalmente vim). Podemos usar otro cambiando la variable de entorno EDITOR

- `crontab -l`
  Muestra tabla de cron

- `crontab mi_tabla`
  El fichero *mi_tabla* pasa a ser nueva tabla de cron

Ambigüedades en la especificación del momento de ejecución

- El día en el que se ejecuta cada orden se puede indicar de 2 maneras:

  - día del mes (3er campo)
  - día de la semana (5° campo)

  En caso de aparecer los dos campos (esto es, que ninguno es "`*`"), la interpretación que hace `cron` es que la orden debe ejecutarse cuando se cumpla *cualquiera* de ellos

  Ejemplo:
  `0,30 * 13 * 5 echo 'Viernes 13!' | wall`
  (ejecuta la orden cada media hora, todos los viernes y además todos los días 13 de cada mes)

Momentos "especiales" (solo Linux)

En lugar de especificar los 5 primeros campos, se puede usar una cadena de las siguientes:

- `@reboot`: Se ejecuta al iniciarse la máquina.

- `@yearly`: Se ejecuta una vez al año.

- `@monthly`: Se ejecuta una vez al mes.

- `@weekly`: Se ejecuta una vez por semana.

- `@daily`: Se ejecuta una vez al día.

- `@hourly`: Se ejecuta una vez por hora.

Entorno de ejecución de las tareas

- Cada tarea de cron se ejecuta por una *shell* `/bin/sh`. (a menos que definamos otra cosa en SHELL)
- Causa de **errores frecuentes:** El PATH con el que cron busca el mandato no es el del usuario, sino `/usr/bin:/bin`. Soluciones:
  - Indicar PATH en la tabla
  - Especificar el path abosoluto del mandato (p.e. `/usr/local/bin/mimandato`)
- Quien ejecuta las tareas no es el dueño de la tabla, sino cron. Aunque emplea algunas variables de entorno del dueño de la tabla, como `LOGNAME` y `HOME`.
- La entrada estándar de cada tarea se redirige de `/dev/null`, la salida estándar y la de error se envían por correo electrónico al propietario de la tarea (si hay servidor de correo)

16. OpenSSH

OpenSSH

- PGP: Pretty Good Privacy. Software criptográfico creado por Phil Zimmermann, año 1991. Base de la norma Open PGP.
- GPG: GNU Privacy Guard. Herramienta para cifrado y firmas digitales, que reemplaza a PGP
- Se puede emplear algoritmos como RSA o ed25519 (algo más moderno), ambos se consideran seguros. El algoritmo DSA ya no es recomendable
- Las distribuciones orientadas a sistemas empotrados no suelen usar OpenSSH sino Dropbear, un cliente y un servidor de ssh, compatible con OpenSSH, más ligero

16.1. Criptografía de clave pública

Criptografía de clave pública

Aparece con el algoritmo Diffie-Hellman, año 1976

- Clave de cifrado o pública E y de descifrado o privada D distintas (asimétricas)
- $D(E(P)) = P$
- Muy dificil romper el sistema (p.e. obtener D) teniendo E.
- Permite intercambiar claves por canal no seguro
- La clave privada sirve para descifrar. Debe mantenerse en secreto
- La clave pública sirve para cifrar. Puede conocerla todo el mundo (lo importante es que se conozca la clave correcta)
- Conociendo la clave pública de alguien, podemos cifrar un mensaje que solo él, con su clave privada, podrá descifrar
- Los algoritmos de clave pública son mucho más lentos que los de clave secreta (100 a 1000 veces). Por eso se suelen usar sólo para el intercambio de claves simétricas de sesión

- También sirve para autenticar (como en OpenSSH)

  Queremos desde una sesión en una máquina local, abrir otra sesión en una máquina remota sin volver a teclear contraseña

  - Una máquina remota, no fiable, contiene clave pública
  - Máquina local, fiable, contiene la clave privada
  - La máquina remota envía un reto cifrado con la clave pública, si la máquina local lo descifra, el usuario queda autenticado y puede abrir sesión en la máquina remota sin teclear contraseña

16.2. Uso de OpenSSH

Uso de OpenSSH

`ssh usuario@maquina`

Abre una sesión remota mediante una conexión segura en la máquina indicada, con el usuario indicado.

- La primera vez que abrimos una sesión en una máquina, ssh nos indica la huella digital de la máquina remota

  ```
  The authenticity of host 'gamma23 (212.128.4.133)' can't be established.
  RSA key fingerprint is de:fa:e1:02:dc:12:8d:ab:a8:79:8e:8f:c9:7d:99:eb.
  Are you sure you want to continue connecting (yes/no)?
  ```

- Si necesitamos la certeza absoluta de que esta máquina es quien dice ser, deberíamos comprobar esta huella digital por un medio seguro, alternativo

- El cliente ssh almacena las huellas digitales de las máquinas en las que ha abierto sesión en el fichero `~/.ssh/known_hosts`

- El servidor almacena su propia huella digital en los ficheros

  ```
  /etc/ssh/ssh_host_rsa_key
  /etc/ssh/ssh_host_ed25519_key
  ```

  Si la huella que tiene el host en la actualidad no coincide con la huella que tenía el host en la primera conexión, ssh mostrará un error

  Esto puede suceder porque

- Alguien esté suplantando la identidad del host
- El host ha sido reinstalado y el administrado no ha conservado estos ficheros

```
ssh -C -X usuario@maquina
```

- La opción `-X` (mayúscula) redirige la salida del cliente X Window de la máquina remota al servidor X Window de la máquina local

  Esto permite lanzar aplicaciones gráficas en la máquina remota, usarán la pantalla local

  Es necesario

  - `X11Forwarding yes`

    en `/etc/ssh/sshd_config` en la máquina remota
  - Que la máquina local admita conexiones entrantes

- La opción `-C` (mayúscula) comprime el tráfico. En conexiones rápidas es conveniente omitir esta opción

Además de para abrir una sesión en una máquina remota, ssh permite la ejecución de una única orden en la máquina remota

- `ssh jperez@alpha ls`

  Ejecuta `ls` en la máquina remota. Muestra en la máquina local el stdout.

- `ssh jperez@alpha 'echo hola > /tmp/prueba'`

  Ejecuta en alpha

  ```
  echo hola > /tmp/prueba
  ```

- `ssh jperez@alpha "echo $HOSTNAME > /tmp/prueba"`

  Al poner comilla doble, la variable se expande en la máquina local. La orden completa, redirección incluida, se ejecuta en la máquina remota

- `ssh jperez@alpha echo $HOSTNAME > /tmp/prueba`

  Al no poner comilla, la variable se expande en la máquina local. El resultado se redirige al fichero `/tmp/prueba` de la máquina local

- `ssh jperez@alpha 'echo $HOSTNAME > /tmp/prueba'`

  Al poner comilla simple y recta, la variable se expande en la máquina remota

Generación de claves

Para evitar teclear contraseña en cada ssh, podemos autentificarnos con claves asimétricas

Una vez configurado para ssh, también queda configurado para los servicios que corren sobre este (scp, sshfs)

- Se generan con `ssh-keygen`
- Se puede añadir una *pass phrase*. Es una contraseña adicional, tradicional. Pero no viaja por la red. Equivalente a la llave del armario de las llaves

Un usuario genera sus claves ejecutando en su *home* (de la máquina local) la orden *ssh-keygen*

- rsa:

```
orden para generar las claves:
    ssh-keygen -t rsa
fichero donde quedará (por omisión) la clave privada:
    ~/.ssh/id_rsa
fichero donde quedará (por omisión) la  clave pública:
    ~/.ssh/id_rsa.pub
```

- ed25519:

```
orden generar las claves:
    ssh-keygen -t ed25519
fichero donde quedará (por omisión) la clave privada:
   ~/.ssh/id_ed25519
fichero donde quedará (por omisión) la clave publica:
   ~/.ssh/id_ed25519.pub
```

Para poder entrar en máquina remota sin emplear contraseña, llevamos la clave pública a la máquina remota, y la escribimos en el fichero

- `~/.ssh/authorized_keys` (Redhat, Debian)
- `/etc/dropbear/authorized_keys` (OpenWrt)

Este fichero (de la máquina remota) en principio no existe

- La primera vez que añadamos una clave, podemos renombrar el fichero con la clave pública para que pase a llamarse `authorized_keys`
- Si posteriormente añadimos otras claves públicas, las pegamos inmediatamente después de las que ya existan, usando un editor de texto o una redirección de la shell

Permisos

Es necesario que el directorio `~/.ssh` (local y remoto):

- Tenga el dueño y el grupo del usuario
- Tenga permisos 700
- Contenga todos sus ficheros con permisos 600
- Todos sus ficheros pertenezcan al usuario y tengan como grupo el del usuario

Es necesario que en mi *home* solo yo pueda escribir

- En OpenWrt, también es necesario que `/etc/dropbear/authorized_keys` tenga permisos 600
- En Docker, en la máquina donde corre el servidor es necesario configurar el demonio:

  ```
  echo "IdentityFile ~/.ssh/id_ed25519" >> /etc/ssh/ssh_config
  ```

  O bien

  ```
  echo "IdentityFile ~/.ssh/id_rsa" >> /etc/ssh/ssh_config
  ```

ssh-copy-id

Disponemos además de de la orden `ssh-copy-id`, que se encarga, automáticamente, de:

- Copiar la clave pública a la máquina remota
- Crear `authorized_keys` si no existe
- Cambiar todos los permisos

```
ssh-copy-id [-i [identity_file]] [user@]machine
```

Ejemplo:

```
ssh-copy-id  jperez@alpha01
```

Atención: antes de usar este *asistente*, asegúrate de que entiendes el proceso y sabes realizar todos los pasos anteriores también *a mano*

Ejemplo: Configuración típica

Una clave privada distinta para cada uno de mis ordenadores

- Soy jperez, a veces trabajo localmente en pc-casa, a veces trabajo localmente en pc-oficina
- Desde ambos sitios quiero entrar en pc-remoto
- Desde casa entro en la oficina
- Desde la oficina, entro en casa
- Creo una clave privada jperez@pc-casa
- Creo una clave privada jperez@pc-oficina
- En el `authorized_keys` de pc-remoto:

  concateno claves públicas de jperez@pc-casa y jperez@pc-oficina
- En el `authorized_keys` de pc-casa

  Escribo la clave pública de jperez@pc-oficina
- En el `authorized_keys` de pc-oficina

  Escribo la clave pública de jperez@pc-casa

Ejemplo: Configuración alternativa

La misma clave para todos mis ordenadores

Aunque a las claves se les pone por omisión una etiqueta usuario@máquina (que aparece como comentario al final de la clave), solo es un comentario orientativo, una misma clave privada puede usarse desde distintas máquinas

- Creo una clave privada jperez, y la copio en pc-casa y en pc-oficina
- En el `authorized_keys` de pc-casa, de pc-oficina y de pc-remoto escribo la clave pública de jperez

Este enfoque es menos flexible y menos seguro

Es posible usar varias claves privadas (cada una en su fichero), basta indicar a ssh cuál (o cuáles) debe emplear

```
ssh jperez@alpha
    # intenta autenticarse con  ~/.ssh/id_rsa o ~/.ssh/id_ed25519
    # (clave por omisión)

ssh -i ~/.ssh/id_alumno alumno@pc01   #
    # lo intenta con id_alumno y con la clave por omisión

ssh -i ~/.ssh/id_alumno -i ~/.ssh/id_profe alumno@pc01
    # lo intenta con id_alumno, con id_profe  y con la clave por omisión
```

scp también admite la opción -i

```
scp -i ~/.ssh/id_alumno alumno@pc01:/tmp/test .
```

(sshfs no admite la opción -i)

ssh-agent

La manera habitual de autentificarse es mediante el demonio *ssh-agent*

- *ssh-agent* contestará por nosotros, gestionando retos y repuestas cifrados
- *ssh-agent* tiene que ser el padre de nuestra shell, o nuestra sesión x
  - Las distribuciones Linux con X Window suelen tenerlo instalado
  - Si no está funcionando (como en ubuntu server)
    `exec ssh-agent /bin/bash`
    Esto hace que nuestra shell actual sea reemplazada por *ssh-agent*, quien a su vez creará una shell hija suya
- `ssh-add` añade una identidad a ssh-agent

- `ssh-add -l` indica las identidades manejadas por el ssh-agent

En ocasiones, por ejemplo si no empleamos *pass phrase*, el *ssh-agent* no es necesario

Depuración

- En el cliente:

  `ssh -v  o  ssh -vv  o -vvv`

- En el servidor:

  `/var/log/auth.log`

Los errores más frecuentes suelen ser ficheros de configuración con nombre incorrecto o permisos incorrectos

16.3. configuración de ssh

Resolución del nombre de máquina

Como ya sabes

- Para acceder a cualquier recurso en internet, por ejemplo un servidor de ssh, normalmente es necesario o bien indicar el nombre completo (FQDN) de la máquina o bien escribir su dirección IP

Ejemplos:

```
ssh jperez@alpha.subdominio.dominio.com
ssh jperez@192.168.1.40
```

Resulta conveniente poder usar directamente el nombre de máquina (p.e. *alpha*), sin dominio completo ni dirección IP

Ejemplo:

```
ssh jperez@alpha
```

Hay dos formas de hacerlo

- Editar el fichero `/etc/hosts`
  - Ventaja: sirve para cualquier servicio
  - Inconveniente: solo *root* puede hacerlo
- Editar el fichero `~/.ssh/config`
  - Ventaja: cualquier usuario puede editar su fichero
  - Inconveniente: solo sirve para ssh (y servicios basados en ssh)

Fichero /etc/hosts

- Antes de que existiera el protocolo DNS (año 1983), se resolvían los nombres mediante el fichero `/etc/hosts`, gestionado a mano
- Sigue en uso, normalmente para resolver direcciones IP de especial interés para el administrador local
- Este fichero prevalece sobre el DNS: si una dirección se encuentra aquí, ya no se realiza una consulta DNS
- Podemos encontrar un fichero con idéntico forma y propósito en
  - `%SystemRoot%\system32\drivers\etc\hosts` (MS Windows)
  - `/private/etc/hosts` (macOS)

Ejemplo de fichero /etc/hosts

```
127.0.0.1 localhost

192.168.254.21 f-l-vm01.subdominio.dominio.com f-l-vm01
192.168.254.30 profes.subdominio.dominio.com profes

192.168.254.40 f-l3109-pc00.subdominio.dominio.com f-l3109-pc00
192.168.254.41 f-l3109-pc01.subdominio.dominio.com f-l3109-pc01

192.168.254.65 f-l3202-pc01.subdominio.dominio.com f-l3202-pc01
192.168.254.66 f-l3202-pc02.subdominio.dominio.com f-l3202-pc02
```

Configuración de ssh en el cliente

El usuario puede configurar el comportamiento de su cliente ssh en el fichero `~/.ssh/config`

Algunas de las opciones más interesante son

- Especificar cuál será el usuario por omisión para una maquina en particular, sin necesidad de especificarlo en cada ocasión
- Indicar el nombre de una máquina sin usar ni la dirección IP, ni el nombre completo (FQDN) ni modificar el fichero `/etc/hosts` (Que exije privilegios de root)
- Enviar periódicamente un mensaje al servidor para que mantenga abierta la conexión

Fichero `~/.ssh/config` de ejemplo

```
host alpha
user juanperez
hostname 192.168.19.27
# Permite hacer directamente 'ssh alpha', sin indicar mi usuario
# (Si el usuario coincide en máquina local y remota, tampoco hace
# falta especificarlo)

host alpha
user juanperez
hostname alpha.midominio.com
# Similar al caso anterior, pero con FQDN

host *
  ServerAliveInterval 60
# Mantiene la conexión abierta
```

Mas información en
`man 5 ssh_config`

Configuración adicional del servidor

- `/etc/ssh/ssh_config`
- `/etc/ssh/sshd_config`

16.4. sshfs

sshfs

Supongamos que, usando la red, quiero trabajar con unos datos que están en una máquina remota, su sitio no es la máquina en la que yo estoy sentado. Tal vez porque uso un ordenador móvil, pero los datos no son móviles

Disponemos de muchísimas soluciones, cada una con sus ventajas e inconvenientes

Podemos trabajar:

- Por ssh

  Pero estamos limitados a la shell. No podemos usar ninguna aplicación gráfica, resulta muy limitado en Windows, ...

- Con una sesión gráfica remota: vnc, escritorio remoto de Windows, X window en remoto, etc

  Pero necesitamos una conexión relativamente buena y cargamos mucho la máquina remota, toda la aplicación está en la máquina remota

- Podemos sincronizar al estilo Dropbox.

  Pero necesitamos una cuenta, vinculada a 1 persona, con limitaciones de tamaño, dependencia del proveedor, etc Además se pueden provocar discrepancias, y los ficheros solo se guardan en la máquina remota cuando abro una sesión en la máquina remota y sincronizo

- Podemos montar un sistema de ficheros por NFS (como en nuestros laboratorios Linux), por SAMBA o similar

  Pero hace falta mucha administración en la máquina remota, y normalmente, por motivos de seguridad, el cliente solo podrá estar en sitios muy concretos

- Podemos usar una VPN, cuya administración no es trivial

- Podemos usar un directorio compartido de VirtualBox, pero solo en el caso (muy) particular de un *guest* VirtualBox y un *host* que lo soporte

- Otra de las alternativas es sshfs

sshfs: Secure SHell FileSystem
Sistema de ficheros de red basado en FUSE (*Filesystem in userspace*)
Permite usar un sistema de fichero remoto como si fuera local

- Hay disponibles implementaciones libres y gratuitas, para Linux, macOS y Windows (SSFS-Win)

- Menos eficiente pero más seguro que NFS

- No hace falta ninguna administración en la máquina remota (servidor) , basta con que tengamos una cuenta de ssh ordinaria

- En el cliente basta instalar el paquete `sshfs` y ejecutar una única orden

Inconvenientes de sshfs

- Dependemos continuamente de la red (con sistemas tipo Dropbox solo hace falta red cuando sincronizamos)

- En el ordenador local necesitamos todas las aplicaciones (mientras que con sistemas tipo vnc, el cliente puede ser mucho más ligero)

- Más pesado y menos eficiente que NFS o samba

Punto de montaje

En Unix/Linux, el punto de montaje es un directorio del sistema de ficheros *normal*, local, donde queremos que sea visible un nuevo sistema de ficheros

En el caso de sshfs, el nuevo sistema de ficheros será el de la máquina remota, a la que accedemos por ssh

- El punto de montaje es un directorio ordinario, que tiene que existir antes de montar el sistema remoto
- Suele estar vacío, pero puede contener ficheros

  En el caso de sshfs, si no está vacío hay que añadir la opción `-o nonempty`

  - Al montar un sistema de ficheros en un punto de montaje, el contenido del punto de montaje queda inaccesible
  - Al desmontar, el contenido vuelve a ser visible

Montar un directorio con sshfs

- Montar el *home* remoto:

  ```
  sshfs  usuario@maquina: /punto/de/montaje
  ```

- Montar un directorio remoto cualquiera

  ```
  sshfs  usuario@maquina:/un/directorio  /punto/de/montaje
  ```

  (Siempre path absoluto, no soporta ~)

- Desmontar:

  ```
  fusermount -u /punto/de/montaje
  ```

- Para poder usar sshfs (en el cliente) sin tener privilegios de root, es necesario activar la opción `user_allow_other` en el fichero `/etc/fuse.conf` (y reiniciar el demonio[13] o la máquina)
- En conexiones lentas puede ser conveniente añadir la opción -C para que comprima el tráfico

  ```
  sshfs -C usuario@maquina:/path  /punto/de/montaje
  ```

[13] systemctl restart ssh

16.5. Túneles con SSH

Túneles con SSH

SSH permite hacer túneles, aka *port forwarding*

Concepto similar al de VPN, pero no es una verdadera VPN

- Se redirige un único puerto
- Solamente TCP, no UDP

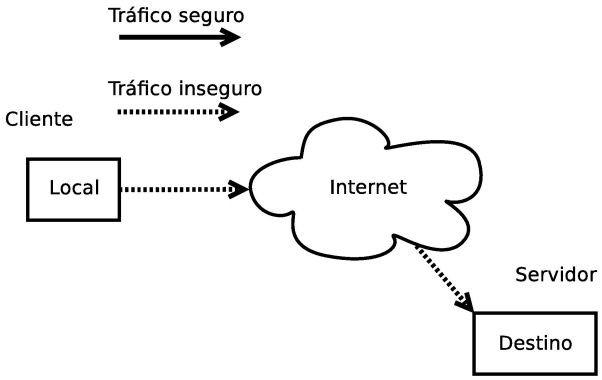
A través de un túnel, las conexiones a cierto puerto TCP de una máquina se redirigen a otro puerto TCP en otra máquina

Dos tipos:

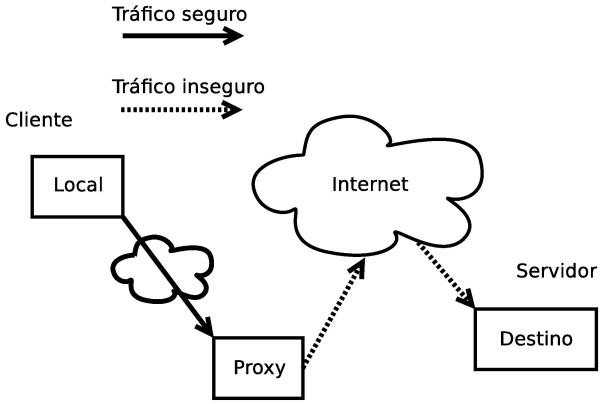
- Túnel local, aka túnel (*a secas*). (*local tunnel, tunnel*)
- Túnel remoto, aka túnel inverso. (*remote tunnel, reverse tunel*)

16.5.1. Túnel local

Túnel local



Escenario típico donde usamos un servicio sobre un canal no seguro



Si tenemos cuenta en una máquina acccesible mediante ssh, podemos usarla como proxy

- Establecemos un túnel ssh desde la máquina local al proxy

Ventajas del túnel local

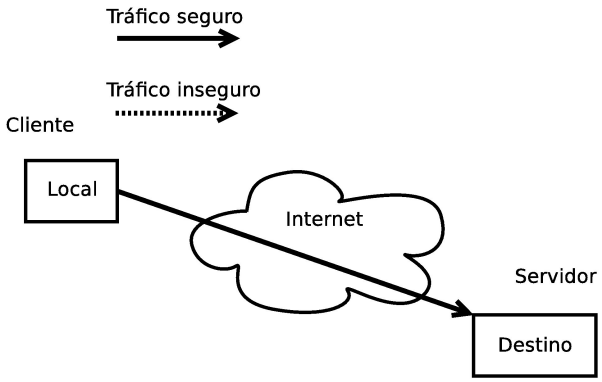
- Permite asegurar el primer tramo, que suele ser el más peligroso
- Para el servidor, las peticiones vienen desde el proxy, no desde la máquina local. Esto es de utilidad
  - Si se trata de un servicio vinculado a la IP del cliente,
  - Para evitar cortafuegos, censura, etc
  - Burlar restricciones en la red del cliente

  El tráfico viaja cifrado en la red de la máquina local, el administrador de esa red pierde todo control sobre él

  - El administrador de la red local puede solucionar este problema prohibiendo todo tráfico cifrado, y con ello los túneles

Inconvenientes

- Encaminamiento en triángulo
- El proxy es un cuello de botella



En caso de que tengamos una cuenta en el servidor, accesible mediante ssh, podemos asegurar todo el trayecto

- Establecimiento del túnel. En `maquina_local` ejecutamos

  ```
  ssh -L puerto_local:maquina_destino:puerto_remoto usuario@proxy
  ```

- Uso del túnel:

  Indicamos al cliente que se conecte a `puerto_local` en `maquina_local`

  (El servicio estará realmente en el `puerto_destino` de `maquina_remota`)

- `ssh -L`

  además de redirigir los puertos, abre una sesión de shell ordinaria en el proxy

- `ssh -fNL`

  Hace que el túnel se lance en segundo plano (tras preguntar contraseñas), pero no abre una sesión de shell en el proxy

  - Esto puede ser conveniente para el usuario experimentado, así no ocupa el terminal
  - Pero despista al usuario principiante, pues no resulta tan claro si el desvío de puerto está activo o no

Ejemplo:
Acceder al servidor web en `bilo.gsyc.es`, usando `epsilon01` como proxy

- Establecemos el túnel en la máquina local:

  ```
  ssh -L 8080:bilo.gsyc.es:80 milogin@epsilon01.aulas.gsyc.es
  ```

- Usamos el túnel:

  En la máquina local, introducimos en el navegador web la url

  `http://localhost:8080`

  El cliente cree conectarse a su máquina local, de hecho eso hace. Pero el túnel redirige ese tráfico al proxy, y del proxy al destino

Proxy SOCKS

Un túnel local ordinario no sirve para navegar normalmente por el web

- La técnica anterior nos permite usar un túnel ssh para acceder a 1 servidor web
- Pero en cuanto hagamos clic sobre un enlace fuera de la máquina remota, dejamos de usar el proxy
- Para una sesión de navegación ordinaria tendríamos que abrir 10, 15, 20 túneles...

Pero openssh puede hacer *port forwarding* dinámico, como servidor del protocolo SOCKS

Configuración de proxy SOCKS

1. El usuario establece un túnel desde la máquina local hasta el proxy, añadiendo la opción -D e indicando un puerto local, con lo que se instala en la máquina local un servidor SOCKS

   - `ssh -D puerto_local usuario@proxy`

   El puerto estándar es el 1080, puede usarse cualquier otro

2. El usuario indica a su navegador web que todas las peticiones debe hacerlas al servidor SOCKS que está en `maquina_local:puerto_local`
3. El servidor de la máquina local pedirá al proxy que haga las peticiones, y este se las hará al servidor web
4. El servidor web responderá al proxy, que reenvía la respuesta al servidor SOCKS, de donde la lee el navegador web

Error frecuente: el usuario indica a su navegador que el servidor SOCKS está en el proxy. Esto es incorrecto. El servidor SOCKS está en la máquina local

La configuración del navegador es, obviamente, dependiente del navegador. Cualquier navegador con un mínimo de calidad permitirá hacerlo

- Firefox:

  ```
  editar | preferencias | general | proxy de red |
  configuración manual del proxy | host SOCKS
  ```

- Google Chrome

  Es necesario lanzarlo desde la shell con el parámetro adecuado

  - Linux

    ```
    google-chrome --proxy-server="socks://localhost:1080"
    ```

  - macOS

    ```
    open -a Google\ Chrome     \
    --args --proxy-server="socks://localhost:1080"
    ```

- Las conexiones ssh pueden caerse.

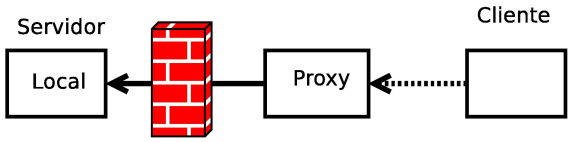
  En vez de `ssh`, podemos emplear `autossh`, que monitoriza la conexión y la reinicia si se cae

  Esto es útil combinado con el acceso automático sin contraseña

- La aplicación `tsocks` permite usar un proxy SOCKS de forma transparente a las aplicaciones (aplicaciones no preparadas para usar SOCKS)

16.5.2. Túnel remoto

Túnel remoto



Con un túnel remoto, aka túnel inverso, podemos traer a la máquina local las conexiones a cierto puerto del proxy

Esto puede tener al menos tres utilidades

1. Proteger el servidor
2. Distribuir servicios
3. Acceder a servidor tras NAT, sin configurar el NAT

Utilidad 1: Proteger el servidor

Un túnel inverso puede servir para proteger un servidor

- El proxy es necesariamente una máquina expuesta, puede necesitar gran visibilidad y muchos servicios (por ejemplo un servidor web)
- Pero el resto de los servicios, los colocamos en el servidor, en una máquina distinta, debidamente aislada

De esta forma, si un atacante comprometiera el proxy

- Tendríamos un problema, podría hacer p.e. un *website defacement*
- Pero el problema estaría contenido, no tendría acceso al resto de servicios, p.e. la base de datos

Naturalmente, cabe la posibilidad de que el atacante rompa la seguridad del túnel y acceda al servidor, pero esto añade una barrera adicional, relativamente robusta

Utilidad 2: Distribuir servicios

El proxy es una máquina distinta al servidor

- Puede ser útil para equilibrar la carga
- Puede ser útil si queremos combinar distinto software o distintos sistemas operativos

Utilidad 3:Acceder a servidor tras NAT, sin configurar el NAT

Tenemos un servidor tras un NAT

- La técnica habitual para permitir conexiones entrantes a este servidor es hacer *port forwarding* aka *abrir puertos* en el router que hace NAT
- Pero en vez esto, podemos conseguir un resultado similar, sin necesidad de modificar la configuración del router NAT

Usar un túnel ssh inverso (en vez del *port forwarding* tradicional), puede ser de utilidad:

- Si es un NAT que nosotros no podemos administrar (somos usuarios ordinarios, sin privilegios de administrador)
- Si *abrir los puertos* del NAT es incómodo
  - Hay un NAT tras otro NAT, tendría que configurar ambos
  - El NAT solo se puede configurar via web, típicamente a mano, resulta complicado automatizarlo

    (Mientras que el túnel inverso se prepara con 1 mandato desde la shell, fácil de incluir en un script, cron, etc)
  - Es necesario detener y reiniciar algún demonio (como el NAT de VirtualBox)

Inconvenientes de esta técnica:

- Dependemos de la existencia y disponibilidad del proxy
- El proxy necesita una IP pública
  - O bien el proxy está tras un NAT que sí podemos administrar. Pondríamos como dirección del proxy la del router que hace NAT, y redigiríamos a su vez esa conexión a una máquina de la red privada
- Solo es aplicable a TCP, no a UDP
- Estamos cifrando todo el tráfico (puede que no sea necesario)

- Establecimiento del túnel. En `maquina_local` ejecutamos

  ```
  ssh -R puerto_proxy:localhost:puerto_local  usuario@proxy
  ```

- Uso del túnel:

  Indicamos al cliente que se conecte a `puerto_proxy` en `proxy`

  El cliente cree conectarse al proxy, de hecho lo está haciendo. Pero el túnel redirige el tráfico al `puerto_local` de `localhost`, que es donde está el servicio

- Como en el túnel local, las opciones `-f` y `-N` son aplicables, con el mismo significado

Ejemplo: Tengo el servicio de escritorio remoto de `pilder01` en el puerto 5900 de la dirección privada 192.168.1.8

Quiero usar como proxy la máquina `miproxy.gsyc.es`, puerto 15900

- En `pilder01` ejecuto:

  ```
  ssh -R 15900:localhost:5900  milogin@miproxy.gsyc.es
  ```

- Para acceder a este servicio, el cliente ejecuta

  ```
  vinagre miproxy.gsyc.es:15900
  ```

  (el servicio está realmente en el puerto 5900 de pilder01)

Para que el cliente pueda estar en una máquina distinta a la máquina proxy, es necesario:

1. En el proxy, en en el fichero

   `/etc/ssh/sshd_config`

   añadir la entrada

   `GatewayPorts yes`

2. `sudo /etc/init.d/ssh restart`

Por omisión, esta opción no está activada (y solo `root` puede activarla)
Por tanto:

- Si tenemos una cuenta `ssh` ordinaria en el proxy, podemos hacer el túnel inverso, pero el cliente deberá estar en el mismo proxy

  - Además, el cliente deberá usar el nombre `localhost` y la dirección IP 127.0.0.1
    En el ejemplo anterior, el cliente solo podría estar en `proxy.gsyc.es` y debería ejecutar

```
vinagre localhost:15900
```

o bien

```
vinagre 127.0.0.1:15900
```

Pero no podría usar el nombre `proxy.gsyc.es`

- Si tenemos privilegios de adminitrador en el proxy y añadimos a `sshd_config` la opción `GatewayPorts yes`, el cliente podrá estar en cualquier lugar

Otro ejemplo de túnel inverso

- `mortuno@gsyc:~$ ssh -R 8080:localhost:80 mortuno@miproxy.gsyc.es`

- El cliente accede a

  `http://myproxy.gsyc.es:8080`

  donde verá

  `http://gsyc.es`

Resumen

Recapitulando, resumimos así el uso de los túneles directo e inverso

- En todos los casos:

  El ssh se hace desde la máquina local hasta el proxy (no hasta la máquina remota)

- Túnel directo:

  ```
  ssh -L puerto_local:maquina_destino:puerto_remoto usuario@proxy
  ```

  - El servicio está en `maquina_destino:puerto_remoto`
  - El cliente está en la máquina local, se conecta al puerto local

- Túnel inverso:

  ```
  ssh -R puerto_proxy:localhost:puerto_local usuario@proxy
  ```

  - El servicio está en `máquina local:puerto_local`
  - El cliente está en una máquina remota cualquiera, desconocida. Se conecta a `proxy:puerto_proxy`

scp

scp va sobre ssh, por tanto

- Usa el mismo servidor (sshd), escuchando en el mismo puerto (22)
- Si preparamos un túnel para entrar en el servidor ssh de una máquina, también podemos hacer scp a esa máquina
- La única diferencia es que, en el cliente, para indicar el puerto
  - ssh usa `-p` (minúscula)
  - scp usa `-P` (mayúscula)

Ejemplo

Tenemos la máquina virtual `pc01`, en el *host* `zeta01`, conectada a la red a través de NAT

- Establecemos el túnel (en este caso, remoto)

```
user@pc01:~$ ssh -R 2222:localhost:22 milogin@zeta01
```

- Desde el *host*, accedemos al servidor de ssh en `pc01`

```
milogin@zeta01:~$ ssh -p 2222 user@localhost  # p minúscula
```

- Desde el *host*, copiamos el fichero `holamundo.txt` al directorio `/tmp` de `pc01`

```
milogin@zeta01:~$ scp -P 2222 holamundo.txt user@localhost:/tmp
                                                    # P mayúscula
```

autossh

Una conexión ssh en desuso se cortará automáticamente. Podemos evitarlo en el lado del cliente añadiendo en `.ssh/config`

```
host *
    ServerAliveInterval 60
```

O mejor aún, usando `autossh`

```
apt update; apt upgrade; apt install -y autossh
```

No lanzaremos ssh directamente desde el terminal, sino desde autossh. Típicamente con opciones como

```
-M 0 -o "ServerAliveInterval 30" -o "ServerAliveCountMax 3"
```

Esto hace que cada 30 segundos se compruebe la conexión. Y se relance ssh si falla 3 veces seguidas.

```
#!/bin/bash
PUERTO_PROXY=9999
USUARIO=jperez
PROXY=miproxy
PUERTO_LOCAL=22
autossh -M 0 -o "ServerAliveInterval 30" -o "ServerAliveCountMax 3" \
   -R ${PUERTO_PROXY}:localhost:${PUERTO_LOCAL} ${USUARIO}@${PROXY}
```

Más información en este enlace:
ssh tunnelling for fun and profit autossh

17. Control de Integridad

17.1. Función hash

Función hash

Técnica básica para garantizar integridad de un mensaje

```
hash:
a mess, jumble, or muddled.
to chop into small pieces; make into hash; mince.
to muddle or mess up.
```

- Función que, a partir de un bloque arbitrario de datos (*message*), genera de forma determinista un *valor hash*, aka *message digest*, aka *digest*.

  Este valor hash identifica de forma prácticamente unívoca al mensaje, de forma que un cambio en el mensaje, aunque sea pequeño, provoque un cambio en el valor hash

- Es posible que dos mensajes distintos generen el mismo valor hash, aunque muy difícil

Función hash ideal:

- Fácil de generar
- Muy difícil generar el mensaje a partir del hash
- Muy difícil modificar el mensaje manteniendo el hash
- Muy difícil encontrar dos mensajes con el mismo hash

Ejemplos de funciones hash: MD2, MD4, MD5, SHA-1, SHA-2

- En el año 1996 aparecieron vulnerabilidades en MD5. En 2005, en SHA-1
- SHA-2 se considera seguro actualmente
- Se espera que SHA-3 sea definido a finales de 2012

```
koji@mazinger:~$ md5sum ubuntu-11.10.iso
c396dd0f97bd122691bdb92d7e68fde5  ubuntu-11.10.iso

koji@mazinger:~$ sha1sum ubuntu-11.10.iso
8492d3daf0c89907c4301cb2c72094fe59037c76  ubuntu-11.10.iso

koji@mazinger:~$ sha224sum ubuntu-11.10.iso
b070d093af7e933cedf6c21cf5a5b71ff7eb29946b9fc2a21d609ca0  ubuntu-11.10.iso
```

17.2. cmp

cmp

- Con el *hash* podemos saber si un fichero ha cambiado. Pero no qué ha cambiado
- La orden *cmp* indica el primer byte que ha cambiado en dos ficheros

```
koji@mazinger:~$ cmp a.pdf b.pdf
a.pdf b.pdf son distintos: byte 63401, línea 716
```

17.3. diff

diff

En caso de que los ficheros estén en formato de texto plano, *diff* indica cuáles son los cambios, línea a línea

Supongamos los ficheros `a.txt` y `b.txt`

```
a.txt:        b:txt
------------------------
rojo          colorado
amarillo      ámbar
verde         verde
```

- Las líneas que estén en `a.txt`, pero no en `b.txt` se muestran precedidas del signo de menor
- Las líneas que estén en `b.txt` pero no en `a.txt`, se muestran precedidas del signo de mayor
- Las que no cambian, no se muestran

```
koji@mazinger:~$ diff a.txt b.txt
1,2c1,2
< rojo
< amarillo
---
> colorado
> ámbar
```

- `1,2c1,2` representa el número de línea donde sucedió el cambio, podemos ignorarlo
- En entornos de programación, esto sirve para ver cambios en programas y distribuir *parches* (la orden *patch* puede construir `b.txt` a partir de `a.txt` y la salida de `diff`)

En administración de sistemas, nos sirve para saber qué ha cambiado entre dos instantes de tiempo
P.e.

- En el instante t_0 ejecutamos `ls /directorio > t0.txt`
- En el instante t_1 ejecutamos `ls /directorio > t1.txt`

Comparamos con `diff t0.txt t1.txt`

- Los ficheros desaparecidos se muestran precedidos del signo de menor
- Los ficheros nuevos se muestran precedidos del signo de mayor

Podemos aplicar esto mismo a `ps`, `netstat`, `who`, ...
Es útil para

- Automatizar mediante scripts
- Observar *a ojo* un número elevado de elementos

18. Benchmark

Benchmark

Resultado de la ejecución de una aplicación que busca estimar el rendimiento de un sistema (informático)

- En español podríamos decir *comparativa*, aunque no resulta tan preciso
- Se puede hacer *benchmark* de cpu, disco, tarjeta gráfica, red, etc

18.1. Benchmark de la cpu

Benchmark de la cpu

- MIPS: Millones de instrucciones por segundo. Forma de medir la potencia de un procesador. Útil para comparar distintos procesadores, siempre que usen mismo juego de instrucciones, con un benchmark en mismo compilador con mismas optimizaciones.
- BogoMIPS: Medida propia de Linux. Estimación aproximada de los MIPS. *Bogus*: Incorrecto, engañoso

  ```
  dmesg |grep -i bogo
  ```

- SPECint: Alternativa más exacta

18.2. Benchmark de red

Benchmark de red

Miden la tasa de transferencia máxima entre un cliente y un servidor (network throughput). También se puede traducir por *ancho de banda digital*, rendimiento de red o caudal de datos. [14]

Permiten usar diferentes protocolos y tamaños de paquete

- iperf. Sencillo, muy popular. Disponible para Windows, Linux, macOs, Android, iOS, FreeBSD
- netperf. Sencillo, muy popular. Centrado en Unix/Linux
- netio. Sencillo, disponible en Windows, Linux y varios Unix
- ... entre otros

[14] Es común denominarlo ancho de banda, sin más. Pero este en rigor es el ancho de banda analógico, un concepto distinto: un rango de frecuencias

inxi

inxi es una herramienta muy útil que recopila diversa información sobre nuestro sistema.

- Instalación:

```
apt update; apt upgrade; apt install -y inxi
```

- Basta ejecutar desde un terminal `inxi -b` para obtener un informe muy completo

```
System:    Host: minisal Kernel: 5.11.0-38-generic x86_64 bits: 64 Console: N/A
           Distro: Ubuntu 20.04.3 LTS (Focal Fossa)
Machine:   Type: Laptop System: Apple product: Macmini7,1 v: 1.0 serial: <superuser/root required>
           Mobo: Apple model: Mac-35C5E08120C7EEAF v: Macmini7,1 serial: <superuser/root required> UEFI: Apple
           v: 431.0.0.0.0 date: 02/22/2021
CPU:       Dual Core: Intel Core i5-4278U type: MT MCP speed: 800 MHz min/max: 800/3100 MHz
Graphics:  Device-1: Intel Haswell-ULT Integrated Graphics driver: i915 v: kernel
           Display: server: X.org 1.20.11 driver: i915 tty: 114x60
           Message: Advanced graphics data unavailable in console. Try -G --display
Network:   Device-1: Broadcom and subsidiaries BCM4360 802.11ac Wireless Network Adapter driver: wl
           Device-2: Broadcom and subsidiaries NetXtreme BCM57766 Gigabit Ethernet PCIe driver: tg3
Drives:    Local Storage: total: 931.51 GiB used: 117.52 GiB (12.6%)
Info:      Processes: 267 Uptime: 3h 56m Memory: 7.65 GiB used: 1.54 GiB (20.2%) Init: systemd runlevel: 5
           Shell: bash inxi: 3.0.38
```

Bancos de memoria

Para saber qué tipo de memoria tiene nuestro equipo, así como cuantos *slots* de memoria libres y ocupados, podemos ejecutar cualquiera de estos comandos

- `sudo inxi -m  # Información básica`

```
Memory:
  RAM: total: 7.64 GiB used: 3.64 GiB (47.7%)
  Array-1: capacity: 8 GiB slots: 2 EC: None
  Device-1: DIMM0 size: 4 GiB speed: 1600 MT/s
  Device-2: DIMM0 size: 4 GiB speed: 1600 MT/s
```

- `sudo dmidecode -t memory  # Información detallada`

```
[...]
Handle 0x0007, DMI type 17, 34 bytes
Memory Device
    [...]
Size: 4 GB
Form Factor: SODIMM
Locator: DIMM0
Bank Locator: BANK 0
Type: DDR3
Speed: 1600 MT/s
    [...]
```

Instalación de iperf

Paquetizado en Ubuntu, lo instalamos de la forma habitual

```
apt update
apt upgrade
apt install -y iperf
```

Uso de iperf

TCP

- Servidor:

  `iperf -s`

- Cliente:

  `iperf -c <SERVIDOR>`

UDP

- Servidor:

  `iperf -u -s`

- Cliente:

  `iperf -u -c <SERVIDOR>`

Por omisión emplea el puerto TCP/UDP 5000. Se puede cambiar con la opción `-p`

19. Sesiones gráficas remotas

19.1. Introducción

Sesiones gráficas remotas

Definiciones

- Máquina local

  Equipo en el que trabaja el usuario, donde tiene su pantalla, teclado, y posiblemente, ratón

- Máquina remota

  Máquina donde se ejecuta la aplicación a usar. El usuario no tiene acceso a su teclado, pantalla ni ratón

Sesiones de texto / Sesiones remotas

- El protocolo ssh nos permite trabajar cómodamente en máquinas Unix remotas, con sesiones de texto
- También se puede usar ssh en Microsoft Windows, aunque con muchas limitaciones, resulta poco natural
- Pero habrá ocasiones en que será conveniente o imprescindible usar sesiones gráficas en máquinas remotas

19.2. Protocolos para sesiones gráficas remotas (1)

Protocolos para sesiones gráficas remotas (1)

- Soluciones propietarias como LogMeIn o TeamViewer
- RDP. *Remote Desktop Services*, también conocido como *Terminal Services*.

  Nativo en Microsoft Windows, usable desde otras plataformas, p.e. `gnome-rdp`, `vinagre` (clientes) o `xrdp` (servidor)

  Puerto por omisión: 3389 TCP

19.3. Protocolos para sesiones gráficas remotas (2)

Protocolos para sesiones gráficas remotas (2)

- X11 forwarding. Forma parte de X Window.

  Tradicional y nativo en Linux/UNIX, usable en Microsoft Windows.

  Normalmente no trabaja sobre un escritorio completo sino con ventanas individuales.

  Anticuado auque sigue disponible. Intrínsecamente poco seguro. Puerto por omisión: 6000 TCP

- VNC

  Protocolo abierto, multiplataforma. Nativo en muchos Linux. Disponible en Microsoft Windows, Unix, *BSD, macOS, Android, iOS, ...

19.4. X11 Forwarding

X11 Forwarding

- X Window (año 1984) es el protocolo tradicional en Unix para mostrar gráficos. En local y también en remoto

  Nada que ver con Microsoft Windows

- En 1987 aparece la versión 11 de X Window, sigue siendo la versión actual. De ahí el nombre X11

- X Window es un protocolo cliente-servidor. Aunque la terminología puede ser anti-intuitiva:

  - La máquina local es el servidor. En ella están los gráficos. Normalmente lo llamaríamos *cliente*, pero es el *servidor X11* (ofrece el servicio de representación gráfica)
  - En la máquina remota está el *cliente X Window*. Aunque en esta máquina están los procesos que usan los gráficos, esto es, los servicios. (Los servicios son los clientes de los gráficos)

- En la máquina local puede ser necesario configurar los permisos del servidor

  ```
  xhost +
  ```

  permite que cualquier cliente X Window desde cualquier máquina lance ventanas en nuestro servidor X Window local.

    - También se pueden dar permisos más específicos

- Entre dos máquinas Ubuntu con el mismo usuario en ambas máquinas, mediante ssh, no es necesario dar permisos adicionales

Para lanzar una aplicación gráfica en una máquina remota:

- Desde la máquina local hacemos ssh a la máquina remota con la opción `-X` (mayúscula)

  ```
  jperez@gamma12:~$ ssh -X alpha
  ```

- Lanzamos la aplicación en segundo plano

  p.e

  ```
  jperez@alpha:~$ xeyes&
  ```

19.5. VNC

VNC

VNC, *Virtual Network Compute* es un protocolo para abrir sesiones gráficas en máquinas remotas

- Arquitectura cliente-servidor, desarrollado por The Olivetti & Oracle Research Lab
- La terminología es la habitual, no la de X Window. El cliente está en la máquina local, el servidor, en la máquina remota
- Implementación liberada como software libre en 2002
- Muy popular. Muchas implementaciones para cualquier plataforma (Microsoft Windows, Linux, Unix, macOS, Android, iOS, Raspberry Pi ...)

  Cualquier servidor de cualquier plataforma puede trabajar con cualquier cliente en cualquier otra

En Ubuntu

- Tenemos un servidor integrado por omisión, llamado `vino`
- Hay un cliente llamado `vinagre`
- También podemos usar *TightVNC*, entre otras implementaciones

En Windows

- Podemos usar *TightVNC*, entre otras implementaciones

En macOS

- Como cliente podemos usar el navegador nativo de macOS, Safari. Basta escribir en la barra de direcciones la dirección del servidor:

  `vnc://maquina:puerto`

- Otra opción (con cliente y servidor) es RealVNC. Tiene una versión comercial y otra libre (RealVNC Open Edition)

En Raspbian (Raspberry Pi)

- El servidor VNC instalado es Real VNC, cuya configuración por omisión es incompatible con VNC estándar. Hay dos soluciones

  1. Usar el cliente de Real VNC (vncviewer), disponible en el web de Real VNC
  2. Configurar el servidor de Real VNC para que use autenticación VNC, no autenticación UNIX

vinagre

El cliente de VNC oficial de Ubuntu es `vinagre`

- Está desarrollado conjuntamente con `vino`, algunas opciones avanzadas pueden funcionar mejor con este servidor
- También tiene soporte para RDP

Uso:
`vinagre <MAQUINA>:<PUERTO>`

vino

- En Ubuntu, el servidor `vino` está instalado por omisión en las máquinas con escritorio gráfico
- Por omisión trabaja en el puerto 5900 TCP
- Para cambiar el puerto del servidor:

  1. Instalamos dconf-editor

     `sudo apt install dconf-editor`

2. Lanzamos dconf-editor
3. En
   `org | gnome | desktop | remote-access`
   Cambiamos `alternative-port`
   Activamos `use-alternative-port`

Normalmente usaremos vino cuando ya tenemos una sesión gráfica abierta en un Ubuntu con escritorio tradicional

Pero no es adecuado para abrir:

- Una segunda sesión gráfica en la misma máquina
- Una sesión gráfica en una máquina remota a la que no tenemos acceso físico o en la que no queremos abrir una sesión gráfica tradicional
- Una sesión en una máquina donde no queramos un escritorio tan pesado como Unity o Gnome

En cualquiera de estos casos

- En vez de usar *vino*, podremos usar *TightVNC*

Servidor de TightVNC en Ubuntu

El servidor de TightVNC es `vncserver`. Para usarlo necesitaremos, además del propio `vncserver`, un escritorio. Podríamos emplear Gnome o Unity, pero son muy pesados. Generalmente será más adecuado

- O bien un escritorio ligero como Xfce4
- O bien un gestor de ventanas como Openbox
  (podemos considerar a Openbox como un escritorio ultra-ligero)

Los pasos son:

1. Instalación de vncserver
2. Instalación del escritorio
3. Preparación del escritorio
4. Lanzamiento del servidor
5. Lanzamiento del cliente

1 Instalación de vncserver

En caso de que TightVNC no esté instalado en nuestro sistema

1. Como en cualquier instalación de un paquete nuevo, suele ser recomendable actualizar el sistema

   `sudo apt update; sudo apt upgrade`

2. Instalamos el paquete

   `sudo apt install tightvncserver`

2 Instalación del escritorio

Si vamos a usar Openbox

- Para saber si está instalado, intentamos ejecutar

  `openbox-session`

- Para instalarlo

  `sudo apt install openbox`

Si vamos a usar Xfce4

- Para saber si está instalado, intentamos ejecutar

  `xfce4-session`

- Para instalarlo

  `sudo apt install xfce4`

3 Preparación del escritorio

1. En el servidor escribiremos un fichero `~/.vnc/xstartup`

   con el siguiente contenido

   - Si vamos a usar Xfce4

     ```
     #!/bin/bash
     xrdb ~/.Xresources
     startxfce4&
     xterm&
     ```

   - Si vamos a usar Openbox

```
#!/bin/bash
xrdb ~/.Xresources
/usr/bin/openbox-session
```

2. Como a cualquier script, le damos permiso de ejecución, p.e.

   `chmod 755 ~/.vnc/xstartup`

- En Openbox lanzamos aplicaciones desde el menú contextual del botón secundario (derecho) del ratón
- Para que funcione puede ser necesario añadir el siguiente fichero

  `~/.config/openbox/menu.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<openbox_menu xmlns="http://openbox.org/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="/usr/share/openbox/menu.xsd">
    <menu id="root-menu" label="Openbox 3">
        <item label="Run Program">
            <action name="Execute">
                <execute>
                    gmrun
                </execute>
            </action>
        </item>
        <separator/>
        <item label="Terminal">
            <action name="Execute">
                <execute>
                    xterm
                </execute>
            </action>
        </item>
  </menu>
</openbox_menu>
```

4 Lanzamiento del servidor

Para lanzar el servidor ejecutamos la orden `vncserver`, indicando el tamaño de la pantalla y la profundidad del color, especificada en número de bits

Ejemplo:

`vncserver  -geometry 1024x768 -depth 16`

Si vamos a usarlo con frecuencia, puede ser conveniente poner esta orden en un script de shell, p.e.

`~/bin/mi_vncserver`

- La primera vez que lancemos `vncserver`, nos preguntará la contraseña de la sesión.

  Quedará almacenada en el fichero

  `~/.vnc/passwd`

- Si necesitamos cambiar la contraseña, ejecutamos

  `vncpasswd`

- También podemos cambiar la contraseña desde un script.

  P.e. la contraseña *sesamo* sería:

  `echo "sesamo\nsesamo\n\n" | vncpasswd`

Como cualquier demonio, `vncserver` deberá atarse a un puerto para aceptar peticiones, en su caso a un puerto TCP

- El puerto por omisión es el 5900 TCP
- Atención, el servidor de VNC no emplea el concepto *puerto TCP*, sino *display port*, donde

  puerto TCP = 5900 + *display port*

- Sin embargo, en el cliente normalmente sí indicaremos el puerto TCP, no el *display port*

Si no indicamos otra cosa, el demonio `vncserver` intentará usar el *display port* 0, puerto TCP 5900.

- Si está libre, mostrará el mensaje

  ```
  New 'X' desktop is gamma:0
  ```

- Si está ocupado (tal vez por vino), lo intentará en el *display port* 1, puerto TCP 5901. Si tiene éxito el mensaje será

  ```
  New 'X' desktop is gamma:1
  ```

  y así sucesivamente con los *display port* 2, 3, etc (puertos TCP 5902, TCP 5903, etc)

- También podemos indicar explícitamente el *display port* que usará el servidor:

  Ejemplo: *display port* 10 (puerto TCP 5910)

  `vncserver :10 -geometry 1024x768 -depth 16`

- La sesión permanecerá abierta hasta que la matemos explícitamente con

  `vncserver -kill :10`

  - Esta orden mata el proceso `vncserver`, la sesión X11 en `/tmp/.X11-unix` y los logs en `~/.vnc/`

5 Lanzamiento del cliente

En la máquina local ejecutamos
`vinagre <SERVIDOR>:<PUERTO`
p.e.
`vinagre gamma:5901`
Observa que en este caso indicamos el puerto TCP, no el *display port*

- Recuerda que para cerrar la sesión hay que matar el servidor, no basta con cerrar el cliente

Uso de VNC en Docker

VNC es una forma conveniente de abrir sesiones gráficas dentro de un contenedor.

Configuración de ejemplo para el *display port* 0, contraseña *sesamo*

`entrypoint.sh`:

```
#!/bin/bash
vncserver :0 -geometry 1024x768 -depth 16
/usr/bin/xterm&
/bin/bash
```

`Dockerfile`:

```
FROM ubuntu:16.04
ENV USER root

RUN apt-get update && DEBIAN_FRONTEND=noninteractive &&  \
    apt-get install -y \
    tightvncserver openbox \
    xterm
```

```
# Expose VNC port
EXPOSE 5900

#set password for vnc
RUN echo "sesamo\nsesamo\n\n" | vncpasswd

COPY entrypoint.sh /
ENTRYPOINT ["/entrypoint.sh"]
```

http://ortuno.es/Dockerfile_vnc.txt

Lanzamiento del contendor:

```
#!/bin/bash
DISPLAY_NUMBER=0
PORT=$((DISPLAY_NUMBER+5900))
IMAGEN=vnc
NOMBRE=jper${IMAGEN}01
USUARIO=jperez

docker run -it --rm -h ${NOMBRE}  --name ${NOMBRE} -p ${PORT}:${PORT} \
    -e DISPLAY=:${DISPLAY_NUMBER} ${USUARIO}/${IMAGEN}
```

http://ortuno.es/lanza_vnc.txt

20. Netstat

netstat

- `netstat` es una herramienta básica en cualquier SSOO (Unix, Linux, macOS, Windows...)
- Muestra información sobre conexiones de red, tablas de encaminamiento, estadísticas de los interfaces, NAT y multicast

Nos permitirá comprobar qué demonios están funcionando en nuestra máquina

- Es una herramienta básica para depurar cualquier servicio
- Un principio básico de seguridad en cualquier sistema es tener activos solo los servicios necesarios, cualquier nuevo servicio siempre implica un cierto riesgo

Información sobre conexiones de red, Linux

- `-tu`

  Muestra información sobre TCP y UDP (y no sobre los *Unix domain sockets*)

- `-p`

  Indica el programa a quien pertenece el socket

  Si lo ejecuta un usuario ordinario, solo muestra algunos nombres

  Si lo ejecuta root, muestra todos los nombres

- `-a`

  Muestra todos los sockets, no solamente las conexiones establecidas sino también los sockets que están *escuchando*

- `-n`

  Muestra direcciones IP y no nombres de máquina.

  Muestra números de puerto, no nombre de servicio asociado (en los *well know ports*). No intenta resolver nombres de máquina.

```
koji@afrodita:~$ sudo netstat -tupan
Conexiones activas de Internet (servidores y establecidos)
Prot Recv-Q Send-Q Dirección_Local Dirección_Externa    Estado      PID/Program name
tcp    0      0 0.0.0.0:22          0.0.0.0:*            ESCUCHAR    27488/sshd
tcp    0      0 127.0.0.1:631       0.0.0.0:*            ESCUCHAR    1320/cupsd
tcp    0      0 193.147.71.120:22   193.147.71.62:56881  ESTABLECIDO 26653/sshd: koji
tcp6   0      0 :::79               :::*                 ESCUCHAR    19514/xinetd
tcp6   0      0 :::22               :::*                 ESCUCHAR    27488/sshd
tcp6   0      0 ::1:631             :::*                 ESCUCHAR    1320/cupsd
udp    0      0 0.0.0.0:49573       0.0.0.0:*                        32398/avahi-daemon:
udp    0      0 0.0.0.0:5353        0.0.0.0:*                        32398/avahi-daemon:
```

```
koji@afrodita:~$ sudo netstat -tupa
Conexiones activas de Internet (servidores y establecidos)
Prot Recv-Q Send-Q Dirección_Local Dirección_Externa Estado      PID/Program name
tcp    0      0 *:ssh                  *:*           ESCUCHAR    27488/sshd
tcp    0      0 localhost:ipp          *:*           ESCUCHAR    1320/cupsd
tcp    0      0 afrodita:ssh           doublas:56881 ESTABLECIDO 26653/sshd: koji
tcp6   0      0 [::]:finger            [::]:*        ESCUCHAR    19514/xinetd
tcp6   0      0 [::]:ssh               [::]:*        ESCUCHAR    27488/sshd
tcp6   0      0 ip6-localhost:ipp      [::]:*        ESCUCHAR    1320/cupsd
udp    0      0 *:49573                *:*                       32398/avahi-daemon:
udp    0      0 *:mdns                 *:*                       32398/avahi-daemon:
```

Problema:

Netstat no comprueba que el demonio que está atado a un puerto sea el demonio *habitual* en ese puerto.

Ejemplo: si atamos un servicio cualquiera al puerto 80, netstat lo tomará por un servidor web

- Un demonio puede escuchar en un puerto (p.e. el 22) de cualquier dirección de la máquina, por tanto, en cualquiera de los interfaces de la máquina

  ```
  tcp    0      0 0.0.0.0:22          0.0.0.0:*            ESCUCHAR    27488/sshd
  ```

- O bien puede escuchar en un puerto (p.e. el 631), pero no de todas las direcciones/todos los interfaces, sino de una en concreto

  ```
  tcp    0      0 127.0.0.1:631       0.0.0.0:*            ESCUCHAR    1320/cupsd
  ```

  Este demonio está en la máquina afrodita, pero no atiende peticiones hechas a la dirección pública de afrodita, solamente a *localhost*/127.0.0.1

- Podemos usar `grep` para filtrar la salida de netstat

  ```
  koji@afrodita:~$ netstat -tupan |grep 22
  tcp        0      0 0.0.0.0:22            0.0.0.0:*               ESCUCHAR    -
  tcp        0      0 193.147.71.120:22     193.147.71.62:34285     ESTABLECIDO -
  tcp6       0      0 :::22                 :::*                    ESCUCHAR    -
  ```

Netsat en macOS

Uso típico:

`netstat -f inet -an`

- Para ver solo las conexiones tcp y udp, la opción no es -tu sino `-f inet`
- `-a` y `-n` se comportan como en Linux
- Para saber qué proceso escucha en cada puerto, no hay un equivalente a `-p`

  Pero podemos usar

  `sudo lsof -iTCP:12345 -sTCP:LISTEN`

  (Donde 12345 sería el puerto a consultar)

  o bien

  `sudo lsof -i -n -P |grep UDP`

21. Editores de texto

21.1. Introducción

Introducción

- Los editores de texto crean y modifican ficheros de texto *plano*

  Se emplea en programación y en configuración de sistemas

- Los procesadores de texto crean y modifican ficheros de texto con formato de fuente (negritas, cursivas, tipos de letra,etc), de página (interlineado, márgenes, etc) e imágenes

En cualquier Linux hay disponibles muchos editores
¿Cuál es mejor?

- Depende en buena parte de gustos personales
- Depende de dónde vayamos a usarlos
- Este es un asunto típico para *guerras de religión*

Bilo y Nano. Javier Malonda. (CC BY-NC-ND 3.0)

Tipos de editor de texto

1. Editores en modo gráfico
   - Su curva de aprendizaje suele ser más suave
   - Adecuados para trabajar como programador en un ordenador *estándar*, local y con gráficos
2. Editores en modo texto (editores de consola)
   - Curva de aprendizaje más dura (excepto algunos muy sencillos/simplones)
   - Permiten trabajar en remoto con la misma facilidad que en local
     - Podemos administrar sin problemas nuestra máquina Linux p.e. desde un Windows prestado y con mala conexión. O incluso una PDA y un teléfono móvil
   - Son los únicos disponibles en sistemas empotrados, como routers
   - Suelen ser los únicos disponibles en ordenadores a medio instalar, averiados, herramientas de rescate, etc

21.2. vi

vi

El editor *estándar* en Unix. Desarrollado por Bill Joy (Co-fundador de Sun Microsystems) en el año 1976.

Hoy usamos clones, especialmente vim o más recientemente, neovim

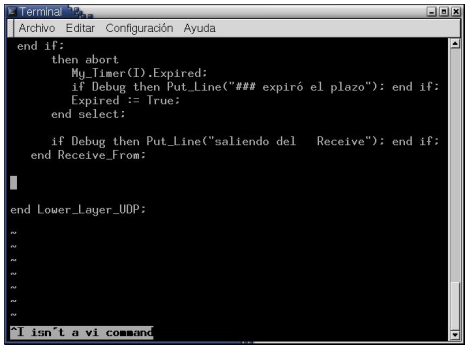
- Si no nos gusta vi/vim/neovim, casi siempre podremos instalar otro
- Pero para poder instalar otro, suele ser imprescindible manejar al menos las órdenes elementales de vi

Ventajas

- Normalmente estará disponible y funcionando en cualquier máquina Unix
- Hay versiones para la mayoría de los SSOO (Windows, macOS...)
- Es muy flexible y potente, conociéndolo bien se puede trabajar a gran velocidad
- Pensado para sesiones remotas con malas conexiones en un *terminal tonto* de los años 70, el ADM-3A

- Si trabajamos en una máquina con gráficos, puede ser conveniente usar un vim en módo gráfico, mejor integrado con el escritorio. Permitirá usar el ratón, funcionará el portapapeles del escritorio y podrá tener menús, de utilidad para ordenes que aún no hemos memorizado

  - En Windows, gvim
  - En Linux, gvim [15]
  - En OS X, MacVim (mvim)



Inconvenientes

- Interfaz de usuario muy anticuado, el usuario debe memorizar órdenes ¡donde hasta las mayúsculas son significativas!

21.2.1. Modos de vi

Modos de vi

1. Modo orden (también llamado modo comando, modo normal)

   En este modo guardamos el fichero, leemos otro, salimos, copiamos, pegamos, etc

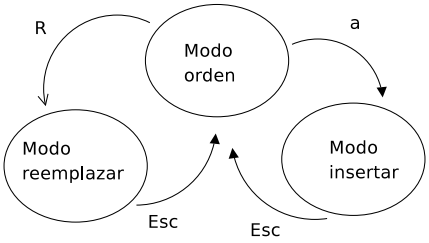
2. Modo insertar (también llamado modo texto o modo entrada)

   En este modo insertamos texto

3. Modo reemplazar (también llamado modo texto o modo entrada, sin distinguirlo del modo insertar)

   En este modo reemplazamos texto

[15] el nombre del paquete es vim-gtk



21.2.2. Órdenes imprescindibles

Órdenes imprescindibles

Desde la shell

```
koji@mazinger:~$ vi nombre_fichero.txt
```

(Edita el fichero del nombre indicado. Si no existe, lo crea)

Desde vi

```
a    Pasar de modo orden a modo insertar
R    Pasar de modo orden a modo reemplazar
Esc  Volver a modo orden

x    Borrar un carácter
J    Unir la línea actual con la línea siguiente
:wq  Escribir el fichero y salir
:q!  Salir sin guardar el fichero
```

Este conjunto de órdenes es suficiente para editar cualquier fichero

21.2.3. Órdenes básicas

Órdenes básicas

```
:r nombre       leer un fichero
:w nombre       escribir fichero
u               Deshacer último cambio
ctrl r          Rehacer lo último deshecho
D               Borrar hasta final de línea
dd              Borrar línea actual
yy              copiar (yanc) linea
p               pegar lo ultimo copiado o borrado
.               Repetir la última orden
/patron         Busca un patrón (hacia adelante)
n               Repetir búsqueda
N               Buscar en dirección inversa a anterior
```

```
G               Ir a Final del archivo
5G              Ir a línea 5
%               Salta al paréntesis que se corresponda con el
                paréntesis actual (o llave, corchete...)
```

Casi todas las órdenes permiten anteponer un número, que indica cuántas veces se repetirá

```
dd              Borrar línea actual
10dd            Borrar 10 líneas

u               Deshacer un cambio
3u              Deshacer últimos 3 cambios

cw              Cambiar una palabra
5cw             Cambiar 5 palabras
```

21.2.4. Otras órdenes

Otras órdenes

```
0               ir a principio línea
$               ir a fin linea
w               ir a siguiente palabra
b               ir a palabra anterior
r               Sustituir 1 carácter
cw              Cambiar palabra (change word)
dw              Borrar hasta fin palabra (delete word)
yw              Copiar  palabra
*               Buscar palabra igual a la palabra sobre la que
                está el cursor
ma              Poner marca de texto a
mb              Poner marca de texto b
'a              ir a marca a
'b              ir a marca b
Ctrl G          Indicar linea actual
~               Pasar de may. a minusc. o al revés


:49,53 w! fichero      Escribir en fichero lineas de 49 a 53
:.,53 w! fichero       Escribir en fichero desde linea actual
                       hasta línea 53
:1,$ s/digo/diego/g    Buscar todas las cadenas "digo" desde la línea
                       1 hasta el final, y reemplazarlas por "diego"
:set nu                Indicar el nº de linea
:set nonu              Desactivar nº de linea
:set ic                Ignore case (Insensible a mayus/min)
:set noic              Desactiva ic
```

Podemos configurar vim de forma persistente creando un fichero de configuración

- En Unix/Linux

  `~/.vimrc`

- En Windows 10 / Windows 11, vim 64 bits

  `c:\Archivos de programa\vim\_vimrc`

- En Windows 10 / Windows 11, vim 32 bits

  `c:\Archivos de programa (x86)\vim\_vimrc`

Por ejemplo, el fichero de configuración puede contener:

```
set vb
set ic
set tabstop=4
syntax on
```

Esto activa la *visual bell* (que elimina los molestos pitidos del terminal), ignora mayúsculas/minúsculas, fija el tabulador en 4 espacios y colorea el texto si reconoce la sintaxis

En Windows podemos añadir

```
set enc=utf-8
set guifont=Consolas:h12:cANSI:qDRAFT
```

De esta forma, empleará por omisión la misma codificación que en Unix/Linux. Y el tipo de letra tendrá un tamaño razonable

Para más información sobre vi, consulta la página web *vi lovers home page*

21.3. Editores ligeros

Editores ligeros

Hemos visto que vi tiene muchas ventajas. Pero si nos *asusta* su interfaz de usuario y necesitamos un editor en modo texto, disponemos de editores ligeros como

- mcedit (editor del mc, midnight commander)
- nano (clon de pico)
- joe

21.4. Emacs / XEmacs

Emacs / XEmacs

Editor clásico en Unix. Uno de los más conocidos, se populariza a mediados de los 80

Emacs trabaja en modo texto, XEmacs en modo gráfico

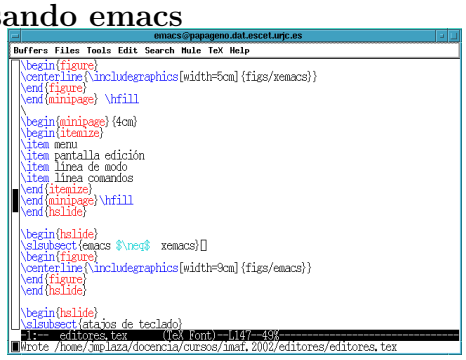
Ventajas

- Completísimo, es mucho más que un editor. Permite leer correo, news, se integra con gran cantidad de herramientas...
- Módulos para muchos lenguajes de programación
- Da formato y color al fuente, con mucha calidad.
- Completamente personalizable (en lisp)
- Puede emular a vi

Inconvenientes

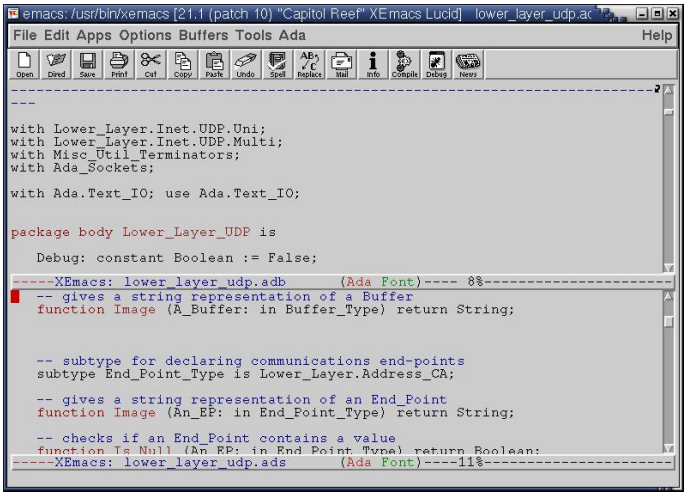
- Muy grande y pesado, consume muchos recursos.
- Su uso resulta complicado
- Aún para las tareas sencillas, tiene alguna peculiaridad que lo hace poco intuitivo al usuario actual

Usando emacs



- menu
- pantalla edición
- línea de modo
- línea comandos

emacs $\neq$ xemacs



Atajos de teclado

- CTRL-K borrar linea
- ESC-X query-replace, ESC-X replace
- ESC-X goto-line
- CTRL-X-S salvar
- CTRL-X-F encontrar fichero
- CTRL-W=cortar, CTRL-Y=pegar
- CTRL-@=marca

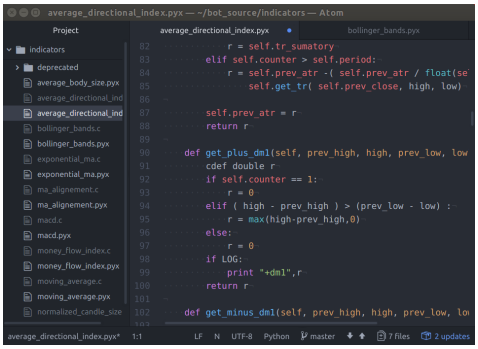
Enlaces sobre Emacs/XEmacs

- Emacs `http://www.gnu.org/software/emacs`
- XEmacs `http://www.xemacs.org`

21.5. Otros editores

21.5.1. Atom

Atom



- Editor de texto, libre y gratuito, disponible para Windows, Linux y macOS

Ventajas

- Más que un editor, es un IDE (Integrated development environment) con mucha funcionalidad: da formato, color, autocompleta, se integra con el compilador, con git, incluye colaboración en tiempo real (teletype)
- Ampliable mediante paquetes, que se pueden instalar desde el terminal (apm)
- Desarrollado por GitHub
- Moderno: la primera versión es de 2014, se ha vuelto muy popular

Inconvenientes

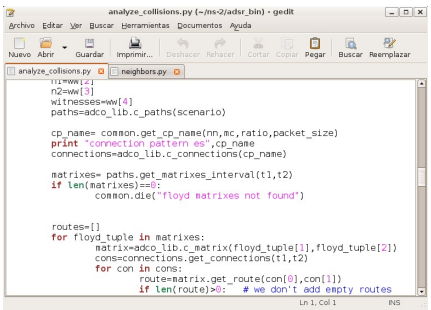
- Exige una sesión gráfica

enlaces

- `https://atom.io/`

21.5.2. gedit

gedit



Editor de texto de propósito general, es el *block de notas* de gnome

Ventajas

- Muy sencillo y fácil de manejar

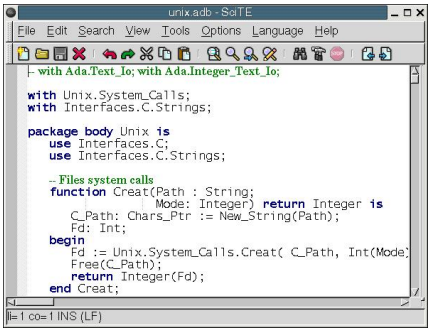
Inconvenientes

- Exige una sesión gráfica
- Ha mejorado mucho, pero sigue teniendo poca funcionalidad
- Tal vez no sea la mejor opción si tenemos disponible editores como atom, scite...

21.5.3. SciTE

SciTE

Editor de texto multiplataforma Ventajas



- Muy completo: Da formato, color, se integra con el compilador...
- Versiones para Win32 y X Window
- Muy fácil de manejar
- Es el editor de *anjuta*, el IDE de gnome

Inconvenientes

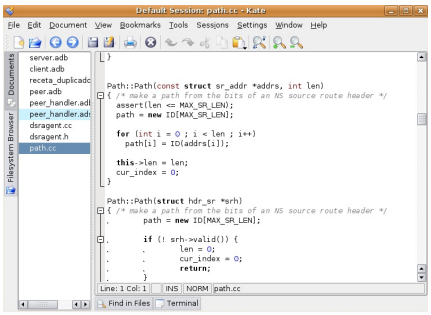
- Exige una sesión gráfica
- No muy extendido
- Hay editores más avanzados

enlaces

- `http://www.scintilla.org/SciTE.html`

21.5.4. Kate

Kate



Es el editor del escritorio KDE

Ventajas

- Muy completo: Da formato, color, se integra con el compilador...
- Muy buen *pretty printing*
- Muy fácil de manejar

Inconvenientes

- Exige una sesión gráfica
- No muy extendido
- Hay cosas editores más avanzados hacen mejor
- Es necesario tener instalado KDE (o al menos buena parte)
- No disponible en otras plataformas

Enlaces

- `http://kate-editor.org`

22. Markdown

Introducción

22.1. Introducción

Markdown es un lenguaje de marcado ligero

- Creado por John Gruber y Aaron Swartz en 2004
- Sintaxis muy sencilla, fácil de escribir a mano con un editor de texto plano
- No intenta reemplazar a lenguajes de marcado más potentes como HTML, Latex, PostScript, DocBook, etc
- Muy popular en la actualidad para entornos donde se desea una cierta *maquetación* del texto, no muy exigente: foros, blogs, mensajería instantánea, documentación de código fuente, ficheros *readme*, documentos internos, manuales de usuario *online*, etc

Herramientas para Markdown

22.2. Herramientas para Markdown

- El texto en Markdown se puede leer *tal cual* o se puede usar para generar otros formatos como html, pdf, rtf (para Microsoft Word), odt (para LibreOffice), entre otros. Empleando para ello
  - Grip, una herramienta muy sencilla para generar html desde markdown
    https://github.com/joeyespo/grip
  - Pandoc, una herramienta muy potente que convierte documentos desde y hasta múltiples formatos (Markdown, Html, epub, docbook, LaTex, RTF, ODT, ...)
    https://pandoc.org/
- También hay herramientas que facilitan su edición
  - Aplicaciones completas de escritorio como Abricotine, Remarkable o ReText
  - Plugins para editores como Atom, Vim o Visual Studio Code
  - Editores *online* como GitBook

Variantes de Markdown

- La definición inicial del lenguaje era bastante informal, con muchas imprecisiones. Herramientas muy variadas lo fueron incorporando, cada una con interpretaciones y extensiones ligeramente distintas en los aspectos no básicos
- Hay por tanto muchas variantes de Markdown. Tal vez el intento de normalización más extendido es GFM (*GitHub Flavored Markdown*, año 2017)

Párrafos

22.3. Párrafos

- Los saltos de línea *ordinarios* se ignoran. En el momento del *rendering*, la línea se rompe por donde donde corresponda
- Para hacer un párrafo (punto y aparte) es necesario escribir al menos una línea en blanco
- Para forzar un salto de línea, se pueden escribir dos espacios antes del salto de línea
    - Esto es problemático porque es invisible. Muchas herramientas admiten escribirlo en html: `<br>`

Secciones en el texto

22.4. Secciones en el texto

```
# Encabezado nivel 1  (Sección)
## Encabezado nivel 2 (Subsección)
(...)
###### Encabezado nivel 6 (Sub-sub-sub-sub-sub-sección)
```

Es necesario dejar un espacio entre la almohadilla y el título

Cursiva, negrita, enlaces

22.5. Cursiva, negrita, enlaces

Cursiva

- Una barra baja al comienzo y final del texto a resaltar `_ejemplo cursiva_`
- O bien un asterisco al comienzo y al final del texto a resaltar `*ejemplo cursiva*`

Negrita

- Dos barras bajas al comienzo y final del texto a resaltar `__ejemplo negrita__`
- O bien dos astericos al comienzo y al final del texto a resaltar `**ejemplo negrita**`

Enlaces

- Descripción del enlace, entre corchetes, seguido de la URL, entre paréntesis

```
[Universidad Rey Juan Carlos](https://urjc.es)
```

22.6. Imágenes

Imágenes (1)

Se pueden añadir imágenes con la misma sintaxis que los enlaces, pero añadiendo una admiración. Esta es una extensión GFM que no soportan todas las herramientas

- Fichero en una dirección web

```
![Logo de la URJC](https://gsyc.urjc.es/~mortuno/urjc.gif)
```

- Fichero en un trayecto absoluto de mi ordenador

```
![Pantallazo 1](/home/jperez/fotos/pantallazo01.png)
```

  Cuidado: incluir el trayecto absoluto de una cuenta de usuario casi siempre es un error: dejará de funcionar cuando el fichero lo abra otro usuario

Imágenes (2)

- Fichero en el mismo directorio que el documento actual

  ```
  ![Pantallazo 2](pantallazo02.png)
  ```

- Fichero en un subdirectorio del directorio actual

  ```
  ![Pantallazo 3](images/pantallazo03.png)
  ```

  Posiblemente esto es lo más recomendable: escribir el fichero en un subdirectorio llamado *images* dentro del directorio donde está el documento. Observa que NO es

  ```
  ![Pantallazo 3](/images/pantallazo01.png)
  ```

  La barra antes del directorio indicaría un trayecto absoluto

22.7. Listas

Listas sin ordenar

Se crean con un guión seguido de un espacio

```
- Sota
   - Sota de espadas
   - Sota de oros
- Caballo
  La figura del caballero, generalmente llamado
*caballo* es una peculiaridad de la baraja española
que sustituye a la figura de la reina que aparece
en la mayoría de las restantes barajas. Son cuatro:
   - Caballo de oros
   - ...
```

- Para hacer sublistas, añadimos indentación. Basta con hacerlo en la primera línea del párrafo. También usamos indentación para que el párrafo pertenezca a un elemento de la lista y no sea un párrafo distinto. Es necesario escribir el mismo número de espacios que en la línea que abrió ese nivel

- En vez de guiones se pueden usar asteriscos

Listas ordenadas

Se crean como las listas no ordenadas, pero con un número, un punto y un espacio

```
1. Análisis
1. Diseño
1. Codificación
1. Prueba
```

El número que escribamos es irrelevante. Puede ser una secuencia correcta como 1,2,3, todo unos, números fuera de secuencia...

22.8. Código

Código

- Código dentro de una línea

  Se escribe entre comillas invertidas. (la comilla a la derecha de la letra P). Ejemplo:

  ```
  Ejecuta la orden `ls -l`
  ```

- Bloques de código

  - Apertura:Tres comillas invertidas. Se puede añadir el nombre del lenguaje
  - Cierre: Tres comillas invertidas

  Propio de Github Markdown, no siempre disponible

  ```
  ```python
  #!/usr/bin/env python3

  def main():
      return

  if __name__ == "__main__":
      main()
  ```
  ```

22.9. Tablas

Tablas

En GFM también se pueden crear tablas

```
Hora   | Lunes | Martes | Miércoles
---|-------|--------|------
9:00 | Tal | Tal | Cual
11:00| Esto | Lo otro | Más allá
```

22.10. Generación de HTML

Generación de HTML

Fichero Markdown de ejemplo

- https://gsyc.urjc.es/ mortuno/md/ejemplo.md

Conversión a HTML con pandoc, sin plantilla de estilo

```
pandoc -s  ejemplo.md -o ejemplo_no_css.html
```

Resultado:

- https://gsyc.urjc.es/ mortuno/md/ejemplo_no_css.html

Es usable pero bastante *feo*. Mejora mucho con la hoja de estilos de pandoc

```
pandoc -s -c pandoc.css ejemplo.md -o ejemplo_css_01.html
```

Resultado:

- https://gsyc.urjc.es/ mortuno/md/ejemplo_css_01.html

En Internet encontraremos muchas otras hojas de estilo css. Por ejemplo esta:

https://gist.github.com/killercup/5917178

Resultado:

- https://gsyc.urjc.es/ mortuno/md/ejemplo_css_02.html

Lo más sencillo es dejar los ficheros css en el mismo directorio que el fichero md. Puedes obtenerlos desde stos enlaces:

- css de pandoc
- css de killercup

Otras opciones de pandoc

- Añadiendo a pandoc la opción `--toc` se crea una tabla de contenidos al principio el documento (un índice)
- Pandoc dará un *warning* porque nuestro documento no tiene título. En Markdown no hay forma estándar de incluir metainformación al documento, pero muchas herramientas soportan el formato de YAML para metadatos, justo al pricipio del documento

```
---
title: Introducción al formato Markdown
---
```

Ejemplo:
fpi_practica_08.md
fpi_practica_08_css_01.html
fpi_practica_08_css_02.html

Este script facilita el uso de pandoc

```bash
#!/bin/bash

# Leave uncomented the style sheet that you prefer
#TEMPLATE=md.css  # https://gist.github.com/killercup/5917178
#TEMPLATE=pandoc.css
TEMPLATE=https://gsyc.urjc.es/~mortuno/pandoc.css

if test $# -eq 0 || test $# -gt 1
then
    echo "Es necesario indicar un argumento (y solo uno)" >&2
    exit
fi

filename=$(basename "$1")
extension="${filename##*.}"
filename="${filename%.*}"

pandoc -s  --toc -c ${TEMPLATE} $1 -o ${filename}.html
```

- Recibe como argumento un fichero *.md* (o *.tex*, entre otros formatos)
- Genera una versión del documento en HTML
- Puedes descargarlo aquí: http://gsyc.urjc.es/ mortuno/mypandoc

Este script usa pandoc para convertir un fichero desde markdown hasta markdown. El formato final no cambia, pero resulta muy útil, porque *limpia* el fichero: líneas homogéneas, estilo homogéneo, etc

```
#!/bin/bash

# -s  standalone document
# -S typographically smart


if test $# -eq 0 || test $# -gt 1
then
    echo "Es necesario indicar un argumento (y solo uno)" >&2
    exit
fi

filename=$(basename "$1")
extension="${filename##*.}"
filename="${filename%.*}"

if test $extension != md
then
    echo "Es necesario que la extensión sea .md" >&2
    exit
fi

tmp_name=/tmp/${filename}.$$.md
pandoc -s  $1 -o ${tmp_name}
mv ${tmp_name} $1
```

Puedes descargarlo aquí: http://gsyc.urjc.es/ mortuno/clean_md

grip

Una herramienta alternativa a pandoc para convertir ficheros markdown en ficheros HTML es grip

- En ocasiones genera HTML de mayor calidad
- Actualmente no permite insertar css, pero podemos añadirlo nosotros fácilmente, basta añadir un elemento `<style>` o mejor un `<link>`

Para instalar grip, ejecutamos (con privilegios de administrador)

```
sudo apt install grip
```

Para usarlo:

```
grip FICHERO_ENTRADA.md  --export FICHERO_SALIDA.html
```

Referencias

https://guides.github.com/features/mastering-markdown/
https://www.markdownguide.org/basic-syntax/