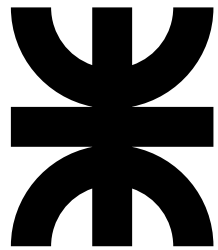
Informática II

Introducción al sistema operativo GNU/Linux, bash y make

Gonzalo F. Pérez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2016 –

Introducción

Sistemas operativos: historia y contexto

- ¿Qué es un sistema operativo?

Introducción

Sistemas operativos: historia y contexto

- ¿Qué es un sistema operativo?
  - Programa o conjunto de programas para administrar los recursos de hardware y dar servicios a los programas de aplicación (software)
  - Administración: tareas (scheduler), memoria, red, seguridad, disco.

Introducción

Sistemas operativos: historia y contexto

- ¿Qué es un sistema operativo?
  - Programa o conjunto de programas para administrar los recursos de hardware y dar servicios a los programas de aplicación (software)
  - Administración: tareas (scheduler), memoria, red, seguridad, disco.
- ¿Cuales conocen?

Introducción

Sistemas operativos: historia y contexto

- ¿Qué es un sistema operativo?
  - Programa o conjunto de programas para administrar los recursos de hardware y dar servicios a los programas de aplicación (software)
  - Administración: tareas (scheduler), memoria, red, seguridad, disco.
- ¿Cuales conocen?
  - Windows

Introducción

Sistemas operativos: historia y contexto

- ¿Qué es un sistema operativo?
  - Programa o conjunto de programas para administrar los recursos de hardware y dar servicios a los programas de aplicación (software)
  - Administración: tareas (scheduler), memoria, red, seguridad, disco.
- ¿Cuales conocen?
  - Windows
  - Android

Introducción

Sistemas operativos: historia y contexto

- ¿Qué es un sistema operativo?
  - Programa o conjunto de programas para administrar los recursos de hardware y dar servicios a los programas de aplicación (software)
  - Administración: tareas (scheduler), memoria, red, seguridad, disco.
- ¿Cuales conocen?
  - Windows
  - Android
  - Unix

Introducción

Sistemas operativos: historia y contexto

- ¿Qué es un sistema operativo?
  - Programa o conjunto de programas para administrar los recursos de hardware y dar servicios a los programas de aplicación (software)
  - Administración: tareas (scheduler), memoria, red, seguridad, disco.
- ¿Cuales conocen?
  - Windows
  - Android
  - Unix
  - Mac OS

Introducción

Sistemas operativos: historia y contexto

- ¿Qué es un sistema operativo?
  - Programa o conjunto de programas para administrar los recursos de hardware y dar servicios a los programas de aplicación (software)
  - Administración: tareas (scheduler), memoria, red, seguridad, disco.

- ¿Cuales conocen?
  - Windows
  - Android
  - Unix
  - Mac OS
  - BSD[1]

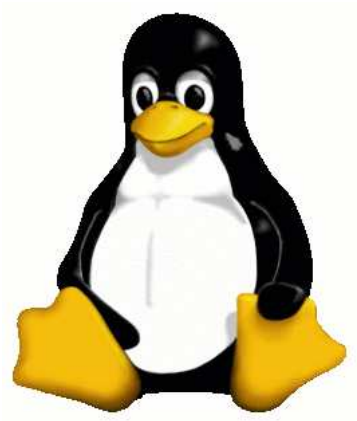
[1] Berkeley Software Distribution, desarrollado desde 1970 (Univ. de California)

Introducción

Sistemas operativos: historia y contexto

- ¿Qué es un sistema operativo?
  - Programa o conjunto de programas para administrar los recursos de hardware y dar servicios a los programas de aplicación (software)
  - Administración: tareas (scheduler), memoria, red, seguridad, disco.

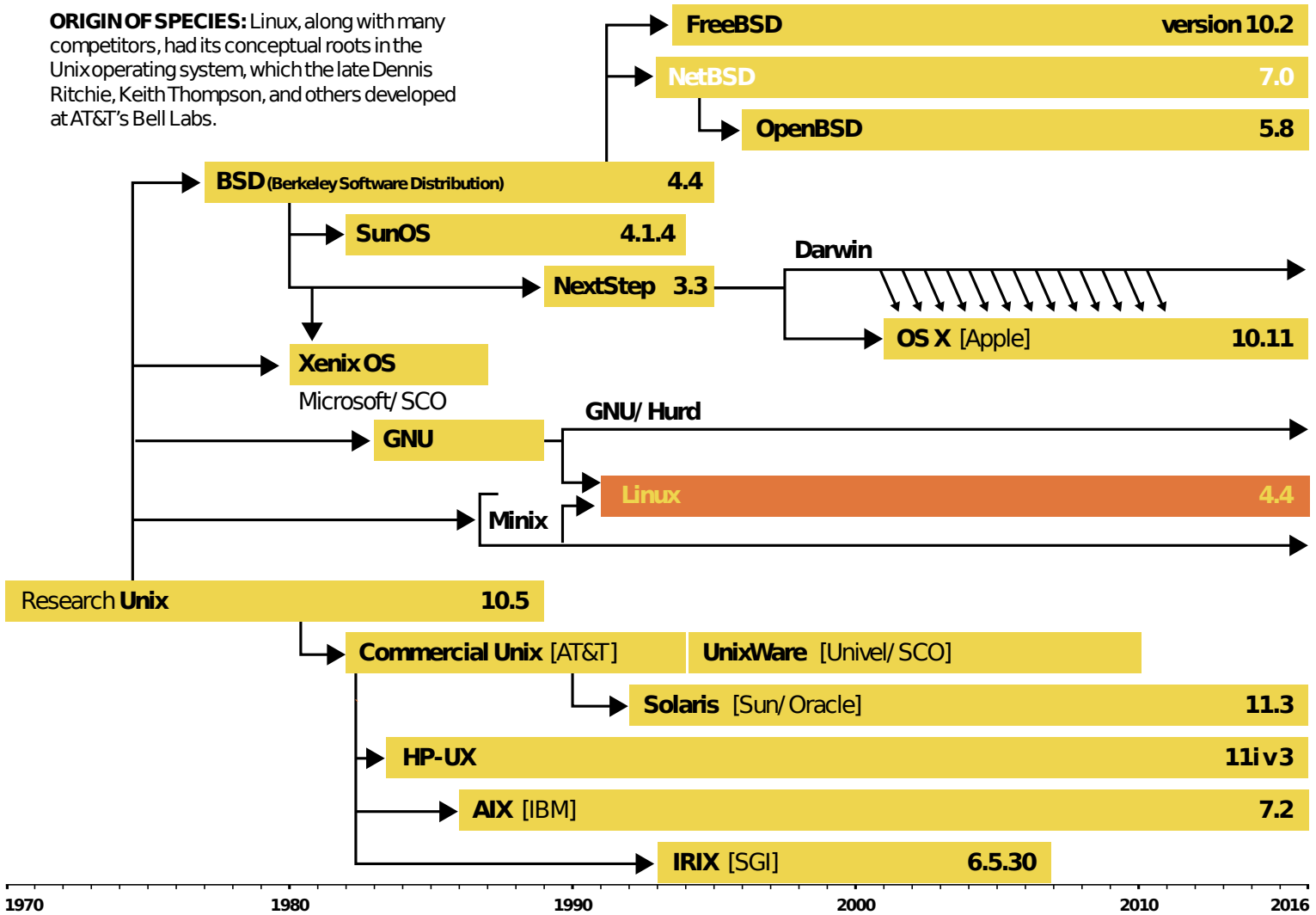
- ¿Cuales conocen?
  - Windows
  - Android
  - Unix
  - Mac OS
  - BSD[1]
  - GNU/Linux[2]



[1] Berkeley Software Distribution, desarrollado desde 1970 (Univ. de California)
[2] GNU: "GNU's Not Unix (clon de Unix), bajo licencia GPL

Introducción

Sistemas operativos: historia y contexto



De *IEEE Spectrum*, abril de 2016

Introducción

GNU/Linux

Richard Stallman

Richard Matthew Stallman (nacido en Manhattan, Nueva York, 16 de marzo de 1953), con frecuencia abreviado como «**rms**»,[1] es un programador estadounidense y fundador del movimiento por el software libre en el mundo.

Entre sus logros destacados como programador se incluye la realización del editor de texto GNU Emacs,[2] el compilador GCC,[3] y el depurador GDB,[4] bajo la rúbrica del Proyecto GNU. Sin embargo, es principalmente conocido por el establecimiento de un marco de referencia moral, político y legal para el movimiento del software libre, como una alternativa al desarrollo y distribución del software no libre o privativo. Es también inventor del concepto de copyleft (aunque no del término), un método para licenciar software de tal forma que su uso y modificación permanezcan siempre libres y queden en la comunidad de usuarios y desarrolladores.

Índice [ocultar]

Richard Matthew Stallman

Richard Stallman en mayo de 2016

Richard Stallman inició el proyecto GNU en enero de 1984

Linus Torvalds

Linus Benedict Torvalds (28 de diciembre de 1969, Helsinki, Finlandia) es un ingeniero de software finés americano,[1] conocido por iniciar y mantener el desarrollo del "*kernel*" (en español, núcleo) Linux, basándose en el sistema operativo libre Minix creado por Andrew S. Tanenbaum y en algunas herramientas, varias utilidades y los compiladores desarrollados por el proyecto GNU. Actualmente Torvalds es responsable de la coordinación del proyecto. Pertenece a la comunidad sueco-parlante de Finlandia.

Sus padres tomaron su nombre de Linus Pauling (estadounidense, Premio Nobel de Química 1954). Comenzó sus andanzas informáticas a los 11 años cuando su abuelo, un matemático y estadístico de la Universidad, compró uno de los primeros microordenadores Commodore en 1980 y le pidió ayuda para usarlo.[2]

A finales de los años 80 tomó contacto con los ordenadores IBM, PC y en 1991 adquirió un ordenador con procesador modelo 80386 de Intel.

En 1988 fue admitido en la Universidad de Helsinki, donde estudio Ciencias de la Computación. Ese mismo año el profesor Andrew S. Tanenbaum saca a la luz el S.O. Minix con propósitos didácticos. Dos años después, en 1990, Torvalds

Linus Torvalds

Linus Torvalds liberó el código del Kernel de Linux en agosto de 1991 (PC 386)

Introducción
GNU/Linux

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them 😀

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes – it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have 😕.

Introducción
GNU/Linux

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them 😀

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes – it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have 😕.

- Linux 1.0 lanzado en 1994
- Desarrollado a partir de Minix (1987), usado por Linus (BSD no corría en PC, y GNU Hurd no estaba listo)
- POXIS (Portable Operating System Interface for Unix): estándar para sistemas operativo tipo Unix (IEEE-CS)
- Propiedades importantes de Linux: Estabilidad, Seguro, No necesita rebuteo frecuente, Portabilidad&Escalabilidad, etc.
- GNU: Licenciado bajo GPL (General Public License)
- Distribuciones (Linux flavors)

Introducción
GNU/Linux

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them 😀

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes – it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have 😕.

- Linux 1.0 lanzado en 1994
- Desarrollado a partir de Minix (1987), usado por Linus (BSD no corría en PC, y GNU Hurd no estaba listo)
- POXIS (Portable Operating System Interface for Unix): estándar para sistemas operativo tipo Unix (IEEE-CS)
- Propiedades importantes de Linux: Estabilidad, Seguro, No necesita rebuteo frecuente, Portabilidad&Escalabilidad, etc.
- GNU: Licenciado bajo GPL (General Public License)
- Distribuciones (Linux flavors)

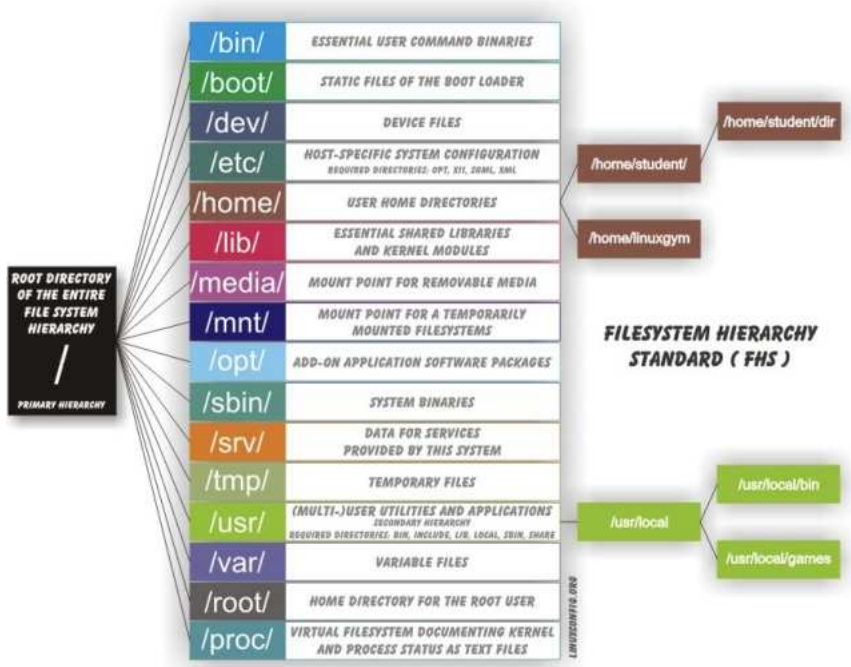
Memorable Linux Milestones

Componentes de un sistema GNU/Linux:

- El kernel/núcleo (Linux)
- El shell o intérprete de comandos (bash, ash, csh, etc.)
- Procesos y archivos/sistema de archivos (File System)
- Entorno de escritorio

Componentes de un sistema GNU/Linux:

- El kernel/núcleo (Linux)
- El shell o intérprete de comandos (bash, ash, csh, etc.)
- Procesos y archivos/sistema de archivos (File System)
- Entorno de escritorio



Software libre

GNU/Linux

- Software libre vs. privativo

Software libre

GNU/Linux

- Software libre vs. privativo
- Software libre vs. gratis

Software libre

GNU/Linux

- Software libre vs. privativo
- Software libre vs. gratis
- RMS: en 1983 inicia oficialmente el proyecto GNU, en 1985 crea la Free Software Foundation, luego redacta la licencia GPL

Software libre

GNU/Linux

- Software libre vs. privativo
- Software libre vs. gratis
- RMS: en 1983 inicia oficialmente el proyecto GNU, en 1985 crea la Free Software Foundation, luego redacta la licencia GPL

Resumen GPL

- Cualquiera es libre de utilizar el Software Libre para cualquier propósito.

Software libre

GNU/Linux

- Software libre vs. privativo
- Software libre vs. gratis
- RMS: en 1983 inicia oficialmente el proyecto GNU, en 1985 crea la Free Software Foundation, luego redacta la licencia GPL

Resumen GPL

- Cualquiera es libre de utilizar el Software Libre para cualquier propósito.
- Cualquiera es libre de acceder a su código fuente y estudiarlo.

Software libre

GNU/Linux

- Software libre vs. privativo
- Software libre vs. gratis
- RMS: en 1983 inicia oficialmente el proyecto GNU, en 1985 crea la Free Software Foundation, luego redacta la licencia GPL

Resumen GPL

- Cualquiera es libre de utilizar el Software Libre para cualquier propósito.
- Cualquiera es libre de acceder a su código fuente y estudiarlo.
- Cualquiera es libre de distribuirlo.

Software libre

GNU/Linux

- Software libre vs. privativo
- Software libre vs. gratis
- RMS: en 1983 inicia oficialmente el proyecto GNU, en 1985 crea la Free Software Foundation, luego redacta la licencia GPL

Resumen GPL

- Cualquiera es libre de utilizar el Software Libre para cualquier propósito.
- Cualquiera es libre de acceder a su código fuente y estudiarlo.
- Cualquiera es libre de distribuirlo.
- Cualquiera es libre de mejorarlo o adaptarlo y de distribuir el programa modificado.

Software libre

GNU/Linux

- Software libre vs. privativo
- Software libre vs. gratis
- RMS: en 1983 inicia oficialmente el proyecto GNU, en 1985 crea la Free Software Foundation, luego redacta la licencia GPL

Resumen GPL

- Cualquiera es libre de utilizar el Software Libre para cualquier propósito.
- Cualquiera es libre de acceder a su código fuente y estudiarlo.
- Cualquiera es libre de distribuirlo.
- Cualquiera es libre de mejorarlo o adaptarlo y de distribuir el programa modificado.
- La única obligación es que si se distribuye, haya que hacerlo bajo la misma licencia GPL.

Software libre

GNU/Linux

- Software libre vs. privativo
- Software libre vs. gratis
- RMS: en 1983 inicia oficialmente el proyecto GNU, en 1985 crea la Free Software Foundation, luego redacta la licencia GPL

Resumen GPL

- Cualquiera es libre de utilizar el Software Libre para cualquier propósito.
- Cualquiera es libre de acceder a su código fuente y estudiarlo.
- Cualquiera es libre de distribuirlo.
- Cualquiera es libre de mejorarlo o adaptarlo y de distribuir el programa modificado.
- La única obligación es que si se distribuye, haya que hacerlo bajo la misma licencia GPL.

Software libre

GNU/Linux

- Software libre vs. privativo
- Software libre vs. gratis
- RMS: en 1983 inicia oficialmente el proyecto GNU, en 1985 crea la Free Software Foundation, luego redacta la licencia GPL

Resumen GPL

- Cualquiera es libre de utilizar el Software Libre para cualquier propósito.
- Cualquiera es libre de acceder a su código fuente y estudiarlo.
- Cualquiera es libre de distribuirlo.
- Cualquiera es libre de mejorarlo o adaptarlo y de distribuir el programa modificado.
- La única obligación es que si se distribuye, haya que hacerlo bajo la misma licencia GPL.

- FOSS: Free and Open Source Software

Software libre

GNU/Linux

- Software libre vs. privativo
- Software libre vs. gratis
- RMS: en 1983 inicia oficialmente el proyecto GNU, en 1985 crea la Free Software Foundation, luego redacta la licencia GPL

Resumen GPL

- Cualquiera es libre de utilizar el Software Libre para cualquier propósito.
- Cualquiera es libre de acceder a su código fuente y estudiarlo.
- Cualquiera es libre de distribuirlo.
- Cualquiera es libre de mejorarlo o adaptarlo y de distribuir el programa modificado.
- La única obligación es que si se distribuye, haya que hacerlo bajo la misma licencia GPL.

- FOSS: Free and Open Source Software
- OSHW: Open Source Hardware

Software libre

Distribuciones – versiones/flavors

Software libre

Distribuciones – versiones/flavors

Linux Distribution Timeline

Software libre

Algunas aplicaciones

- Oficina: LibreOffice, LaTeX, LyX
- Edición imágenes, diagramas, etc.: Inkscape, GIMP, Blender
- Programación: GCC, GNU Compiler Collection
- IDEs de programación: Gedit, Geany, Code::Blocks, Eclipse CDT, Qt Creator, Vim, Emacs
- Diseño de circuitos (esquemático y PCB), simulación, etc.: KiCAD, QUCS
- Matemática: GNU Octave, Maxima (wxMaxima), Scilab, R, etc.
- Otros: Firefox, Iceweasel, Pidgin, Evince, Okular, VLC, Audacity, ...








- Display Manager: LightDM, GDM, KDM, LXDM, etc.
- Window Manager: Compiz, Metacity, Mutter, W9dk, fluxbox, etc.
- Desktop Environment: Gnome (GTK), KDE (Qt), LXDE (Lightweight X11 Desktop Environment), XFCE
- Graphical User Interface (GUI)

- Partición de disco (gparted). Dual-boot
  - MBR[1]: Master Boot Record (particiones primarias, extendidas y lógicas)
  - GPT[2]: GUID Partition Table (mas de 2TB y mayor # de particiones primarias)

[1] Origen en IBM-PC 1980s, MS-DOS
[2] UEFI, Unified Extensible Firmware Interface

- Partición de disco (gparted). Dual-boot
  - MBR[1]: Master Boot Record (particiones primarias, extendidas y lógicas)
  - GPT[2]: GUID Partition Table (mas de 2TB y mayor # de particiones primarias)
- Instalación CD, USB (bootable), red, etc.

[1] Origen en IBM-PC 1980s, MS-DOS
[2] UEFI, Unified Extensible Firmware Interface

- Partición de disco (gparted). Dual-boot
  - MBR[1]: Master Boot Record (particiones primarias, extendidas y lógicas)
  - GPT[2]: GUID Partition Table (mas de 2TB y mayor # de particiones primarias)
- Instalación CD, USB (bootable), red, etc.
- Sistema de gestión de paquetes (gestor de paquetes): colección de herramientas para automatizar el proceso de instalación, actualización, configuración y eliminación de paquetes de software.
  - Debian, `dpkg`: usa el formato `.deb`. Herramienta de resolución de dependencias `apt` (`apt-get update`, `apt-get install ...`)
  - Red Hat, `rpm`: RPM Package Manager
  - Suse, `YaST`: Yet another Setup Tool

[1] Origen en IBM-PC 1980s, MS-DOS
[2] UEFI, Unified Extensible Firmware Interface

- Partición de disco (gparted). Dual-boot
  - MBR[1]: Master Boot Record (particiones primarias, extendidas y lógicas)
  - GPT[2]: GUID Partition Table (mas de 2TB y mayor # de particiones primarias)
- Instalación CD, USB (bootable), red, etc.
- Sistema de gestión de paquetes (gestor de paquetes): colección de herramientas para automatizar el proceso de instalación, actualización, configuración y eliminación de paquetes de software.
  - Debian, `dpkg`: usa el formato `.deb`. Herramienta de resolución de dependencias `apt` (`apt-get update`, `apt-get install ...`)
  - Red Hat, `rpm`: RPM Package Manager
  - Suse, `YaST`: Yet another Setup Tool
- Repositorio (mirrors)

[1] Origen en IBM-PC 1980s, MS-DOS
[2] UEFI, Unified Extensible Firmware Interface

El shell de Linux

Usuarios, comandos y variables de entorno

Linux es un SO tipo Unix, multiplataforma, multitarea y multiusuario.

El shell de Linux

Usuarios, comandos y variables de entorno

Linux es un SO tipo Unix, multiplataforma, multitarea y multiusuario.

Usuario

- Para usar el SO es necesario abrir una sesión de trabajo (identificarse). Nro de usuario, UID. `whoami`
- Los usuarios se organizan en grupos. Nro de grupo, GID. `groups`. `id`
- Usuario **root** (superusuario o admin), cuenta con todos los privilegios

El shell de Linux

Usuarios, comandos y variables de entorno

Linux es un SO tipo Unix, multiplataforma, multitarea y multiusuario.

Usuario

- Para usar el SO es necesario abrir una sesión de trabajo (identificarse). Nro de usuario, UID. `whoami`
- Los usuarios se organizan en grupos. Nro de grupo, GID. `groups`. `id`
- Usuario **root** (superusuario o admin), cuenta con todos los privilegios

Comandos

- Archivo binario o ejecutable localizado en el sistema. Variable `PATH`. `which`
- Auto-completar `TAB`

El shell de Linux

Usuarios, comandos y variables de entorno

Linux es un SO tipo Unix, multiplataforma, multitarea y multiusuario.

Usuario

- Para usar el SO es necesario abrir una sesión de trabajo (identificarse). Nro de usuario, UID. `whoami`
- Los usuarios se organizan en grupos. Nro de grupo, GID. `groups`. `id`
- Usuario **root** (superusuario o admin), cuenta con todos los privilegios

Comandos

- Archivo binario o ejecutable localizado en el sistema. Variable `PATH`. `which`
- Auto-completar `TAB`

- **bash**: prompt. Aparece en la línea de comandos indicando que está a la espera de órdenes.
- Opciones de los comandos, letra seguido de '-'. O bien '--' (`--help`, `--version`)
- Comandos `echo`, `printenv`, `man`, `apropos`
- `Manpages`: manual en línea (RTFM). `man man`, `man 1 printf`, `man 3 printf`

- Entrada estándar, salida estándar, y salida estándar de error

- Entrada estándar, salida estándar, y salida estándar de error
- Se puede cambiar el flujo estándar de datos para ser redirigido a otro dispositivo, programa, archivo, etc.

- Entrada estándar, salida estándar, y salida estándar de error
- Se puede cambiar el flujo estándar de datos para ser redirigido a otro dispositivo, programa, archivo, etc.
- Algunos ejemplos:
  - `>`: salida estándar a un archivo o dispositivo (`$>data > /dev/null`)
  - `<`: entrada estándar a un archivo o dispositivo (`$>cat < file`)
  - `>>`: agrega salida estándar a archivo (no sobreescribe)
  - `|`: comunica dos procesos por medio de entrada salida

- Estructura de archivos en árbol. Archivos tipo directorio.

- Estructura de archivos en árbol. Archivos tipo directorio.
- Directorios (raíz/root), camino/path. Directorio actual (.), anterior o padre (..)

- Estructura de archivos en árbol. Archivos tipo directorio.
- Directorios (raíz/root), camino/path. Directorio actual (.), anterior o padre (..)
- Directorio `/home`. Variable de entorno `HOME`. Comando `pwd`

- Estructura de archivos en árbol. Archivos tipo directorio.
- Directorios (raíz/root), camino/path. Directorio actual (.), anterior o padre (..)
- Directorio `/home`. Variable de entorno `HOME`. Comando `pwd`
- Camino absoluto (comenzando en `/`), y camino relativo (comenzando en `./` o `../`)

- Estructura de archivos en árbol. Archivos tipo directorio.
- Directorios (raíz/root), camino/path. Directorio actual (.), anterior o padre (..)
- Directorio `/home`. Variable de entorno `HOME`. Comando `pwd`
- Camino absoluto (comenzando en `/`), y camino relativo (comenzando en `./` o `../`)
- Comandos `ls`, `touch`, `rm`, `cd`, `mkdir`, `rmdir`, `cp`, `mv`. (`$>which cd`)

- Estructura de archivos en árbol. Archivos tipo directorio.
- Directorios (raíz/root), camino/path. Directorio actual (.), anterior o padre (..)
- Directorio `/home`. Variable de entorno `HOME`. Comando `pwd`
- Camino absoluto (comenzando en `/`), y camino relativo (comenzando en `./` o `../`)
- Comandos `ls`, `touch`, `rm`, `cd`, `mkdir`, `rmdir`, `cp`, `mv`. (`$>which cd`)
- Atributos de archivos (`$>ls -l /`)

```
-rw-rw-r-- 1 gfpp gfpp 11 ago  4 15:25 hola.txt
^^^^^^^^^^ ^  ^    ^    ^   ^  ^ ^  ^   ^
|||||||||| |  |    |    |   |  | |  |   |_______ nombre del archivo
|||||||||| |  |    |    |   |  | |  |
|||||||||| |  |    |    |   |  | |  |___________ minutos     : Fecha y
|||||||||| |  |    |    |   |  | |______________ hora        : hora de la
|||||||||| |  |    |    |   |  |_________________ día del mes : última
|||||||||| |  |    |    |   |____________________ meas        : modificación
|||||||||| |  |    |    |
|||||||||| |  |    |    |___________________ Tamaño en bytes
|||||||||| |  |    |
|||||||||| |  |    |____________________ Nombre del grupo
|||||||||| |  |_________________________ Nombre del propietario
|||||||||| |
|||||||||| |____________________________ nro. de enlace rígido (hard link)
||||||||||
||||||||||_____________001__________ permiso de ejecución : Para
|||||||||_____________002__________ permiso de escritura : un usuario
||||||||_____________004__________ permiso de lectura   : cualquiera
|||||||
|||||||_____________010__________ permiso de ejecución : Para usuario
||||||_____________020__________ permiso de escritura : perteneciente
|||||_____________040__________ permiso de lectura   : al grupo
||||
||||____________________100__________  permiso de ejecución : Para usuario
|||____________________200__________  permiso de escritura : propietario
||____________________400__________  permiso de lectura   :
|
|_____________________________________ Tipo de archivo
```

Alterar permisos de archivos. Comando `chmod`

- `$>chmod u-r hola.txt`.

Alterar permisos de archivos. Comando `chmod`

- `$>chmod u-r hola.txt`.
- `$>cat hola.txt`

Alterar permisos de archivos. Comando `chmod`

- `$>chmod u-r hola.txt`.
- `$>cat hola.txt`
- `$>chmod o+x hola_bash.sh` (Crear script Shell)

Alterar permisos de archivos. Comando `chmod`

- `$>chmod u-r hola.txt`.
- `$>cat hola.txt`
- `$>chmod o+x hola_bash.sh` (Crear script Shell)

Notación numérica de los permisos

```
r w x | - w x | r - x          Equivale a un permiso 735
4 2 1 | 0 2 1 | 4 0 1          4+2+1, 0+2+1, 4+0+1 = 7,3,5
```

- En un sistema monotarea se utiliza generalmente el término programa.
- En un sistema multitarea se utiliza el término proceso, indicando que el está arrancado y en funcionamiento.
- Un programa puede dar lugar a varios procesos.
- Un proceso puede estar detenido (dormido), pero existe información del estado del mismo.
- Cada proceso tiene un nro que lo identifica, PID, PPID
- Comandos `pstree`, `ps`, `top`/`htop`, `kill` (señal 15 y 9)
- `$>pstree -p`, `$>ps axf`

- En un sistema monotarea se utiliza generalmente el término programa.
- En un sistema multitarea se utiliza el término proceso, indicando que el está arrancado y en funcionamiento.
- Un programa puede dar lugar a varios procesos.
- Un proceso puede estar detenido (dormido), pero existe información del estado del mismo.
- Cada proceso tiene un nro que lo identifica, PID, PPID
- Comandos `pstree`, `ps`, `top`/`htop`, `kill` (señal 15 y 9)
- `$>pstree -p`, `$>ps axf`

Atributos de los procesos

- `PID`: Valor numérico que identifica al proceso
- `TTY`: Terminal asociada al proceso
- `STAT`: Estado del proceso
- `TIME`: Tiempo de CPU consumido por el proceso
- `COMMAND`: Comandos y argumentos utilizados

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.
- Puede compilar C++

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.
- Puede compilar C++
- Realiza la optimización del código

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.
- Puede compilar C++
- Realiza la optimización del código
- Genera información de depuración (debugging)

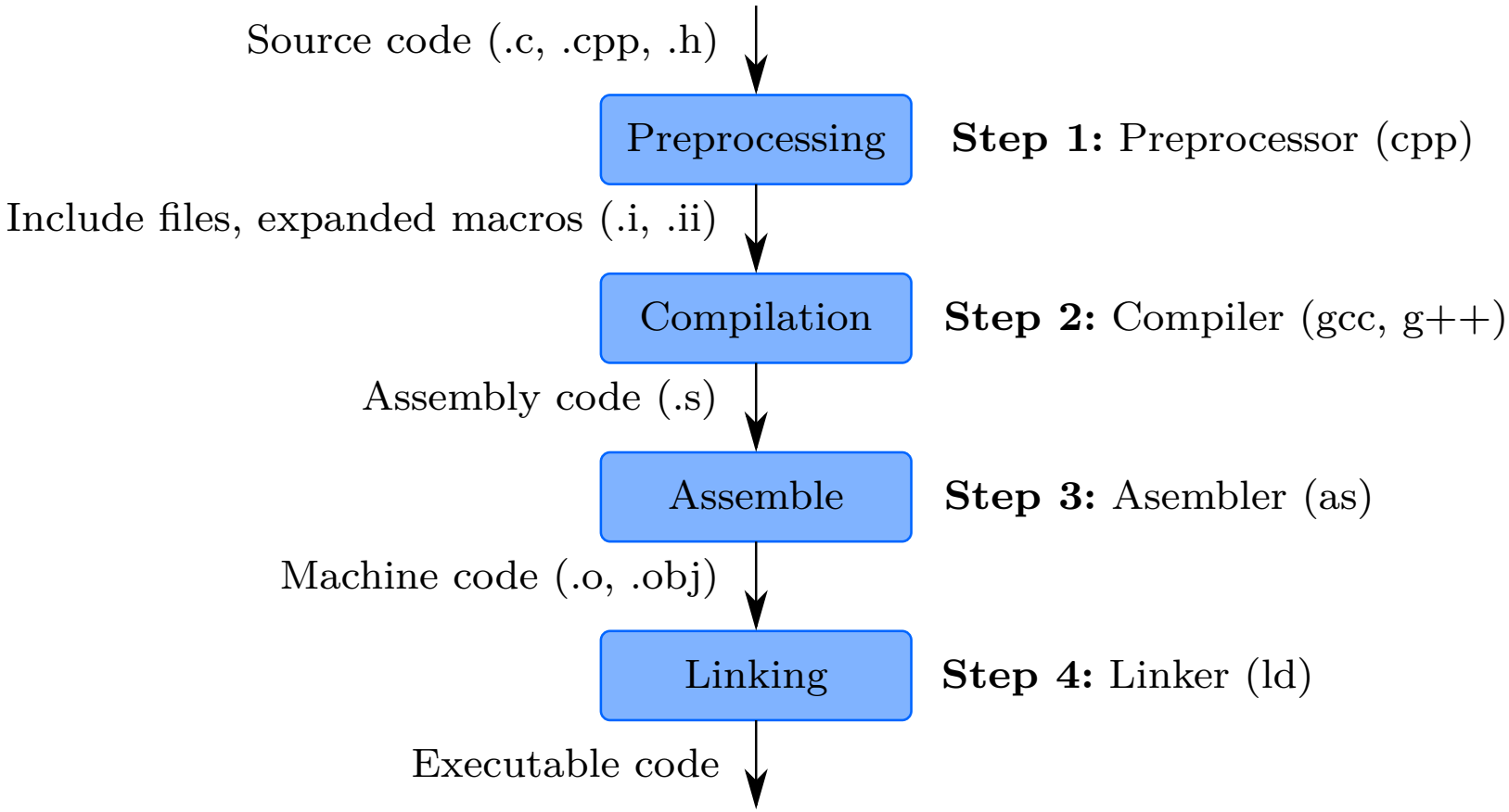
Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.
- Puede compilar C++
- Realiza la optimización del código
- Genera información de depuración (debugging)
- Es un compilador cruzado (cross-compiler)
  `$gcc --version`
  `$gcc -v`
  (`--build`, `--host`, `--target`)

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.
- Puede compilar C++
- Realiza la optimización del código
- Genera información de depuración (debugging)
- Es un compilador cruzado (cross-compiler)
  `$gcc --version`
  `$gcc -v`
  (`--build`, `--host`, `--target`)

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y linkeo)


Source code (.c, .cpp, .h)
Preprocessing
Step 1: Preprocessor (cpp)
Include files, expanded macros (.i, .ii)
Compilation
Step 2: Compiler (gcc, g++)
Assembly code (.s)
Assemble
Step 3: Asembler (as)
Machine code (.o, .obj)
Linking
Step 4: Linker (ld)
Executable code

```c
/* Source file: hello.c */
#include <stdio.h>

int main(void)
{
  printf("Hello, Linux programming world!\n");
  return 0;
}
```

Gestión de proyectos de software

Etapas de compilación

```c
/* Source file: hello.c */
#include <stdio.h>

int main(void)
{
  printf("Hello, Linux programming world!\n");
  return 0;
}
```

Compilar

```
$ gcc hello.c
```

Gestión de proyectos de software

Etapas de compilación

```c
/* Source file: hello.c */
#include <stdio.h>

int main(void)
{
  printf("Hello, Linux programming world!\n");
  return 0;
}
```

Compilar

```
$ gcc hello.c
```

```
$ gcc hello.c -o hello
```

```c
/* Source file: hello.c */
#include <stdio.h>

int main(void)
{
  printf("Hello, Linux programming world!\n");
  return 0;
}
```

Compilar

```
$ gcc hello.c
```

```
$ gcc hello.c -o hello
```

Ejecutar

```
$ ./hello
```

```c
/* Source file: hello.c */
#include <stdio.h>

int main(void)
{
  printf("Hello, Linux programming world!\n");
  return 0;
}
```

Compilar

```
$ gcc hello.c
```

```
$ gcc hello.c -o hello
```

Ejecutar

```
$ ./hello
```

Salida

```
$ Hello Linux programming world!
```

Preprocessing

```
$ gcc -E hello.c
```

Preprocessing

```
$ gcc -E hello.c
```

```
$ gcc -E hello.c -o hello.i
```

Preprocessing

```
$ gcc -E hello.c
```

```
$ gcc -E hello.c -o hello.i
```

```
$ cpp hello.c
```

Preprocessing

```
$ gcc -E hello.c
```

```
$ gcc -E hello.c -o hello.i
```

```
$ cpp hello.c
```

Copiar el archivo fuente y modificarlo

```c
/* Hello Linux */
#include <stdio.h>

#define STRING "Hello, Linux programming world!\n"

int main(void)
{
  /* Using a macro to print a message */
  printf(STRING);
  return 0;
}
```

Gestión de proyectos de software

Etapas de compilación, paso a paso

Preprocessing

```
$ gcc -E hello.c
```

```
$ gcc -E hello.c -o hello.i
```

```
$ cpp hello.c
```

Copiar el archivo fuente y modificarlo

```
1  /* Hello Linux */
2  #include <stdio.h>
3
4  #define STRING "Hello, Linux programming world!\n"
5
6  int main(void)
7  {
8    /* Using a macro to print a message */
9    printf(STRING);
10   return 0;
11 }
```

Observar: macros, comentarios, archivos cabecera (includes)

Compilar (continuando de la etapa anterior)

```
$ gcc -x cpp-output -c hello.i -o hello.o
```

La opción `-x` le indica que continúe desde la etapa indicada (man)

Compilar (continuando de la etapa anterior)

```
$ gcc -x cpp-output -c hello.i -o hello.o
```

La opción `-x` le indica que continúe desde la etapa indicada (man)

Linkeo

```
$ gcc hello.o -o hello
```

Compilar (continuando de la etapa anterior)

```
$ gcc -x cpp-output -c hello.i -o hello.o
```

La opción `-x` le indica que continúe desde la etapa indicada (man)

Linkeo

```
$ gcc hello.o -o hello
```

Flag `-save-temps` genera archivos intermedios

```
$ gcc -save-temps hello.c -o hello
```

Compilar (continuando de la etapa anterior)

```
$ gcc -x cpp-output -c hello.i -o hello.o
```

La opción `-x` le indica que continúe desde la etapa indicada (man)

Linkeo

```
$ gcc hello.o -o hello
```

Flag `-save-temps` genera archivos intermedios

```
$ gcc -save-temps hello.c -o hello
```

(Flag `-Wall`)

Ejemplo de varios archivos fuentes

- Editar `juego.c`
  - Construir el binario `juego`
  - Compilar (unicamente) el código fuente
  - Analizar errores de compilación, linkeo, y warnings

Ejemplo de varios archivos fuentes

- Editar `juego.c`
  - Construir el binario `juego`
  - Compilar (unicamente) el código fuente
  - Analizar errores de compilación, linkeo, y warnings
- Editar `tirador.c`
  - Construir el binario
  - Analizar errores y warnings

Ejemplo de varios archivos fuentes

- Editar `juego.c`
  - Construir el binario `juego`
  - Compilar (unicamente) el código fuente
  - Analizar errores de compilación, linkeo, y warnings
- Editar `tirador.c`
  - Construir el binario
  - Analizar errores y warnings
- Corregir errores y warnings

Ejemplo de varios archivos fuentes

- Editar `juego.c`
  - Construir el binario `juego`
  - Compilar (unicamente) el código fuente
  - Analizar errores de compilación, linkeo, y warnings
- Editar `tirador.c`
  - Construir el binario
  - Analizar errores y warnings
- Corregir errores y warnings
- Directorio de archivos cabecera (opción `-I`)

Bibliotecas estáticas:

- El código se incluye en el binario/ejecutable

Bibliotecas estáticas:

- El código se incluye en el binario/ejecutable
- El sufijo (extensión) es `.a` (`archive`)

Bibliotecas estáticas:

- El código se incluye en el binario/ejecutable
- El sufijo (extensión) es `.a` (`archive`)
- Contiene un conjunto de archivos objeto

Bibliotecas estáticas:

- El código se incluye en el binario/ejecutable
- El sufijo (extensión) es `.a` (`archive`)
- Contiene un conjunto de archivos objeto
- Construcción: código fuente → compilar código objeto → crear biblioteca

Bibliotecas estáticas:

- El código se incluye en el binario/ejecutable
- El sufijo (extensión) es `.a` (`archive`)
- Contiene un conjunto de archivos objeto
- Construcción: código fuente → compilar código objeto → crear biblioteca

Bibliotecas compartidas:

- El código no se incluye en el binario

Bibliotecas estáticas:

- El código se incluye en el binario/ejecutable
- El sufijo (extensión) es `.a` (`archive`)
- Contiene un conjunto de archivos objeto
- Construcción: código fuente → compilar código objeto → crear biblioteca

Bibliotecas compartidas:

- El código no se incluye en el binario
- El sufijo (extensión) es `.so` (`share object`)

Bibliotecas estáticas:

- El código se incluye en el binario/ejecutable
- El sufijo (extensión) es `.a` (`archive`)
- Contiene un conjunto de archivos objeto
- Construcción: código fuente → compilar código objeto → crear biblioteca

Bibliotecas compartidas:

- El código no se incluye en el binario
- El sufijo (extensión) es `.so` (`share object`)
- Enlaces (soft-link), y variable de entorno `LD_LIBRARY_PATH`

Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

Herramienta `make`

Herramienta para la construcción (re-construcción) de software.

- `make` simplifica el proceso de construcción de proyectos de muchos archivos fuentes, que generalmente requieren varias llamadas al compilador
- Automatiza: qué partes construir, cómo construirlas, y cuando.
- Le permite al programador poder concentrarse en el código
- `make` minimiza el tiempo de construcción (determina qué archivos cambiaron), además trabaja con dependencias

Herramienta make

Ejemplo de múltiples archivos fuentes

```c
/* main.c */
#include "a.h"


. . .
```

```c
/* 2.c */
#include "a.h"
#include "b.h"

. . .
```

```c
/* 3.c */
#include "b.h"
#include "c.h"

. . .
```

Herramienta `make`

Ejemplo de múltiples archivos fuentes

```c
/* main.c */
#include "a.h"


. . .
```

```c
/* 2.c */
#include "a.h"
#include "b.h"

. . .
```

```c
/* 3.c */
#include "b.h"
#include "c.h"

. . .
```

- Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados

Herramienta `make`

Ejemplo de múltiples archivos fuentes

```c
/* main.c */
#include "a.h"


. . .
```

```c
/* 2.c */
#include "a.h"
#include "b.h"

. . .
```

```c
/* 3.c */
#include "b.h"
#include "c.h"

. . .
```

- Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- El archivo `3.c` depende del archivo `c.h`

Herramienta `make`

Ejemplo de múltiples archivos fuentes

```c
/* main.c */
#include "a.h"


. . .
```

```c
/* 2.c */
#include "a.h"
#include "b.h"

. . .
```

```c
/* 3.c */
#include "b.h"
#include "c.h"

. . .
```

- Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- El archivo `3.c` depende del archivo `c.h`
- Qué pasa si se modifica `b.h` y no se recompila `2.c`

Herramienta make

Ejemplo de múltiples archivos fuentes

```c
/* main.c */
#include "a.h"


. . .
```

```c
/* 2.c */
#include "a.h"
#include "b.h"

. . .
```

```c
/* 3.c */
#include "b.h"
#include "c.h"

. . .
```

- Si se modifica `c.h`, los archivos `main.c` y `2.c` no necesitan ser recompilados
- El archivo `3.c` depende del archivo `c.h`
- Qué pasa si se modifica `b.h` y no se recompila `2.c`

Dependencias:

```
myapp: main.o 2.o 3.0
main.o: main.c a.h
2.o: 2.c a.h b.h
3.o: 3.c b.h c.h
```

Un archivo **`Makefile`** es un archivo de texto que contiene *reglas* que le indican a **`make`** qué construir y cómo. Una *regla* consiste en:

Un archivo **`Makefile`** es un archivo de texto que contiene *reglas* que le indican a **`make`** qué construir y cómo. Una *regla* consiste en:

- Un *target* (objetivo): lo que se debe construir
- Una lista de una o más *dependencias*: archivos necesarios para construir el *target*
- Una lista de *comandos* a ejecutar para construir el objetivo

Un archivo **Makefile** es un archivo de texto que contiene *reglas* que le indican a **make** qué construir y cómo. Una *regla* consiste en:

- Un *target* (objetivo): lo que se debe construir
- Una lista de una o más *dependencias*: archivos necesarios para construir el *target*
- Una lista de *comandos* a ejecutar para construir el objetivo

```
target: dependency dependency [...]
        command
        command
        [...]
```

Un archivo **Makefile** es un archivo de texto que contiene *reglas* que le indican a **make** qué construir y cómo. Una *regla* consiste en:

- Un *target* (objetivo): lo que se debe construir
- Una lista de una o más *dependencias*: archivos necesarios para construir el *target*
- Una lista de *comandos* a ejecutar para construir el objetivo

```
target: dependency dependency [...]
        command
        command
        [...]
```

Cuando se ejecuta, **make** busca los archivos **GNUmakefile**, **makefile**, y **Makefile**, en ese orden.

Herramienta make

Ejemplo de archivo Makefile

```
1  editor : editor.o screen.o keyboard.o
2           gcc -o editor editor.o screen.o keyboard.o
3
4  editor.o : editor.c editor.h keyboard.h screen.h
5           gcc -c editor.c
6
7  screen.o : screen.c screen.h
8           gcc -c screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11          gcc -c keyboard.c
12
13 clean :
14          rm editor *.o
```

Herramienta make

Ejemplo de archivo Makefile

```
1  editor : editor.o screen.o keyboard.o
2           gcc -o editor editor.o screen.o keyboard.o
3
4  editor.o : editor.c editor.h keyboard.h screen.h
5           gcc -c editor.c
6
7  screen.o : screen.c screen.h
8           gcc -c screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11          gcc -c keyboard.c
12
13 clean :
14          rm editor *.o
```

Construir/compilar el proyecto **editor**

```
$ make
```

Herramienta make

Ejemplo de archivo Makefile

```
1  editor : editor.o screen.o keyboard.o
2           gcc -o editor editor.o screen.o keyboard.o
3
4  editor.o : editor.c editor.h keyboard.h screen.h
5           gcc -c editor.c
6
7  screen.o : screen.c screen.h
8           gcc -c screen.c
9
10 keyboard.o : keyboard.c keyboard.h
11          gcc -c keyboard.c
12
13 clean :
14          rm editor *.o
```

Construir/compilar el proyecto **editor**

```
$ make
```

(**$ make clean**)

Herramienta make

Probando make

```
1  myapp: main.o 2.o 3.o
2          gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5          gcc -c main.c
6
7  2.o: 2.c a.h b.h
8          gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11         gcc -c 3.c
```

Herramienta make

Probando make

```
1  myapp: main.o 2.o 3.o
2          gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5          gcc -c main.c
6
7  2.o: 2.c a.h b.h
8          gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11         gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error

Herramienta make

Probando make

```
1  myapp: main.o 2.o 3.o
2          gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5          gcc -c main.c
6
7  2.o: 2.c a.h b.h
8          gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11         gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`

Herramienta make

Probando make

```
1  myapp: main.o 2.o 3.o
2          gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5          gcc -c main.c
6
7  2.o: 2.c a.h b.h
8          gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11         gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`
- Editar `main.c`, `2.c` y `3.c`

Herramienta make

Probando make

```
1  myapp: main.o 2.o 3.o
2          gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5          gcc -c main.c
6
7  2.o: 2.c a.h b.h
8          gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11         gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`
- Editar `main.c`, `2.c` y `3.c`
- `$ make -f Makefile1`

Herramienta make

Probando make

```
1  myapp: main.o 2.o 3.o
2          gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5          gcc -c main.c
6
7  2.o: 2.c a.h b.h
8          gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11         gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`
- Editar `main.c`, `2.c` y `3.c`
- `$ make -f Makefile1`
- Renombrar archivo makefile

Herramienta make

Probando make

```
1  myapp: main.o 2.o 3.o
2          gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5          gcc -c main.c
6
7  2.o: 2.c a.h b.h
8          gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11         gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`
- Editar `main.c`, `2.c` y `3.c`
- `$ make -f Makefile1`
- Renombrar archivo makefile
- `$ touch b.h`

Herramienta make

Probando make

```
1  myapp: main.o 2.o 3.o
2          gcc -o myapp main.o 2.o 3.o
3
4  main.o: main.c a.h
5          gcc -c main.c
6
7  2.o: 2.c a.h b.h
8          gcc -c 2.c
9
10 3.o: 3.c b.h c.h
11         gcc -c 3.c
```

- `$ make -f Makefile1`, y analizar error
- Crear archivos header. `$ touch a.h`, `$ touch b.h`, `$ touch c.h`
- Editar `main.c`, `2.c` y `3.c`
- `$ make -f Makefile1`
- Renombrar archivo makefile
- `$ touch b.h`
- Eliminar el archivo `2.o`, y probar nuevamente

Herramienta make

Probando make. Comentarios y macros.

```
1  all: myapp
2
3  # Which compiler
4  CC = gcc
5
6  # Where are include file kept
7  INCLUDE = .
8
9  # Options for development
10 CFLAGS = -g -Wall -ansi
11
12 # Options for release
13 # CFLAGS = -O -Wall-ansi
14
15 myapp: main.o 2.o 3.o
16         $(CC) -o myapp main.o 2.o 3.o
17
18 main.o: main.c a.h
19         $(CC) -I$(INCLUDE) $(CFLAGS) -c main.c
20
21 2.o: 2.c a.h b.h
22         $(CC) -I$(INCLUDE) $(CFLAGS) -c 2.c
23
24 3.o: 3.c b.h c.h
25         $(CC) -I$(INCLUDE) $(CFLAGS) -c 3.c
```

($ rm myapp *.o, y $ make -f Makefile2)

Bibliografía

- *Linux Programming Unleashed*, 2nd Edition, SAMS (1999)
- *Beginning Linux Programming*, 4th Edition, Wiley (2007)
- *IEEE Spectrum*, abril de 2016
- *Curso de Linux para novatos, brutos y extremadamente torpes*, Ed. nov. 2008
- Manuales y tutoriales de la web