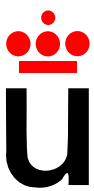
Introducción y estructura del sistema

Laboratorio de Sistemas

Enrique Soriano, Gorka Guardiola

GSYC

1 de marzo de 2021



Universidad Rey Juan Carlos

Historia


Western Electric
A08A
Bell Laboratories
Western Electric
ALLENTOWN
Home of the Transistor

Historia

- 1945-1955: todo se hacía desde cero, se usan relevadores electromecánicos y después los tubos de vacío.
- 1955-1965: aparecen lenguajes de programación (FORTRAN) y sistemas operativos básicos (bibliotecas). Aparecen los circuitos integrados, sistemas por lotes.
- 1965-1980: aparecen lenguajes de programación de alto nivel (C, COBOL), minicomputadoras, multiprogramación y tiempo compartido. Los sistemas operativos evolucionan: MULTICS y después UNIX.
- 1980: aparecen los ordenadores personales, redes de ordenadores, alto nivel de integración, distintos paradigmas de lenguajes de alto nivel, sistemas operativos modernos.

¿Qué es UNIX?

UNIX fue un sistema operativo creado por Ken Thompson y Dennis Ritchie, empezaron en 1969:

"...the number of UNIX installations has grown to 10, with more expected..."
- Dennis Ritchie and Ken Thompson, June 1972

¿Qué es UNIX?

Hay muchos sistemas derivados y reimplementaciones, se llaman sistemas *UNIX-like*[1]:

1BSD, 2BSD, 3BSD, 4BSD, 4.4BSD Lite 1, 4.4BSD Lite 2, 386 BSD, Acorn RISC iX, Acorn RISC Unix, AIX, AIX PS/2, AIX/370, AIX/6000, AIX/ESA, AIX/RT, AMiX, Android , AOS Lite, AOS Reno, AppleTV, ArchBSD, ASV, Atari Unix, A/UX, BBX, BOS, BRL Unix, BSD Net/1, BSD Net/2, BSD/386, BSD/OS, CB Unix, Chorus, Chorus/MiX, Coherent, CTIX, CXOs, Darwin, Debian GNU/Hurd, DEC OSF/1 ACP, Dell Unix, DesktopBSD, Digital Unix, DragonFly BSD, Dynix, Dynix/ptx, ekkoBSD, Eunice, FireFly BSD, FreeBSD, FreeDarwin, GNU, GNU-Darwin, Gnuppix GNU/Hurd-L4, HPBSD, HP-UX, HP-UX BLS, IBM AOS, IBM IX/370, Inferno, Interactive 386/ix, Interactive IS,iOS, iPhone OS, iPod OS, IRIS GL2, IRIX, Junos OS, Lites, LSX, macOS (Mac OS X), Mach, MERT, MicroBSD, MidnightBSD, Mini Unix, Minix, Minix-VMD, MIPS OS RISC/os, MirBSD, Mk Linux, Monterey, more/BSD, mt Xinu, MVS/ESA OpenEdition, NetBSD, NeXTSTEP, NonStop-UX, Open Desktop, Open UNIX, OpenBSD, OpenDarwin, OpenIndiana, OpenServer, OpenSolaris, OPENSTEP, OS/390 OpenEdition, OS/390 Unix, OSF/1, OS X, PC-BSD, PC/IX,Plan 9, Plurix, PureDarwin, PWB, PWB/UNIX, QNX, QNX RTOS, QNX/Neutrino, QUNIX, ReliantUnix, Rhapsody, RISC iX, RT, SCO UNIX, SCO UnixWare, SCO Xenix, SCO Xenix System V/386, Security-Enhanced Linux, Silver OS, Sinix, Sinix ReliantUnix,Solaris, SPIX, SunOS, Triance OS, Tru64 Unix, Trusted IRIX/B, Trusted Solaris, Trusted Xenix, TS, Tunis, UCLA Locus, UCLA Secure Unix, Ultrix, Ultrix 32M, Ultrix-11, Unicos, Unicos/mk, Unicos/mp, Unicox-max, UNICS, UniSoft UniPlus, UNIX 32V, UNIX Interactive, UNIX System III, UNIX System IV, UNIX System V, UNIX System V Release 2, UNIX System V Release 3, UNIX System V Release 4, UNIX System V/286, UNIX System V/386, UNIX Time-Sharing System, UnixWare, UNSW, USG, Venix, Xenix OS, Xinu, xMach, z/OS Unix System Services, ...

... nosotros nos centraremos en uno llamado **GNU/Linux**

[1] Ver https://www.levenez.com/unix/

¿Qué es un sistema operativo?

- Def.- Programas que te dejan usar la máquina, es subjetivo.
- Ventajas:
  - Similar a una biblioteca $\rightarrow$ reutilización.
  - Abstrae de la máquina: no necesitas conocer los detalles para usarla.
  - Gestiona y reparte la máquina: no necesitas preocuparte de gestionar el tiempo que ejecuta un programa, organizarle la memoria, etc.

¿Qué es un sistema operativo?

- Es una **máquina abstracta**: el sistema operativo proporciona una máquina que realmente no existe, es una máquina ficticia que nos ofrece dispositivos virtuales: ficheros, directorios, procesos, conexiones de red, ventanas...
- Hoy en día tenemos distintos tipos de software de sistemas: sistema operativo, hipervisores, contenedores, etc.

¿Qué es un sistema operativo?

Es un gestor de recursos:

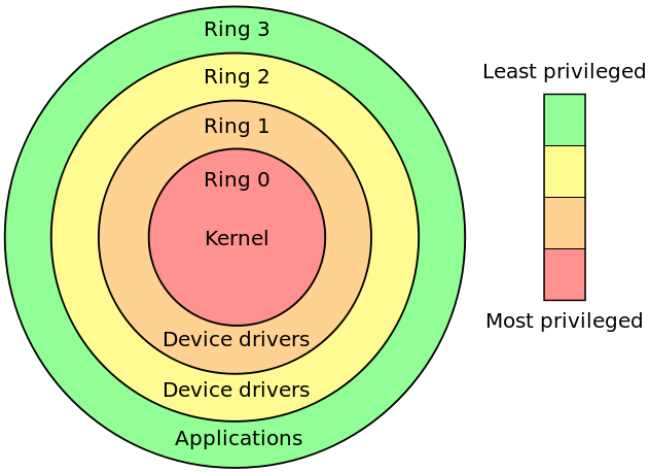
- Multiplexa en tiempo: p. ej. procesador, red.
  - ¿Tienes que preocuparte de soltar el procesador en tu aplicación?
  - Ejemplo: abstracción llamada **proceso**.
- Multiplexa en espacio: p. ej. memoria, disco.
  - ¿Tienes que preocuparte de no pisar la memoria de otra aplicación?
  - Ejemplo: abstracción llamada **fichero**.

Procesos y programas

- Programa: conjunto de datos e instrucciones que implementan un algoritmo.
- Proceso: programa que está en ejecución, un programa vivo que tiene su propio **flujo de control** y es independiente de los otros procesos.
- El sistema operativo crea la ilusión de que cada proceso tiene su propia CPU.

Niveles de privilegio de la CPU

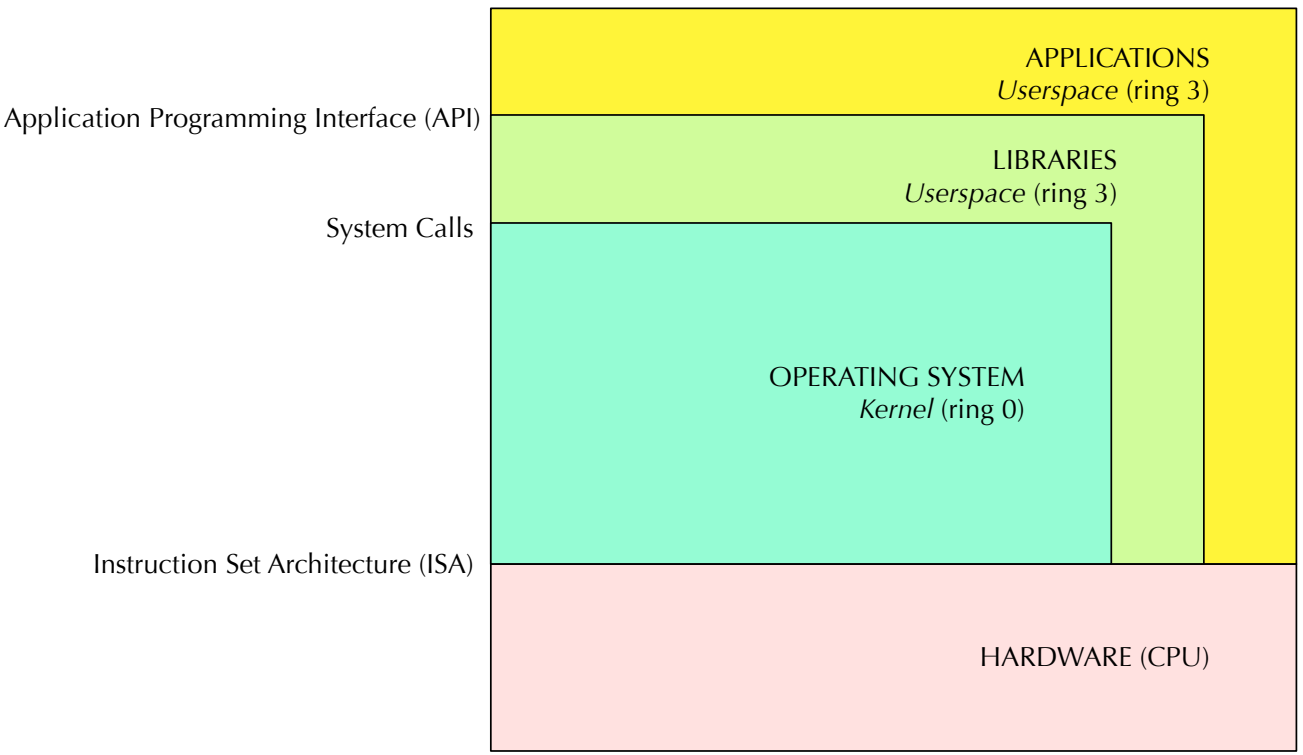
Visión clásica:



Las máquinas modernas tienen otros niveles *negativos*, el sistema operativo no es consciente de ellos:

- Ring -1: Intel VT-x, AMD-V, soporte para virtualización de sistema opertivo.
- Ring -2: System Management Mode (SMM), ejecuta el firmware de la máquina para gestionar la energía, manejar errores del hardware, etc.
- Ring -3: Intel ME, sistema que que ejecuta en una CPU separada y activo en todo momento, con acceso a toda la memoria física, a las interfaces de red, etc. Este sistema se activa antes de arrancar la CPU principal.

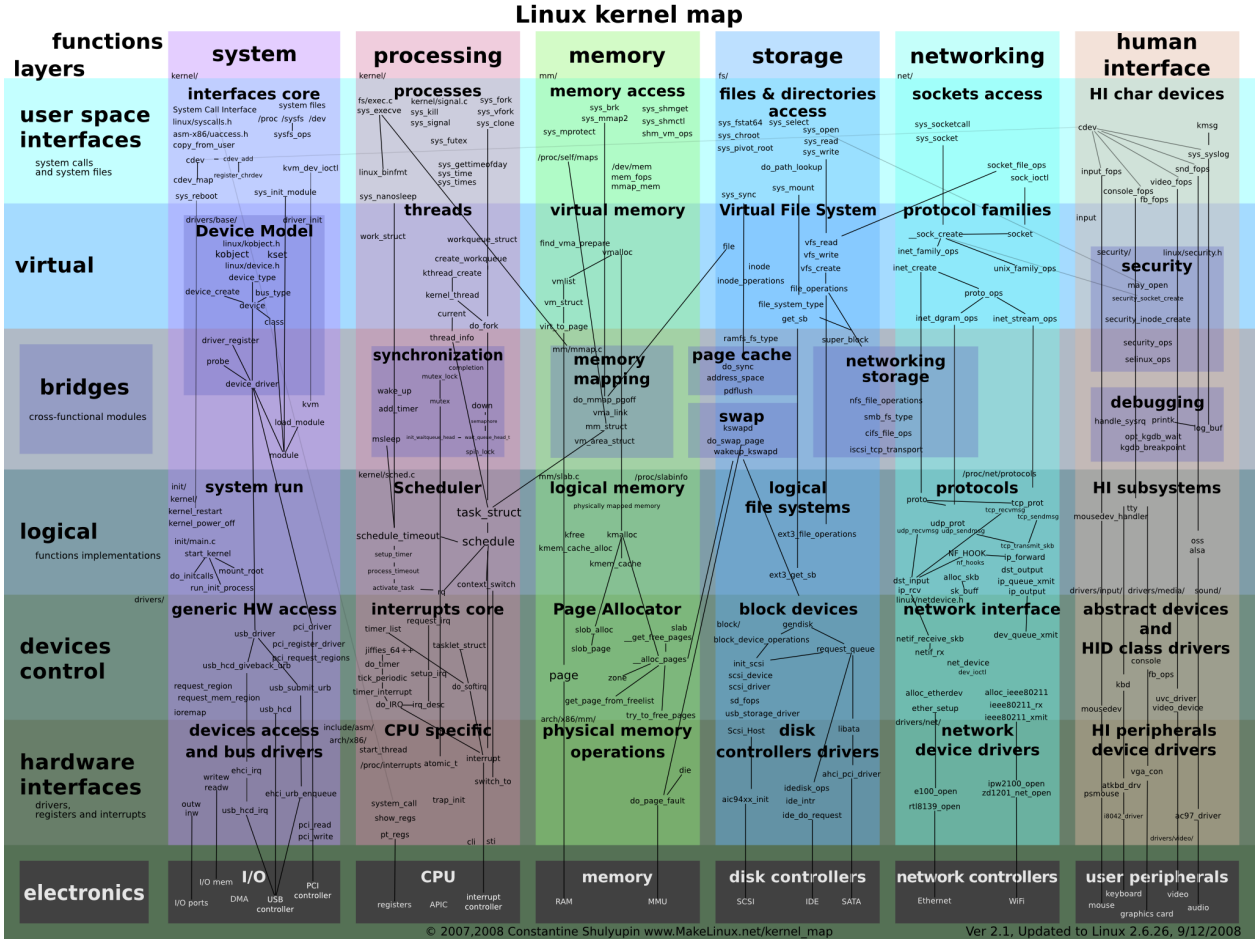
Estructura del sistema



Núcleo (*kernel*)

- Ejecución en **modo privilegiado** (ring 0): se pueden ejecutar instrucciones especiales (acceder a ciertos registros de la CPU, invalidar caches, etc.).
- Multiplexa la máquina (espacio y tiempo): implementa las **políticas y mecanismos** para repartir la CPU, memoria, disco, red,...
- Maneja el hardware: **drivers**.
- Proporciona **abstracciones**:
  - **Proceso**: programa en ejecución.
  - **Fichero**: datos agrupados bajo un nombre.
  - ...
- Da servicio al resto de programas en ejecución, que no ejecutan en modo privilegiado. Si el kernel es **reentrante**, puede dar servicio a múltiples simultáneamente. Todos los kernels de tipo UNIX lo son.

Núcleo (*kernel*)



Área de usuario (*userspace, userland*)

- Así ejecutan los programas del usuario (aplicaciones, herramientas, GUI, etc.).
- Se ejecutan en **modo no privilegiado** (ring 3): no se pueden ejecutar instrucciones peligrosas.
- Piden servicio al kernel realizando **llamadas al sistema**.

Kernel modular

La mayoría de los kernels actuales permiten la carga dinámica de módulos para ampliar/reducir su funcionalidad sin la necesidad de rearrancar el sistema.

- Ventaja: sólo se cargan los drivers necesarios $\rightarrow$ ahorro de memoria.
- Desventaja: seguridad.
- Ejemplos: Linux (.ko), Mac OSX (.kext), FreeBSD (.kld), Windows (.sys).

Distribuciones Linux

- Una distribución es una colección concreta de software de área de usuario y un núcleo del sistema operativo.
- Hay muchas distribuciones de Linux, agrupadas en familias. Hay unas 600 distribuciones.
- Las que usan el kernel de Linux y las herramientas de área de usuario de GNU se denominan GNU/Linux. Por ejemplo: Debian, Ubuntu, Red Hat, SUSE, etc.
- Una distribución suele tener su sistema de gestión de paquetes para instalar el software. Por ejemplo: apt es el sistema de paquetes de las distribuciones basadas en Debian, el formato de los paquetes es .deb. RPM es el sistema de paquetes de Red Hat.

Distribuciones Linux

- Cambian cosas entre distribuciones: instalador, estructura del árbol de ficheros, algunas herramientas, software propietario, etc.
- Cada distribución tiene su esquema de versiones (números, versiones con soporte a largo/corto plazo, etc.).
- Cuando bajamos una distribución, necesitamos saber la arquitectura de la máquina (amd64, x86, arm, etc.).
- Hay distribuciones con soporte comercial: Ubuntu, Red Hat, SUSE, etc.
- Hay distribuciones dirigidas a nichos: seguridad (p. ej. Kali), privacidad (p. ej. Tails), routers (p. ej. OpenWrt), ocio (p. ej. Kodi), distintas plataformas (p. ej. Raspian), ...

Uso básico del sistema

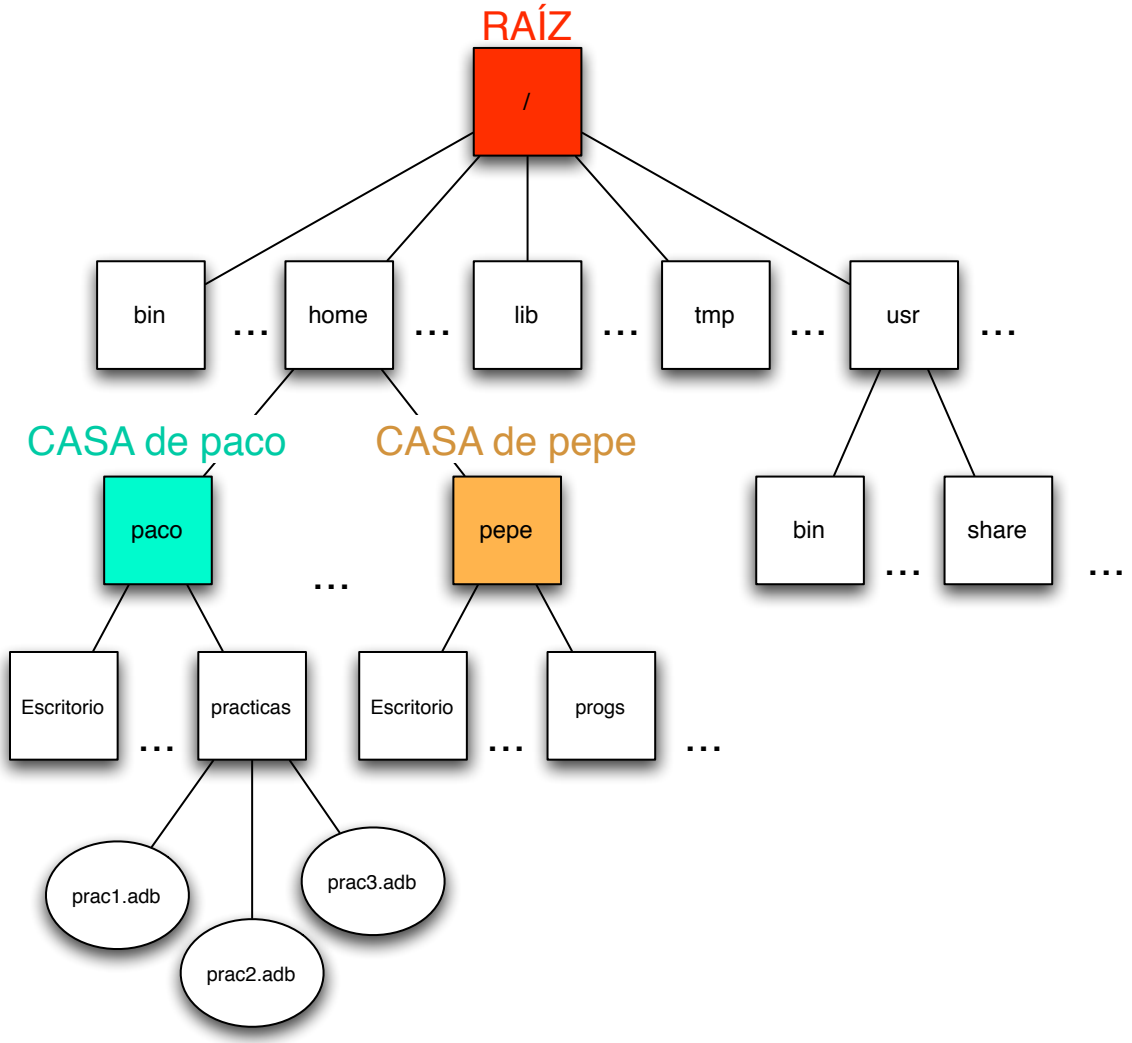
Definiciones

- **Comando o mandato** (command): cadena de texto que identifica a un programa u orden.
- **Shell**: programa que te deja ejecutar *comandos*. Por lo general, permite crear programas (scripts) en un lenguaje propio. Hay distintos tipos de shells, usaremos `bash`. Básicamente, un shell hace esto:
  1. leer un línea de comandos
  2. sustituir algunas cosas en esa línea
  3. crear los procesos para ejecutar los comandos descritos por la línea
- **Prompt**: texto que indica que el **shell** está esperando una orden.
- **Usuario** (login name): el nombre de usuario, todos los programas ejecutan en nombre de un usuario.
- **root**: superusuario o administrador del sistema.

Ficheros y directorios

- Organizados en **árbol** → directorio **raíz** (root).
- Dos ficheros que están en distintos directorios son dos ficheros diferentes.
- **Directorio de trabajo**, pwd.
- **Directorio casa**, $HOME

Árbol de ficheros



Directorios en GNU/Linux

- /bin tiene ejecutables.
- /dev tiene dispositivos.
- /etc tiene ficheros de configuración.
- /home tiene los datos personales de los usuarios.
- /lib tiene las bibliotecas (código) que usan los programas ejecutables.
- /proc y /sys ofrecen una interfaz para interaccionar con el núcleo del sistema.
- /sbin tiene los ejecutables del sistema.
- /tmp sirve para almacenar los ficheros temporales, se borra en cada reinicio.
- /usr existe por razones históricas (tamaño de almacenamiento) y contiene gran parte del sistema: contiene directorios similares a los anteriores (/usr/bin o /usr/lib), con los datos y recursos para los programas de usuario (no del sistema).
- /var tiene los datos que se generan en tiempo de ejecución (cache, logs y otros ficheros que generan los programas).
- /boot continen los ficheros de arranque del sistema.
- /media y /mnt puntos de montaje
- /opt contiene ficheros para programas *de terceros*.

Rutas (paths)

- Ruta absoluta: serie de directorios desde el raíz separados por barras.
  - `/home/alumnos/pepe/fichero.txt`
- Ruta relativa: serie de directorios desde el directorio actual.
  - `alumnos/pepe/fichero.txt`
- .. : directorio padre.
  - `../pepe/fichero.txt`
- . : directorio actual.
  - `./fich1`
- Para indicar que queremos ejecutar un fichero del directorio actual:
  - `./miprograma`

Texto plano

- ASCII: usa 1 byte para representar 128 caracteres (7 bits + 1 bit).
- ISO-Latin 1 (8859-1): usa 1 byte para almacenar caracteres (8 bits).
- UTF-8: puede usar 1,2, o más bytes. Compatible hacia atrás.
- Hay muchas otras.

Ficheros de texto plano

Caracteres de control:

- El carácter `'\n'` indica nueva línea en el texto. Es una convención usada en todos los programas, bibliotecas, etc. en Unix.
- En otros sistemas operativos no tiene por qué ser así (Windows usa la secuencia `'\n\r'`).
- El carácter `'\t'` indica un tabulador.
- No hay carácter EOF: invención de los lenguajes.

Manual

- Las páginas de manual se pueden consultar con el comando `man`: `man sección asunto`
  Por ejemplo: `man 1 gcc`
- Secciones de interés: comandos (1), llamadas al sistema(2), llamadas a biblioteca(3).
- Para buscar sobre una palabra: `apropos`.
  Por ejemplo: `apropos gcc`.

Comandos básicos

- `cd`: cambia de directorio actual.
- `echo`: escribe sus argumentos por su salida.
- `touch`: cambia la fecha de modificación de un fichero. Si no existe el fichero, se crea.
- `ls`: lista el contenido de un directorio.
- `cp`: copia ficheros.
- `mv`: mueve ficheros.
- `rm`: borra ficheros.
- `mkdir`: crea directorios.
- `rmdir`: borra directorios vacíos.
- `date`: muestra la fecha.

Comandos básicos

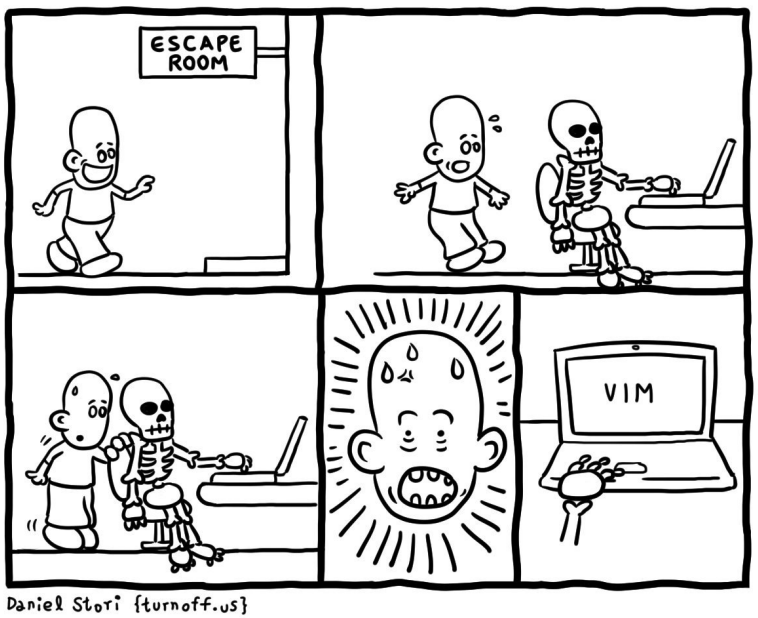
- `who`: muestra los usuarios que están en el sistema.
- `whoami`: muestra tu nombre de usuario.
- `sort`: ordena las líneas de un fichero.
- `wc`: cuenta caracteres, palabras y líneas de ficheros.
- `fgrep, grep`: buscan cadenas dentro de ficheros.
- `cmp, diff`: comparan ficheros.
- `cat`: escribe en su salida el contenido de uno o varios ficheros.
- `less`: permite leer un fichero de texto en el terminal usando *scroll*.

Comandos básicos

- `file`: da pistas sobre el contenido de un fichero.
- `od`: escribe en su salida el los datos de un fichero en distintos formatos.
- `head, tail`: escriben el las primeras/últimas líneas del fichero en su salida.
- `tar`: crea un fichero con múltiples ficheros dentro (comprimidos o no).
- `gzip/gunzip`: comprime/descomprime un fichero.
- `top`: muestra los procesos y el estado de sistema.
- `reset`: restablece el estado del terminal.
- `exit`: el shell termina su ejecución.

Editor en modo texto: vi

- Hay múltiples editores para usar en el terminal (nano, pico, vim, emacs, etc.).
- **vi** es el editor clásico.
- **vim** es un editor basado en **vi**. Implementa un superconjunto. En ciertas distribuciones, `vi` es en realidad `vim`.
- Tiene fama de ser complicado :)



Vi: comandos para sobrevivir

Tiene dos modos: modo inserción (para escribir) y modo comando.

- **i** pasa a modo inserción
- **ESC** pasa a modo comando

En modo comando:

- **:q!** sale del editor sin guardar
- **:x** salva el fichero y sale (también se puede con **:wq**)
- **:w** salva el fichero
- **:número** se mueve a esa línea del fichero
- **i** inserta antes del cursor
- **a** inserta después del cursor
- **o** inserta en una línea nueva
- **dd** corta una línea
- **p** pega la línea cortada anteriormente
- **h, j, k, l** mueve el cursor a izquierda, abajo, arriba, derecha

Variables

- Variable de shell: son locales a la shell, los programas ejecutados por el shell no tienen dichas variables.
- Variable de entorno: los programas ejecutados por el shell sí tienen su propia copia de la variable, con el mismo valor.
- `mivar=hola` define la variable de shell con nombre *mivar*, cuyo valor será *hola*.
- `$mivar` el shell sustituye eso por el valor de dicha variable (si no existe, lo sustituye por nada).
- `export mivar` exporta la variable (ahora es una variable de entorno).

Variables

- El comando `set` muestra todas las variables (de shell y de entorno).
- El comando `printenv` muestra las variables de entorno (también lo hace el comando `env`).
- El comado `unset` elimina una variable.
- Variables populares:
  - $PATH: la ruta de los programas.
  - $HOME: la ruta de tu directorio casa.
  - $USER: el nombre de usuario
  - $PWD: la ruta actual del shell
  - $LANG: configuración de localización (*locales*).
  - $LC_xxx: otras variables de localización (*locales*).

Usando el terminal

Globbing (wildcards): caracteres especiales para el shell que sirven para hacer referencia a nombres de ficheros:

- **?** cualquier carácter.
- * cualquier secuencia de caracteres.
- **[ab]** cualquiera de los caracteres que están dentro de los corchetes (letra a o la letra b en el ejemplo).
- **[b-z]** cualquier carácter que se encuentre entre esas dos (de la letra b a la z en el ejemplo).

Para escribir caracteres especiales sin que haya sustitución:

- ' ' las comillas simples *escapan* todo lo que tienen dentro (ya no tienen un significado especial).
- " " las comillas dobles *escapan* todo menos algunas sustituciones (p. ej. las variables de entorno).

Usando el terminal

- ↑ repite los comandos ejecutados anteriormente en la shell.
- El tabulador completa nombres de ficheros.
- Ctrl+r deja buscar comandos que ejecutamos hace tiempo.
- Ctrl+c mata el programa que se está ejecutando.
- Ctrl+z detiene el programa que se está ejecutando.
- Ctrl+d termina la entrada (o manda lo pendiente).
- Ctrl+a: mueve el cursor al principio de la línea.
- Ctrl+e: mueve el cursor al final de la línea.
- Ctrl+w: borra la palabra anterior en la línea.
- Ctrl+u: borra desde el cursor hasta el principio de la línea.
- Ctrl+k: borra desde el cursor hasta el final de la línea.
- Ctrl+s: congela el terminal. Ctrl+q lo descongela.
- Ctrl+l: limpia el terminal.

Tabulador

- En alguna distribuciones, el tabulador escapa el $ en una variable en la shell bash.
- $HOME/t y apretar el tabulador completa \$HOME/t en lugar de /home/paurea/tmp.
- shopt -s direxpand o shopt -s cdable_vars (producen comportamientos levemente diferentes, pruébalos) deshabilitan este comportamiento.
- Para que sea permanente, `cd` y
  `echo "shopt -s direxpand" >> .bashrc`