

Introduction to Unix and GNU / Linux

Introduction to Unix and GNU / Linux

Michael Opdenacker

Free Electrons

<http://free-electrons.com>



Thanks

- To the OpenOffice.org project, for their presentation and word processor tools which satisfied all my needs.
- To the Handhelds.org community, for giving me so much help and so many opportunities to help.
- To the members of the whole Free Software and Open Source community, for sharing the best of themselves: their work, their knowledge, their friendship.
- To people who sent comments and corrections:
Jeff Ghislain



Copying this document

© 2004, Michael Opdenacker
michael@free-electrons.com

This document is released under the GNU Free Documentation License, with no invariant sections.

Permission is granted to copy and modify this document provided this license is kept.

See <http://www.gnu.org/licenses/fdl.html> for details

Document updates available

on http://free-electrons.com/training/intro_unix_linux

Corrections, suggestions and contributions are welcome!



Document history

Unless specified, contributions are from Michael Opdenacker

- Sep 28, 2004. First public release
- Sep 20-24, 2004. First session for [Atmel](#), Rousset (France)



About this document

- This document is first of all meant to be used as visual aids by a speaker or a trainer. Hence, this is just a summary or a complement to what is said. Hence, the explanations are not supposed to be exhaustive.
- However, this document is also meant to become a reference for the audience. It also targets readers interested in self-training. So, a bit more details are given, making the document a bit less visually attractive.



Training contents (1)

Introduction

- History of Unix
- Unix philosophy and features
- The various layers in a Unix system
- The GNU project, the GPL license
- Linux, Distributions
- Other free Unix systems



Training Contents (2)

Shells, filesystem and file handling

- Command line interpreters
- Unix filesystem structure
- Handling files and directories
- Displaying, scanning and sorting files
- Symbolic and hard links
- File access rights



Training contents (3)

Standard I/O, redirections, pipes

- Standard input and output
- Redirecting standard input or output to files
- Pipes: redirecting standard output to other commands
- Standard error



Training contents (4)

Task control

- Unix: multitask since the beginning
- Executing in background, suspending, resuming and aborting
- List of active tasks
- Aborting 1 or several tasks
- Environment variables
- The PATH environment variable
- Shell aliases, `.bashrc` file



Training contents (5)

Misc

- Text editors
- Compression and archiving
- Printing files
- Comparing files
- Looking for files
- Getting information about users
- Quick overview of a few desktop applications (e-mail, web browsing, word processor)
- Unix - Windows equivalent programs



Training contents (6)

Going further

- Getting help, accessing manual pages
- Searching the Internet for resources
- Using GNU / Linux at home (supplied cdrom)

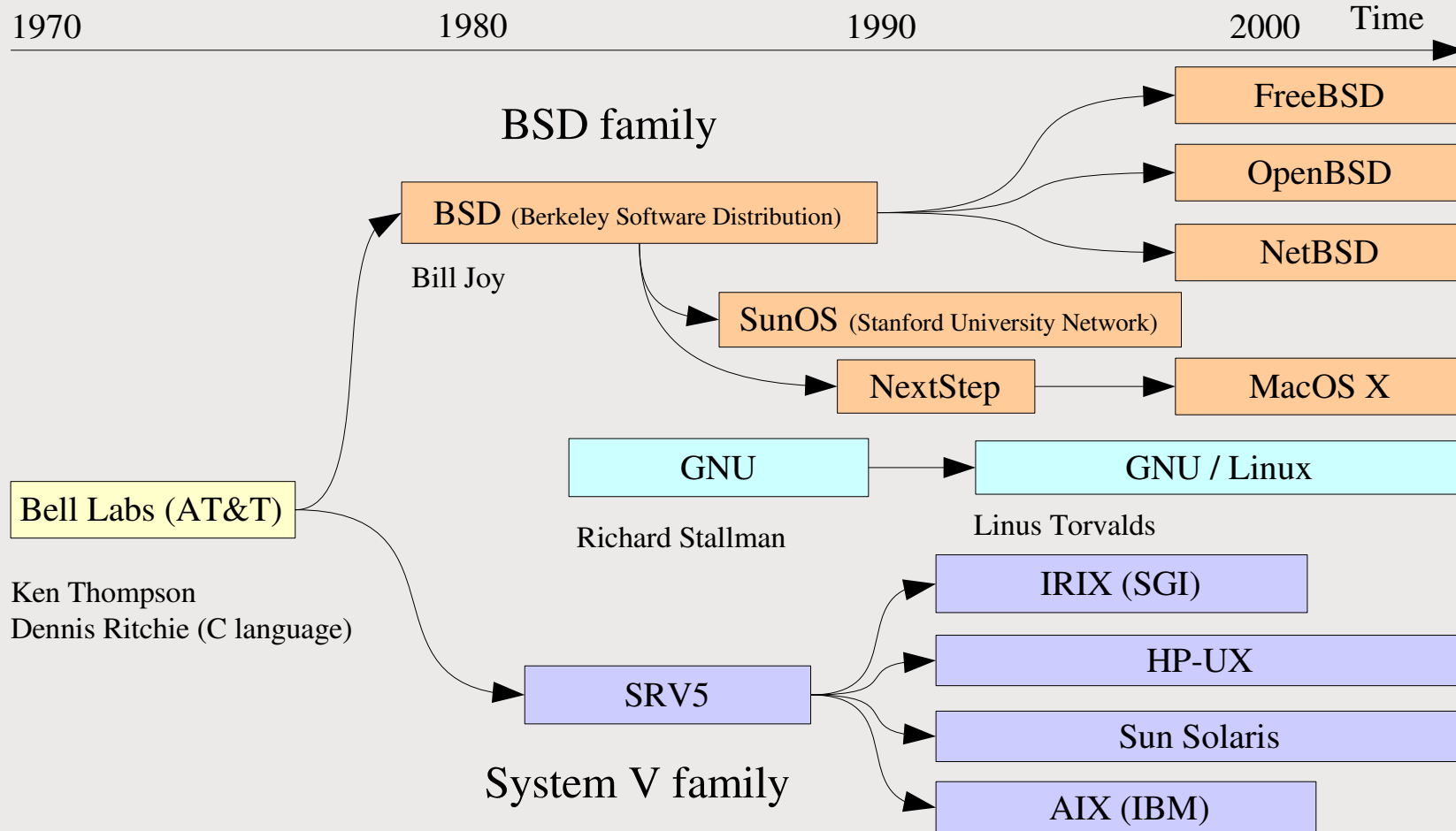


Introduction to Unix and GNU / Linux

Introduction



History of Unix



The Unix philosophy

- Small is beautiful
- Make each program do one thing well
- Choose portability over efficiency
- Avoid captive user interfaces
- System abstraction
 - Kernel: hardware layer
 - Shell: text mode layer
 - X Windows: GUI layer



Main Unix features

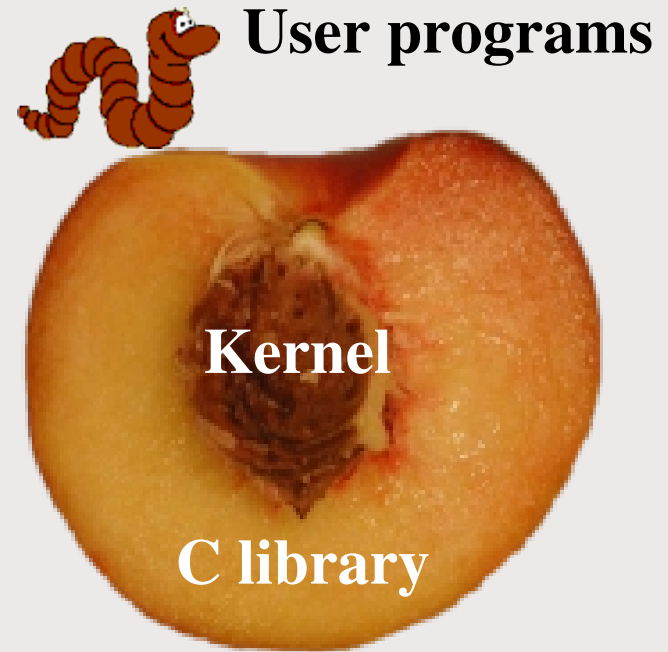
Since the beginning in the seventies!

- Multi-user and secure:
Regular users can't mess with other user's files (by default)
In particular, regular users can't modify system settings, can't remove programs, etc.
“root”: administrator user with all privileges
- Multi-task
- Supports multiple processors
- Extremely flexible
- Networking support



Layers in a Unix system

- Kernel
- C library
- System libraries
- Application libraries
- User programs



The GNU Project

GNU = GNU is Not Unix

- Project to implement a completely free Unix-like operating system
- Started by Richard Stallman in 1984, an MIT researcher, in a time when Unix sources were no longer free.
- Initial components: C compiler (gcc), make (GNU make), Emacs, C library (glibc), coreutils (ls, cp ...)
- However, in 1991, the GNU project was still missing a kernel and was running only on proprietary unice.



About Free Software

- GNU, Linux and many other programs are *Free Software*
- *Free Software* grants the below 4 freedoms to the user:
 - The freedom to run the program, for any purpose
 - The freedom to study how the program works, and adapt it to one's needs
 - The freedom to redistribute copies to help others
 - The freedom to improve the program, and release one's improvements to the public
- See <http://www.gnu.org/philosophy/free-sw.html>



The GNU General Public License (GPL)

- *Copyleft* licenses use copyright laws to make sure that modified versions are free software too
- The GNU GPL requires that modifications and derived works are GPL too:
 - Only applies to **released** software
 - Any program using GPLed code (either by static or even dynamic linking) is considered as an extension of this code
- More details:
 - Copyleft: <http://www.gnu.org/copyleft/copyleft.html>
 - GPL FAQ: <http://www.gnu.org/licenses/gpl-faq.html>



Linux

- Free Unix-like kernel created in 1991 by Linus Torvalds
- The whole system uses GNU tools:
C library, gcc, binutils, fileutils, make, emacs...
- So the whole system is called “GNU / Linux”
- Shared very early as free software (GPL license), which attracted more and more contributors and users.
- Since 1991, growing faster than any other operating system (not only Unix).



GNU / Linux distributions

- Take care of releasing a compatible set of kernel, C libraries, compilers and tools... A lot of work indeed!
- Tools available in *packages* which can be easily installed, removed or upgraded. Tool version dependencies are automatically managed.
- Commercial distributions: include support. Sources are free but not binaries.
- Community distributions: both sources and binaries are free. No support by default.
- Name and version confusion
 - Don't say "Linux 9.0". Say "Red Hat 9.0" or "Red Hat GNU / Linux" instead.



Commercial distributions

- Red Hat: <http://www.redhat.com/>
The most popular. Reliable, safe, user friendly, easy to install, supported by all hw and sw vendors.
- Suse (Novell): <http://www.suse.com/>
The main alternative. Easy to install, user friendly. Can't tell about stability. Not supported yet by all vendors.
- Mandrake: <http://www.mandrakelinux.com/>
User friendly, easy to install, more innovative, but less stable (perhaps better since the introduction of community pre-releases). More targeted to individual users. Little vendor support.



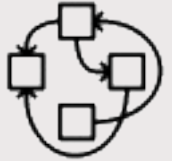
Community distributions

- Debian: <http://debian.org/>
Very stable and safe, but more difficult to configure and install. Developer but no user friendly yet. Stable releases not frequent enough (every 2 or 3 years). The best for servers, but not for beginners!
- Fedora Core: <http://fedora.redhat.com/>
Stable, secure, user friendly, easy to install. Frequent full releases.
- Mandrake Community: <http://www.mandrakelinux.com/>
Easy to install, secure, user friendly, frequent full releases, but less stable (not enough testing and taking user feedback into account).



Other Free Unix systems

- GNU / Hurd: <http://www.gnu.org/software/hurd/hurd.html>
 - GNU tools with the Hurd, the GNU kernel (microkernel)
 - Getting mature, but not enough yet for general use. Only used by Hurd developers so far (2004).
- BSD Family
 - FreeBSD: <http://www.freebsd.org/>
Powerful, multiplatform, secure, and popular BSD system
 - OpenBSD: <http://openbsd.org/>
Built for extreme security and reliability. Popular in Internet servers.
 - NetBSD: <http://netbsd.org/>
BSD distribution which goal is to be extremely portable (available on ARM, for example)



Introduction to Unix and GNU / Linux

Shell, filesystem and file handling



Command line interpreters

- Shells: tools to execute user commands
- Called “shells” because they hide the details on the underlying operating system under the shell's surface.
- Commands are input in a text terminal, either a window in a graphical environment or a text-only console.
- Results are also displayed on the terminal. No graphics are needed at all.
- Shells can be scripted: provide all the resources to write complex programs (variable, conditionals, iterations...)



Well known shells

Most famous and popular shells

- sh: The Bourne shell (obsolete) (find Bourne name and Traditional, basic shell found on Unix systems)
- csh: The C shell (obsolete)
Once popular shell with a C-like syntax
- tcsh: The TC shell (still very popular)
A C shell compatible implementation with evolved features (command completion, history editing and more...)
- bash: The Bourne Again shell (most popular)
An improved implementation of sh with lots of added features too.



Files and directories

Almost everything in Unix / Linux is a file

- Even directories are files: contain a list of files and directories
- Since the beginning of Unix, no obvious limitation in file name length. Any character (whitespace in particular) can be used in file names, an extensions are optional. File names are case sensitive.
- A *path* is a sequence of nested directories with a file or directory at the end, separated by the / character
 - Relative path: `documents/fun/microsoft_jokes.html`
Relative to the current directory
 - Absolute path: `/home/bill/bugs/crash9402031614568`
Path from the root system directory (/)



GNU / Linux filesystem structure (1)

Not imposed by the system. Can vary from one system to the other, even between two GNU/Linux installations!

/	Root directory
/bin/	Basic, essential system commands
/boot/	Kernel images, initrd and configuration files
/dev/	Files representing devices /dev/hda: first IDE hard disk
/etc/	System configuration files
/home/	User directories
/lib/	Basic system shared libraries



GNU / Linux filesystem structure (2)

<code>/lost+found</code>	Corrupt files the system tried to recover
<code>/mnt/</code>	Mounted filesystems <code>/mnt/usbdisk/</code> , <code>/mnt/windows/</code> ...
<code>/opt/</code>	Specific tools installed by the sysadmin <code>/usr/local/</code> often used instead
<code>/proc/</code>	Access to system information <code>/proc/cpuinfo</code> , <code>proc/version</code> ...
<code>/root/</code>	root user home directory
<code>/bin/</code>	Administrator-only commands
<code>/sys/</code>	System / device vice controls (cpu frequency, etc.)



GNU / Linux filesystem structure (3)

<code>/tmp/</code>	Temporary files
<code>/usr/</code>	Regular user tools (not essential to the system) <code>/usr/bin/</code> , <code>/usr/lib/</code> , <code>/usr/sbin...</code>
<code>/usr/local/</code>	Specific software installed by the sysadmin (often preferred to <code>/opt/</code>)
<code>/var/</code>	Data used by the system or system servers <code>/var/log/</code> , <code>/var/spool/mail</code> (incoming mail), <code>/var/spool/lpd</code> (print jobs)...



ls command

Lists the files in the current directory, in alphanumeric order, except files starting with the “.” character.

- `ls -a` (all)
Lists all the files (including .* files)
- `ls -l` (long)
Long listing (type, date, size, owner, permissions)
- `ls -t` (time)
Lists the most recent files first
- `ls -r` (reverse)
Reverses the sort order
- `ls -ltr` (options can be combined)
Long listing, most recent files at the end



File name pattern substitutions

Better introduced by examples!

- `ls a*.txt`

The shell first replaces `a*.txt` by all the file names starting by `a` and ending by `.txt` (except files starting with `.`), and then executes the `ls` command line.

- `ls .*`

Lists all the files starting with `.`

- `cat ?.log`

Displays all the files which names start by 1 character and end by `.log`



Special directories (1)

./

- The current directory. Useful for commands taking a directory argument. Also sometimes useful to run commands in the current directory (see later)
- So `./readme.txt` and `readme.txt` are equivalent

../

- The parent directory. Belongs to the directory contents (see `ls -a`) as the only reference to the parent directory.
- Typical usage:
`cd ..`



Special directories (2)

~/

- Not a special directory indeed. Shells just substitute it by the home directory of the current user.
- Cannot be used in most programs, as it is not a real directory.

~sidney/

- Similarly, substituted by shells by the home directory of the `sidney` user.



cd and cp commands

- `cd <dir>`
Change the current directory to <dir>
- `cp <source_file> <target_file>`
Copies the source file to the target
- `cp file1 file2 file3 ... dir`
Copies the files to the target directory (last argument)
- `cp -i` (interactive)
Asks for user confirm if the target file already exists
- `cp -r <source_dir> <target_dir>` (recursive)
Copies directories recursively



Smart directory copy with rsync

- rsync (remote sync) has been designed to keep in sync directories on machines with a low bandwidth connection.
- rsync features
 - Only copies files that have changed. Files with the same size are compared by checksums.
 - Only transfers the blocks that differ within a file!
 - Can compress the transferred blocks
 - Preserves symbolic links and file permissions: also very useful for local copies.
 - Can work through ssh (secure remote shell). Very useful to update the contents of a website, for example.



rsync examples (1)

- `rsync -a /home/arvin/sd6_agents/ /home/sidney/misc/`
 - a: archive mode. Equivalent to `-r1ptgoD...` easy way to tell you want recursion and want to preserve almost everything.
 - `rsync -Pav --delete /home/steve/ideas/ /home/bill/my_ideas/`
 - P: `--partial` (keep partially transferred files) and `--progress` (show progress during transfer)
 - `--delete`: delete files in the target which don't exist in the source.
- Caution: directory names have to end with `/`. Otherwise, you get a `my_ideas/ideas/` directory at the destination.



rsync examples (2)

- Copying to a remote machine

```
rsync -Pav /home/bill/legal/arguments/ \  
bill@www.sco.com:/home/legal/arguments/
```

User `bill` will be prompted for a password

- Copying from a remote machine through ssh

```
rsync -Pav -e ssh  
homer@tank.duff.com/prod/beer/ \  
fridge/homer/beer/
```

User `homer` will be prompted for his ssh key password



mv and rm commands

- `mv <old_name> <new_name>`
Renames the given file or directory
- `mv -i` (interactive)
If the new file already exists, asks for user confirm
- `rm file1 file2 file3 ...`
Removes the given files
- `rm -i` (interactive)
Always ask for user confirm
- `rm -r dir` (recursive)
Recursively removes the given files or directories



Creating and removing directories

- `mkdir dir1 dir2 dir3 ...`
Creates directories with the given names
- `rmdir dir dir2 dir3 ...`
Removes the given directories
Safe: only works when directories are empty
Alternative: `rm -r`



Displaying files

Several ways of displaying a file

- `cat file1 file2 file3 ...`

Outputs the contents of the given files

- `more file1 file2 file3 ...`

Asks the user to continue every time the screen is full
Also can jump to the first occurrence of a keyword

- `less file1 file2 file3 ...`

Does more than more with less

Doesn't read the whole file before starting

Supports backward movement in the file



The head and tail commands

- `head [-<n>] <file>`
Displays the first <n> lines (or 10 by default) of the given file.
Doesn't have to open the whole file to do this!
- `tail [-<n>] <file>`
Displays the last <n> lines (or 10 by default) of the given file.
No need to load the whole file in RAM! Very useful for huge files.
- `tail -f <file>`
(f: follow). Displays the last 10 lines of the given file and continues to display new lines when they are appended to the file.
Very useful to follow the changes in a log file, for example.
- Examples
`head windows_bugs.txt`
`tail -f outlook_vulnerabilities.txt`



The grep command

- `grep <pattern> <files>`
Scans the given files and displays the lines which match the given pattern.
- `grep error *.log`
Displays all the lines containing error in the *.log files
- `grep -i error *.log`
Same, but case insensitive
- `grep -ri error .`
Same, but recursively in all the files in . and its subdirectories
- `grep -v info *.log`
Outputs all the lines in the files except those containing info



The sort command

- `sort <file>`
Sorts the lines in the given file in character order and outputs them
- `sort -r <file>`
Same, but in reverse order
- `sort -ru <file>`
u: unique. Same, but just outputs identical lines once.
- More possibilities described later!



Symbolic links

A symbolic link is a special file which is just a reference to the name of another one (file or directory):

- Useful to reduce disk usage and complexity when 2 files have the same content
- Example:
`anakin_skywalker_biography -> darth_vador_biography`
- How to identify symbolic links:
 - `ls -l` displays `->` and the linked file name
 - GNU `ls` displays links with a different color



Creating symbolic links

- To create a symbolic link (same order as in `cp`):
`ln -s file_name link_name`
- To create a link with to a file in another directory, with the same name:
`ln -s ../README.txt`
- To create multiple links at once in a given directory:
`ln -s file1 file2 file3 ... dir`
- To remove a link:
`rm link_name`
Of course, this doesn't remove the linked file!



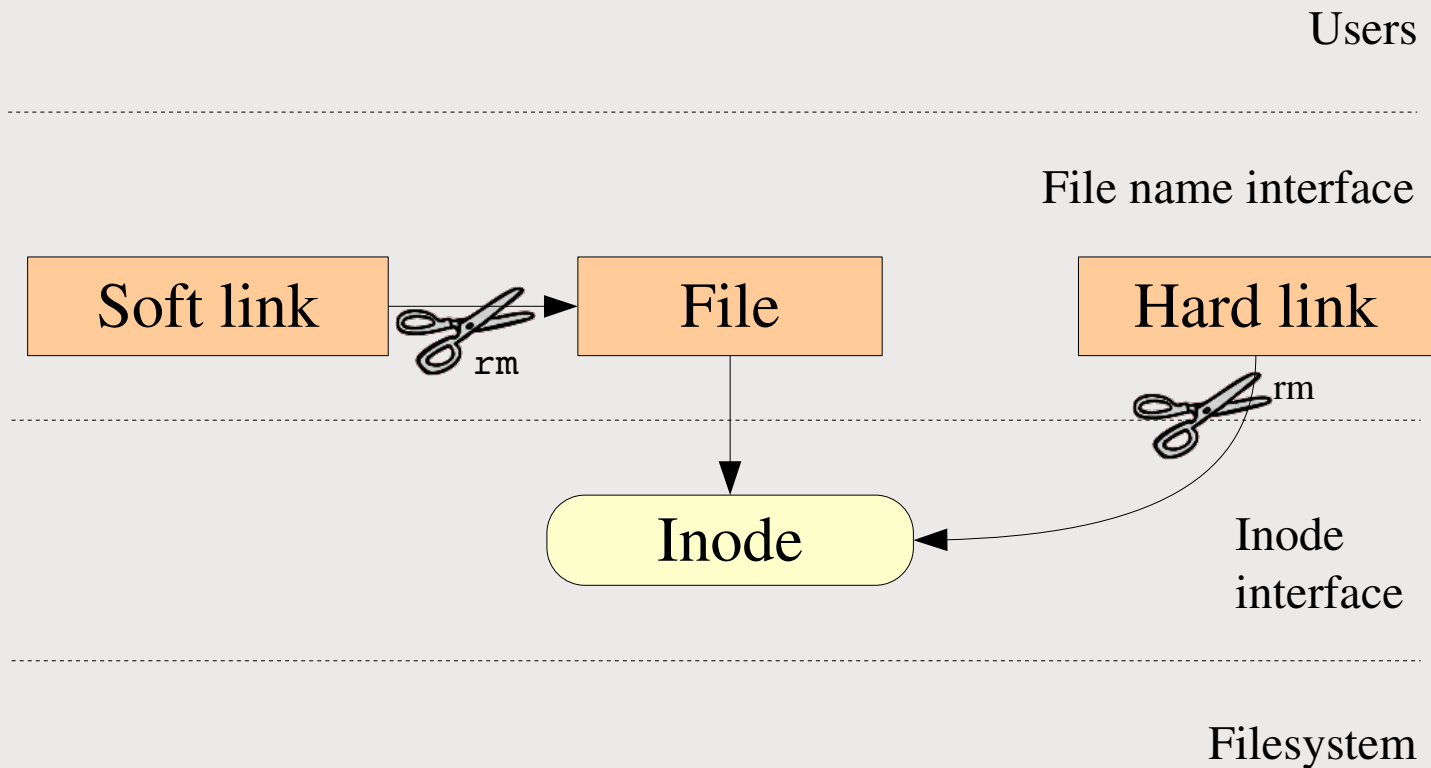
Hard links

- The default behavior for `ln` is to create *hard links*
- A *hard link* to a file is a regular file with exactly the same physical contents
- While they still save space, hard links can't be distinguished from the original files.
- If you remove the original file, the hard link contents are not impacted!
- The contents are removed when there are no more files (hard links) to them.



Files names and inodes

Makes hard and symbolic (soft) links easier to understand!



File access rights

- Use `ls -l` to check file access rights
- 3 types of access rights:
 - Read access (`r`)
 - Write access (`w`)
 - Execute rights (`x`)
- 3 types of access levels:
 - User (`u`): for the owner of the file
 - Group (`g`): each file also has a “group” attribute, corresponding to a given list of users
 - Others (`o`): for all other users



Access right constraints

- x without r is legal but is useless
You have to be able to read a file to execute it
- Both r and x permissions needed for directories. x to enter, r to list its contents.
- You can't rename, remove, copy files in a directory if you don't have w access to this directory.
- If you have w access to a directory, you CAN remove a file even if you don't have write access to this file
(Remember that a directory is just a file describing a list of files).
This even lets you modify (remove + recreate) a file even without w access to it.



Access rights examples

- `-rw-r--r--`

Readable and writable for file owner, only readable for others

- `-rw-r-----`

Readable and writable for file owner, only readable for users belonging to the file group.

- `drwx-----`

Directory only accessible by its owner

- `-----r-x`

File executable by others but neither by your friends nor by yourself. Nice protections for a trap...



chmod: changing permissions

- `chmod <permissions> <files>`
2 formats for permissions:
- Octal format (abc):
 $a, b, c = r*4 + w*2 + x$ (r, w, x: booleans)
Example: `chmod 644 <file>`
(rw for u, r for g and o)
- Or symbolic format. Easy to understand by examples:
`chmod go+r`: add read permissions to group and others
`chmod u-w`: remove write permissions from user
`chmod a-x`: remove execute permission from all



More chmod

- `chmod -R a+rX linux/`
Makes linux and everything in it available to everyone!
- R: apply changes recursively
- X: x, but only for directories.
Very useful to open recursive access to directories,
without adding execution rights to all files.



Introduction to Unix and GNU / Linux

Standard I/O, redirections, pipes



Standard output

More about command output

- All the commands outputting text on your terminal do it by writing to their *standard output*.
- Standard output can be written (redirected) to a file using the > symbol
- Standard output can be appended to an existing file using the >> symbol



Standard output redirection examples

- `ls ~saddam/* > ~gwb/weapons_mass_destruction.txt`
- `cat obiwan_kenobi.txt > starwars_biographies.txt`
`cat han_solo.txt >> starwars_biographies.txt`
- `echo "README: No such file or directory" > README`
Useful way of creating a file without a text editor.
Nice Unix joke too in this case.



Standard input

More about command input

- Lots of command, when not given input arguments, can take their input from *standard input*.

- `sort`
`windows`
`linux`
`[Ctrl][D]`
`linux`
`windows`
sort takes its input from the standard input: in this case, what you type in the terminal (ended by `[Ctrl][D]`)

- `sort < participants.txt`
The standard input of sort is taken from the given file.



Pipes

- Unix pipes are very useful to redirect the standard output of some commands to the standard input of other ones.
- Examples
 - `cat *.log | grep -i error | sort`
 - `grep -ri error . | grep -v "ignored" | sort -u \> serious_errors.log`
 - `cat /home/*/homework.txt | grep mark | more`
- This one of the most powerful features in Unix shells!



The tee command

```
tee [-a] file
```

- The `tee` command can be used to send standard output to the screen and to a file simultaneously
- `make | tee build.log`
Runs the `make` command and stores its output to `build.log`
- `make install | tee -a build.log`
Runs the `make install` command and appends its output to `build.log`



Standard error

- Error messages are usually output (if the program is well written) to *standard error* instead of standard output.
- Standard error can be redirected through `2>` or `2>>`
- Example:

```
cat f1 f2 nofile > newfile 2> errfile
```
- Note: `1` is the descriptor for standard output, so `1>` is equivalent to `>`
- Can redirect both standard output and standard error to the same file using `&>`

```
cat f1 f2 nofile &> wholefile
```



Introduction to Unix and GNU / Linux

Task control



Full control on tasks

- Since the beginning, Unix supports true preemptive multitasking.
- Ability to run many tasks in parallel, and abort them even if they corrupt their own state and data.
- Ability to choose which programs you run.
- Ability to choose which input your programs takes, and where their output goes.



Running jobs in background

Same usage throughout all the shells

- Useful
 - For command line jobs which output can be examined later, especially for time consuming ones.
 - To start graphical applications from the command line and then continue with the mouse.
- Starting a task: add & at the end of your line:

```
find_prince_charming --cute --clever --rich &
```



Background job control

- jobs

Returns the list of background jobs from the same shell

```
[1]-  Running ~/bin/find_meaning_of_life --without-god &  
[2]+  Running make mistakes &
```

- fg

fg %<n>

Puts the last / nth background job in foreground mode

- Moving the current task in background mode:

[Ctrl] Z

bg

- kill %<n>

Aborts the nth job.



Job control example

```
> jobs
[1]-  Running ~/bin/find_meaning_of_life --without-god &
[2]+  Running make mistakes &

> fg
make mistakes

> [Ctrl] Z
[2]+  Stopped make mistakes

> bg
[2]+  make mistakes &

> kill %1
[1]+  Terminated ~/bin/find_meaning_of_life --without-god
```



Listing all processes

... whatever shell, script or process they are started from

- `ps -ux`

Lists all the processes belonging to the current user

- `ps -aux` (Note: `ps -edf` on System V systems)

Lists all the processes running on the system

- `ps -aux | grep bart | grep bash`

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
bart	3039	0.0	0.2	5916	1380	pts/2	S	14:35	0:00	/bin/bash
bart	3134	0.0	0.2	5388	1380	pts/3	S	14:36	0:00	/bin/bash
bart	3190	0.0	0.2	6368	1360	pts/4	S	14:37	0:00	/bin/bash
bart	3416	0.0	0.0	0	0	pts/2	RW	15:07	0:00	[bash]

- PID: Process id

VSZ: Virtual process size (code + data + stack)

RSS: Process resident size: number of KB currently in RAM

TTY: Terminal

STAT: Status: R (Runnable), S (Sleep), W (paging), Z (Zombie)...



Killing processes

- `kill <pids>`

Sends an abort signal to the given processes ids. Lets processes save data and exit by themselves. Should be used first. Example:

```
kill 3039 3134 3190 3416
```

- `kill -9 <pids>`

Sends an immediate termination signal to the given processes. The system itself terminates the processes. Useful when a process is really stuck (doesn't answer to `kill -1`).

- `killall [-<signal>] <command>`

Kills all the job running `<command>`. Example:

```
killall bash
```

- `kill -9 -1`

Kills all the processes of the current user. `-1`: means all processes.



Live process activity

- top

Displays most important processes, sorted by cpu percentage

```
top - 15:44:33 up 1:11, 5 users, load average: 0.98, 0.61, 0.59
Tasks: 81 total, 5 running, 76 sleeping, 0 stopped, 0 zombie
Cpu(s): 92.7% us, 5.3% sy, 0.0% ni, 0.0% id, 1.7% wa, 0.3% hi, 0.0% si
Mem: 515344k total, 512384k used, 2960k free, 20464k buffers
Swap: 1044184k total, 0k used, 1044184k free, 277660k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3809	jdoe	25	0	6256	3932	1312	R	93.8	0.8	0:21.49	bunzip2
2769	root	16	0	157m	80m	90m	R	2.7	16.0	5:21.01	X
3006	jdoe	15	0	30928	15m	27m	S	0.3	3.0	0:22.40	kdeinit
3008	jdoe	16	0	5624	892	4468	S	0.3	0.2	0:06.59	autorun
3034	jdoe	15	0	26764	12m	24m	S	0.3	2.5	0:12.68	kscd
3810	jdoe	16	0	2892	916	1620	R	0.3	0.2	0:00.06	top

- You can change the sorting order by typing
M: Memory usage, P: %CPU, T: Time.
- You can kill a task by typing k and the process id.



Sequential commands

- Can type the following command in your terminal even when the current one is not over (editing more difficult)
- Can separate commands with the `;` symbol:
`echo "I love thee"; sleep 10; echo " not"`
- Conditional use `||` (or) or `&&` (and):
`more God || echo "Sorry, God doesn't exist"`
Runs echo only if the first command fails
`ls ~sd6 && cat ~sd6/* > ~sidney/recipes.txt`
Only cats the directory contents if the `ls` command succeeds (means read access).



environment variables

- Shells let the user define *variables*:
Variables that can be reused in shell commands.
Convention: lower case names
- You can also define *environment variables*: variables that can be used within scripts or executables called from the shell.
Convention: upper case names
- `env`
Lists all defined environment variables and their value



Shell variables examples

Shell variables (bash)

- `projdir=/home/marshall/coolstuff`
`ls -la $projdir; cd $projdir`

Environment variables (bash)

- `cd $HOME`
- `export DEBUG=1`
`./find_extraterrestrial_life`
(displays debug information if DEBUG is set)



Main standard environment variables

Used by lots of applications!

- `LD_LIBRARY_PATH`
Shared library search path
- `DISPLAY`
Screen id to display X
(graphical) applications on.
- `EDITOR`
Default editor (vi, emacs...)
- `HOME`
Current user home directory
- `HOSTNAME`
Name of the local machine
- `MANPATH`
Manual page search path
- `PATH`
Command search path
- `PRINTER`
Default printer name
- `SHELL`
Current shell name
- `TERM`
Current terminal name / mode
- `USER`
Current user name



PATH environment variables

- PATH

Specifies the shell search order for commands

```
/
home/acox/bin:/usr/local/bin:/usr/kerberos/bin:
/usr/bin:/bin:/usr/X11R6/bin:/bin:/usr/bin
```

- LD_LIBRARY_PATH

Specifies the shared library (binary code libraries shared by applications, like the C library) search order for ld

```
/usr/local/lib:/usr/lib:/lib:/usr/X11R6/lib
```

- MANPATH

Specifies the search order for manual pages

```
/usr/local/man:/usr/share/man
```



PATH usage warning

It is strongly recommended not to have the “.” directory in your PATH environment variable, in particular not at the beginning:

- A cracker could place a malicious `ls` file in your home directory. It would get executed when you run `ls` in this directory and could do naughty things to your data.
- If you have an executable file called `test` in a directory, this will override the default `test` program and some scripts will stop working properly.
- Each time you `cd` to a new directory, the shell will waste time updating available commands.

Call your local commands as follows: `./test`



Alias

Shells let you define command *aliases*: shortcuts for commands you use very frequently

Examples

- `alias ls='ls -la'`
Useful to always run commands with default arguments
- `alias rm='rm -i'`
Useful to make rm always ask for confirms
- `alias frd='find_rambaldi_device --asap --risky'`
Useful to replace very long and frequent commands
- `alias cia='. /home/sidney/env/cia.sh'`
Useful to set an environment in a quick way
(. is a shell command to execute the content of a shell script)



.bashrc file

- **.bashrc**
Shell script read each time a bash shell is created
- You can use this file to define
 - Your default environment variables (PATH, EDITOR...)
 - Your aliases
 - Your prompt (see the bash manual for details)
 - A greeting message



Introduction to Unix and GNU / Linux

Misc utilities



Text editors

Either

- Graphical text editors
Fine for most needs
 - nedit
 - Emacs, Xemacs
- Text-only text editors
Often needed for sysadmins and great for power users
 - vi
 - nano



nedit

```
Makefile - /data/mike/handhelds/stock_kernel/linux-2.6.8.1/arch/arm/
File Edit Search Preferences Shell Macro Windows Help
#
# arch/arm/Makefile
#
# This file is subject to the terms and conditions of the GNU General Public
# License. See the file "COPYING" in the main directory of this archive
# for more details.
#
# Copyright (C) 1995-2001 by Russell King

LDFLAGS_vmlinux :=-p --no-undefined -X
LDFLAGS_BLOB :=--format binary
AFLAGS_vmlinux.lds.o = -DTEXTADDR=$(TEXTADDR) -DDATAADDR=$(DATAADDR)
OBJCOPYFLAGS :=-O binary -R .note -R .comment -S
GZFLAGS :=-9
#CFLAGS +=-pipe

ifeq ($(CONFIG_FRAME_POINTER), y)
CFLAGS +=-fno-omit-frame-pointer -mapcs -mno-sched-prolog
endif

ifeq ($(CONFIG_CPU_BIG_ENDIAN), y)
CFLAGS += -mbig-endian
AS += -EB
LD += -EB
AFLAGS += -mbig-endian
else
CFLAGS += -mlittle-endian
AS += -EL
LD += -EL
AFLAGS += -mlittle-endian
endif

comma = ,

# This selects which instruction set is used.
# Note that GCC does not numerically define an architecture version
# macro, but instead defines a whole series of macros which makes
# testing for a specific architecture or later rather impossible.
```

<http://www.nedit.org/>

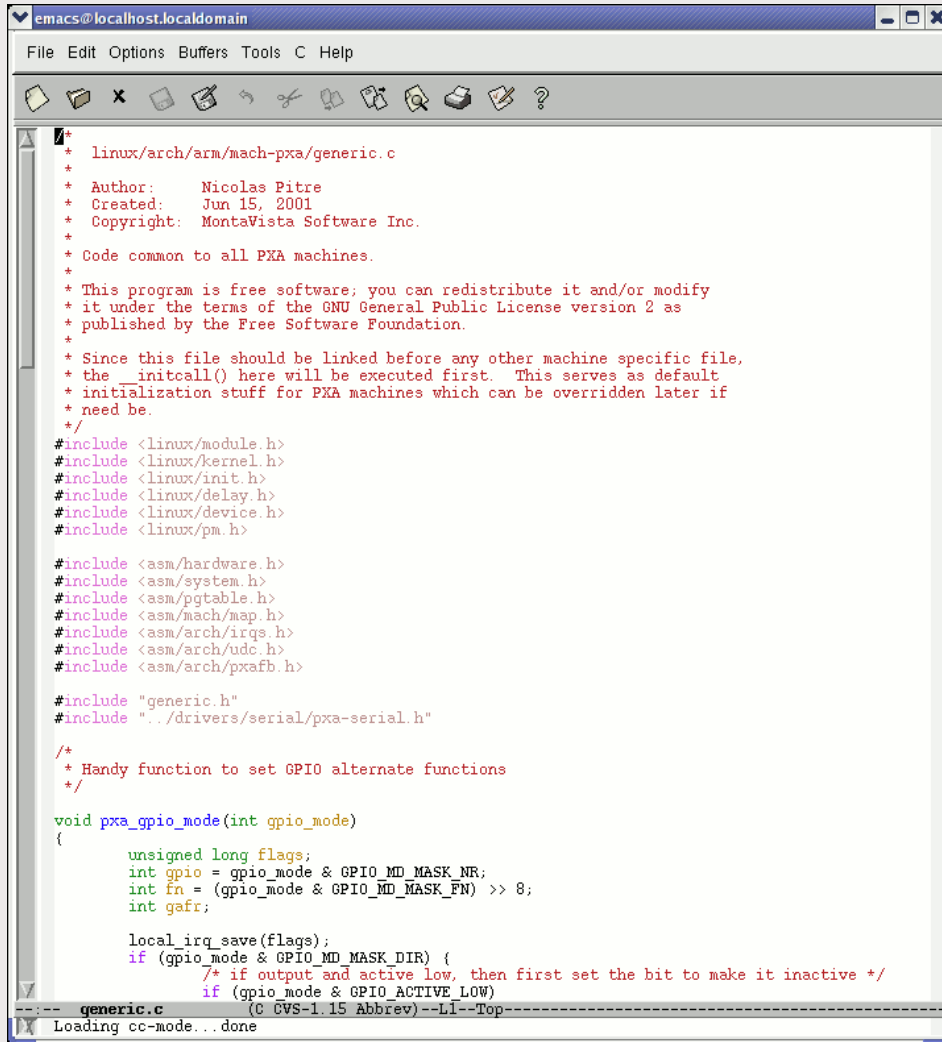


nedit (2)

- Best text editor for non vi or emacs experts
- Feature highlights:
 - Very easy text selection and moving
 - Syntax highlighting for most languages and formats. Can be tailored for your own log files, to highlight particular errors and warnings.
 - Easy to customize through menus
- Not installed by default by all distributions



Emacs / Xemacs



```
emacs@localhost.localdomain
File Edit Options Buffers Tools C Help

/*
 * linux/arch/arm/mach-pxa/generic.c
 *
 * Author:      Nicolas Pitre
 * Created:     Jun 15, 2001
 * Copyright:   MontaVista Software Inc.
 *
 * Code common to all PXA machines.
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 *
 * Since this file should be linked before any other machine specific file,
 * the __initcall() here will be executed first. This serves as default
 * initialization stuff for PXA machines which can be overridden later if
 * need be.
 */
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/device.h>
#include <linux/pm.h>

#include <asm/hardware.h>
#include <asm/system.h>
#include <asm/pgtable.h>
#include <asm/mach/map.h>
#include <asm/arch/irqs.h>
#include <asm/arch/udc.h>
#include <asm/arch/pxafb.h>

#include "generic.h"
#include "../drivers/serial/pxa-serial.h"

/*
 * Handy function to set GPIO alternate functions
 */

void pxa_gpio_mode(int gpio_mode)
{
    unsigned long flags;
    int gpio = gpio_mode & GPIO_MD_MASK_NR;
    int fn = (gpio_mode & GPIO_MD_MASK_FN) >> 8;
    int gafr;

    local_irq_save(flags);
    if (gpio_mode & GPIO_MD_MASK_DIR) {
        /* if output and active low, then first set the bit to make it inactive */
        if (gpio_mode & GPIO_ACTIVE_LOW)
            generic.c
            (C CVS-1.15 Abbrev)--LI--Top
Loading cc-mode... done
```

- Emacs and Xemacs are pretty similar (up to your preference)
- Extremely powerful text editor features
- Great for power users
- Less ergonomic than nedit
- Non standard shortcuts
- Much more than a text editor (games, e-mail, shell, browser)
- Some power commands have to be learnt



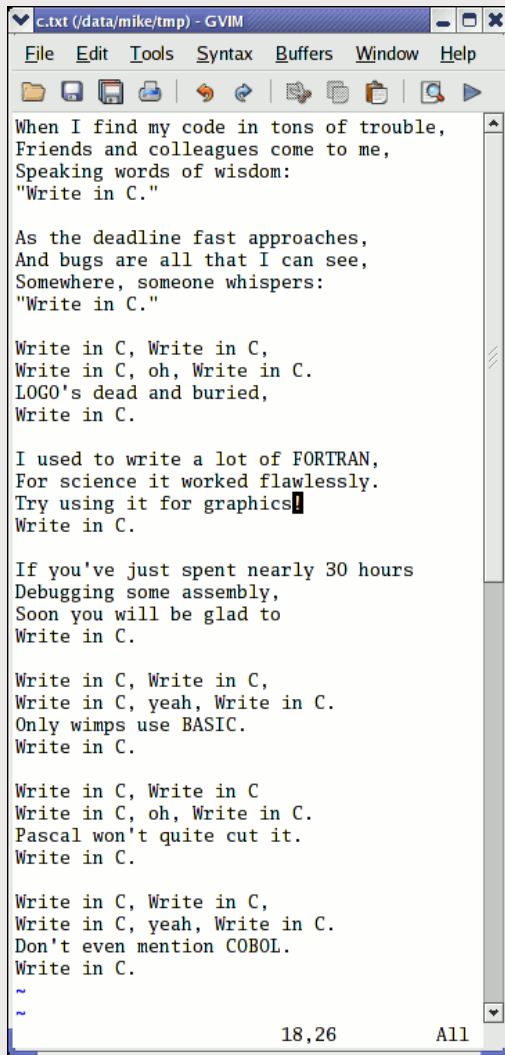
Text-mode text editor available in all Unix systems.

Created before computers with mice appeared.

- Difficult to learn for beginners used with graphical text editors.
- Very productive for power users.
- Often can't be replaced to edit files in system administration or in Embedded Systems, when you just have a text console.



vim - vi improved



```
c.txt (/data/mike/tmp) - GVIM
File Edit Tools Syntax Buffers Window Help
When I find my code in tons of trouble,
Friends and colleagues come to me,
Speaking words of wisdom:
"Write in C."

As the deadline fast approaches,
And bugs are all that I can see,
Somewhere, someone whispers:
"Write in C."

Write in C, Write in C,
Write in C, oh, Write in C.
LOGO's dead and buried,
Write in C.

I used to write a lot of FORTRAN,
For science it worked flawlessly.
Try using it for graphics
Write in C.

If you've just spent nearly 30 hours
Debugging some assembly,
Soon you will be glad to
Write in C.

Write in C, Write in C,
Write in C, yeah, Write in C.
Only wimps use BASIC.
Write in C.

Write in C, Write in C
Write in C, oh, Write in C.
Pascal won't quite cut it.
Write in C.

Write in C, Write in C,
Write in C, yeah, Write in C.
Don't even mention COBOL.
Write in C.
~
~
18,26 All
```

- vi implementation now found in most GNU / Linux host systems
- Implements lots of features available in modern editors: syntax highlighting, command history, help, unlimited undo and much much more.
- Cool feature example: can directly open compressed text files.
- Comes with a GTK graphical interface (gvim)
- Unfortunately, not free software (because of a small restriction in freedom to make changes)



vi basic command summary

[Esc]	Switch to command mode (when in edit mode)
i	Insert text
a	Identical to insert mode, except starts inserting at character after cursor
r	Replace character under cursor
R	Replace multiple characters
J	Join lines (default 2)
ndd	Delete n lines. Deleted lines are copied to buffer
nyy	Yank n lines. n lines starting at cursor line are copied to buffer
p	Paste contents of buffer to text starting at the line following the cursor line
D	Delete from cursor position to end of line
nG	Goto line n. If n is not specified, goes to the last line
H	Goto first line of file
/string	Find next occurrence of string
:1,\$s/str1/str2/g	Replaces every occurrence of the str1 with str2, starting from line 1 to the end of text
n	Find next occurrence of the last search string
?string	Find previous occurrence of string
^f	Go forward a page
^b	Go backward a page
h	Move cursor left
l	Move cursor right
j	Move cursor down
k	Move cursor up
^L	Redraw screen
u	Undo the latest change
u	Undo all changes on a line, while not having moved off of it
:wq	Exit vi, and save file
:w name	Save to file "name"
:x,y w name	Writes lines x through y to file "name"
:q!	Exit vi without saving changes
:f	Displays file information on bottom on screen



GNU nano

<http://www.nano-editor.org/>

- Another small text-only, mouse free text editor.
- An enhanced Pico clone (non free editor in Pine)
- Friendly and Easier to learn for beginners thanks to on screen command summaries.
- Available in binary packages for several platforms.
- An alternative to vi in embedded systems. However, not available as a busybox built-in.



GNU nano screenshot

```
GNU nano 1.2.3           File: fortune.txt

The herd instinct among economists makes sheep look like independent thinkers.

Klingon phaser attack from front!!!!
100% Damage to life support!!!

Spock: The odds of surviving another attack are 13562190123 to 1, Captain.

Quantum Mechanics is God's version of "Trust me."

I'm a soldier, not a diplomat.  I can only tell the truth.
    -- Kirk, "Errand of Mercy", stardate 3198.9

Did you hear that there's a group of South American Indians that worship
the number zero?

Is nothing sacred?

They are called computers simply because computation is the only significant
job that has so far been given to them.

As far as the laws of mathematics refer to reality, they are not
certain, and as far as they are certain, they do not refer to reality.
    -- Albert Einstein

Tact, n.:
    The unsaid part of what you're thinking.

Support bacteria -- it's the only culture some people have!

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Txt  ^T To Spell
```



Introduction to Unix and GNU / Linux

Misc

Compressing and archiving



Compressing

Very useful for shrinking huge files and saving space

- `[un]compress <file>`
Traditional Unix compression utility. Creates `.z` files.
Only kept for compatibility. Average performance.
- `g[un]zip <file>`
GNU zip compression utility. Creates `.gz` files.
Pretty good performance (similar to Zip).
- `b[un]zip2 <file>`
The most recent and performing compression utility.
Creates `.bz2` files. Usually 20-25% better than `gzip`.
Use this one! Now available on all Unix systems.



Archiving (1)

Useful to backup or release a set of files within 1 file

- `tar`: originally “tape archive”

- Creating an archive:

```
tar cvf <archive> <files or directories>
```

`c`: create

`v`: verbose. Useful to follow archiving progress

`f`: file. Archive created in file (tape used otherwise)

- Example:

```
tar cvf /backup/home.tar /home
```

```
bzip2 /backup/home.tar
```



Archiving (2)

- Viewing the contents of an archive or integrity check:
`tar tvf <archive>`
t: test
- Extracting all the files from an archive:
`tar xvf <archive>`
- Extracting just a few files from an archive:
`tar xvf <archive> <files or directories>`
Files or directories are given with paths relative to the archive root directory.




Extra options in GNU tar

`tar = gtar = GNU tar` on GNU / Linux

Can compress and uncompress archives on the fly.

Useful to avoid creating huge intermediate files

Much simpler to do than with `tar` and `bzip2`!

- `j` option: [un]compresses on the fly with `bzip2`
- `z` option: [un]compresses on the fly with `gzip`
- Examples (which one will you remember?) 
 - `gtar jcvf bills_bugs.tar.bz2 bills_bugs`
 - `tar cvf - bills_bugs | bzip2 > bills_bugs.tar.bz2`



Introduction to Unix and GNU / Linux

Misc Printing



Unix printing

- Multi-user, multi-job, multi-client, multi-printer
In Unix / Linux, printing commands don't really print. They send jobs to printing queues, possibly on the local machine, on network printing servers or on network printers.
- Printer independent system:
Print servers only accept jobs in PostScript or text. Printer drivers on the server take care of the conversion to each printers own format.
- Robust system:
Reboot a system, it will continue to print pending jobs.



Printing commands

- Useful environment variable: `PRINTER`
Set the default printer on the system. Example:
`export PRINTER=lp`
- `lpr [-P<queue>] <files>`
Sends the given files to the specified printing queue
The files must be in text or PostScript format. Otherwise, you only print garbage.
- `a2ps [-P<queue>] <files>`
“Any to PostScript” converts many formats to PostScript and send them to the specified queue. Useful features: several pages / sheet, page numbering, info frame...



Print job control

- `lpq [-P<queue>]`

Lists all the print jobs in the given or default queue.

```
lp is not ready
```

Rank	Owner	Job	File(s)	Total Size
1st	asloane	84	nsa_windows_backdoors.ps	60416 bytes
2nd	amoore	85	gw_bush_iraq_mistakes.ps	65024000 bytes

- `cancel <job#> [<queue>]`

Removes the given job number from the default queue



Using PostScript and PDF files

Viewing a PostScript file

- PostScript viewers exist, but their quality is pretty poor.
- Better convert to PDF with `ps2pdf`:
`ps2pdf decss_algorithm.ps`
`xpdf decss_algorithm.pdf &`

Printing a PDF file

- You don't need to open a PDF reader!
- Better convert to ps with `pdf2ps`:
`pdf2ps rambaldi_artifacts_for_dummies.pdf`
`lpr rambaldi_artifacts_for_dummies.ps`



Comparing files and directories

- `diff file1 file2`
Reports the difference between 2 files, or nothing if the files are identical.
- `diff -r dir/ dir2/`
Reports all the differences between files with the same name in the 2 directories.
- To investigate differences in details, better use graphical tools!



tkdiff

<http://tkdiff.sourceforge.net/>

Useful tool to compare files and merge differences

```
75 machine-$(CONFIG_ARCH_C0285) := footbridge
76 incdir-$(CONFIG_ARCH_C0285) := ebsa285
77 - machine-$(CONFIG_ARCH_FTVPCEI) := ftvpci
78 - incdir-$(CONFIG_ARCH_FTVPCEI) := nexuspci
79 - machine-$(CONFIG_ARCH_TBOX) := tbox
80 machine-$(CONFIG_ARCH_SHARK) := shark
81 machine-$(CONFIG_ARCH_SAL100) := sal100
82 ifeq ($(CONFIG_ARCH_SAL100),y)
83 # SAL111 DMA bug: we don't want the kernel to live in p
84 textaddr-$(CONFIG_SAL111) := 0xc0208000
85 endif
86 machine-$(CONFIG_ARCH_PXA) := pxa
87 machine-$(CONFIG_ARCH_L7200) := l7200
88 machine-$(CONFIG_ARCH_INTEGRATOR) := integrator
89 machine-$(CONFIG_ARCH_CAMELOT) := epxa10db
90 textaddr-$(CONFIG_ARCH_CLPS711X) := 0xc0028000
91 machine-$(CONFIG_ARCH_CLPS711X) := clps711x
92 textaddr-$(CONFIG_ARCH_FORTUNET) := 0xc0008000
93 machine-$(CONFIG_ARCH_IOP3XX) := iop3xx
94 ! machine-$(CONFIG_ARCH_ADI55XX) := adifcc
95 machine-$(CONFIG_ARCH_OMAP) := omap
96 machine-$(CONFIG_ARCH_S3C2410) := s3c2410
97 machine-$(CONFIG_ARCH_LH7A40X) := lh7a40x
98 machine-$(CONFIG_ARCH_VERSATILE_PB) := versatile
99
100 TEXTADDR := $(textaddr-y)
```

```
76 machine-$(CONFIG_ARCH_C0285) := footbridge
77 incdir-$(CONFIG_ARCH_C0285) := ebsa285
78
79 machine-$(CONFIG_ARCH_SHARK) := shark
80 machine-$(CONFIG_ARCH_SAL100) := sal100
81 ifeq ($(CONFIG_ARCH_SAL100),y)
82 # SAL111 DMA bug: we don't want the kernel to live in p
83 textaddr-$(CONFIG_SAL111) := 0xc0208000
84 endif
85 machine-$(CONFIG_ARCH_PXA) := pxa
86 machine-$(CONFIG_ARCH_L7200) := l7200
87 machine-$(CONFIG_ARCH_INTEGRATOR) := integrator
88 machine-$(CONFIG_ARCH_CAMELOT) := epxa10db
89 textaddr-$(CONFIG_ARCH_CLPS711X) := 0xc0028000
90 machine-$(CONFIG_ARCH_CLPS711X) := clps711x
91 textaddr-$(CONFIG_ARCH_FORTUNET) := 0xc0008000
92 machine-$(CONFIG_ARCH_IOP3XX) := iop3xx
93 ! machine-$(CONFIG_ARCH_IXP4XX) := ixp4xx
94 machine-$(CONFIG_ARCH_OMAP) := omap
95 machine-$(CONFIG_ARCH_S3C2410) := s3c2410
96 machine-$(CONFIG_ARCH_LH7A40X) := lh7a40x
97 machine-$(CONFIG_ARCH_VERSATILE_PB) := versatile
98 +ifeq ($(CONFIG_ARCH_EBSA110),y)
99 +# This is what happens if you forget the IOCS16 line.
100 +# PCMCIA cards stop working.
101 +CFLAGS_3c589_cs.o :=-DISA_SIXTEEN_BIT_PERIPHERAL
102 +export CFLAGS_3c589_cs.o
103 +endif
104
105 TEXTADDR := $(textaddr-y)
```



kompare

Another nice tool to compare files and merge differences
Part of the kdesdk package (Fedora Core)

```
File Difference Settings Help
Makefile
76 incdir-$(CONFIG_ARCH_C0285) := ebsa285
77 machine-$(CONFIG_ARCH_FTVPCCI) := ftvpcci
78 incdir-$(CONFIG_ARCH_FTVPCCI) := nexuspcci
79 machine-$(CONFIG_ARCH_TBOX) := tbox
80 machine-$(CONFIG_ARCH_SHARK) := shark
81 machine-$(CONFIG_ARCH_SA1100) := sa1100
82 ifeq ($(CONFIG_ARCH_SA1100),y)
83 # SA1111 DMA bug: we don't want the kernel to live in p
84 textaddr-$(CONFIG_SA1111) := 0xc0208000
85 endif
86 machine-$(CONFIG_ARCH_PXA) := pxa
87 machine-$(CONFIG_ARCH_L7200) := l7200
88 machine-$(CONFIG_ARCH_INTEGRATOR) := integrator
89 machine-$(CONFIG_ARCH_CAMELOT) := epxa10db
90 textaddr-$(CONFIG_ARCH_CLPS711X) := 0xc0028000
91 machine-$(CONFIG_ARCH_CLPS711X) := clps711x
92 textaddr-$(CONFIG_ARCH_FORTUNET) := 0xc0008000
93 machine-$(CONFIG_ARCH_IOP3XX) := iop3xx
94 machine-$(CONFIG_ARCH_ADIFCC) := adifcc
95 machine-$(CONFIG_ARCH_OMAP) := omap
96 machine-$(CONFIG_ARCH_S3C2410) := s3c2410
97 machine-$(CONFIG_ARCH_LH7A40X) := lh7a40x
98 machine-$(CONFIG_ARCH_VERSATILE_PB) := versatile
99
100 TEXTADDR := $(textaddr-y)
101 ifeq ($(incdir-y),)
102 incdir-y := $(machine-y)
103 endif
104 INCDIR := arch-$(incdir-y)
105
106 export TEXTADDR GZFLAGS
107
Makefile
75 incdir-$(CONFIG_FOOTBRIDGE) := ebsa285
75 textaddr-$(CONFIG_ARCH_C0285) := 0x60008000
76 machine-$(CONFIG_ARCH_C0285) := footbridge
77 incdir-$(CONFIG_ARCH_C0285) := ebsa285
78 machine-$(CONFIG_ARCH_SHARK) := shark
79 machine-$(CONFIG_ARCH_SA1100) := sa1100
80 ifeq ($(CONFIG_ARCH_SA1100),y)
82 # SA1111 DMA bug: we don't want the kernel to live in p
83 textaddr-$(CONFIG_SA1111) := 0xc0208000
84 endif
85 machine-$(CONFIG_ARCH_PXA) := pxa
86 machine-$(CONFIG_ARCH_L7200) := l7200
87 machine-$(CONFIG_ARCH_INTEGRATOR) := integrator
88 machine-$(CONFIG_ARCH_CAMELOT) := epxa10db
89 textaddr-$(CONFIG_ARCH_CLPS711X) := 0xc0028000
89 machine-$(CONFIG_ARCH_CLPS711X) := clps711x
90 textaddr-$(CONFIG_ARCH_FORTUNET) := 0xc0008000
91 machine-$(CONFIG_ARCH_IOP3XX) := iop3xx
92 machine-$(CONFIG_ARCH_IXP4XX) := ixp4xx
93 machine-$(CONFIG_ARCH_OMAP) := omap
94 machine-$(CONFIG_ARCH_S3C2410) := s3c2410
95 machine-$(CONFIG_ARCH_LH7A40X) := lh7a40x
96 machine-$(CONFIG_ARCH_VERSATILE_PB) := versatile
97
98 ifeq ($(CONFIG_ARCH_EBSA110),y)
99 # This is what happens if you forget the IOCS16 line.
100 # PCMCIA cards stop working.
101 CFLAGS_3c589_cs.o := -DISA_SIXTEEN_BIT_PERIPHERAL
102 export CFLAGS_3c589_cs.o
103 endif
104
105 TEXTADDR := $(textaddr-y)
```

Comparing file file:/data/mike/handhelds/stock_kernel/linux-2.6....data/mike/handhelds/stock_kernel/linux-2.6.8.1/arch/arm/Makefile 1 of 11 differences, 0 applied 1 of 1 file

The find command

Better explained by a few examples!

- `find . -name "*.pdf"`

Lists all the *.pdf files in the current (.) directory or subdirectories.

- `find docs -name "*.pdf" -exec xpdf {} ';'`

Finds all the *.pdf files in the docs directory and displays one after the other.

- Many more possibilities available! However, the above 2 examples cover most needs.



Getting information about users

- `who`
Lists all the users logged on the system
- `whoami`
Tells what user I am logged as
- `groups`
Tells which groups I belong to
- `groups <user>`
Tells which groups <user> belongs to
- `finger <user>`
Tells more details (real name, etc) about <user>
Disabled in some systems (security reasons)



Misc commands

- `sleep 60`
Waits for 60 seconds (doesn't consume system resources)
- `wc report.txt`
`438 2115 18302 report.txt`
Counts the number of lines, words and characters in a file or in standard input.



Introduction to Unix and GNU / Linux

Misc

Overview of a few desktop applications



Overview of desktop applications

To be shown on the screen with the projector!

- Mozilla: web browser, mail client and HTML editor
- Firefox: lightweight Mozilla based web browser
- OpenOffice: full featured, MS Office compatible office suite: word processor, spreadsheet, presentations, graphics...
- The GIMP: extremely powerful graphics editor
- Gqview: photo gallery viewer
- Evolution: Outlook-like e-mail client and calendar



GNU/Linux alternatives of Windows tools

Internet Explorer

IIS

Money

MS Office

MS Outlook

MS Project

Nero

Photoshop

WinAmp

W. Media Player

Mozilla

Firefox

Apache

GNU Cash

OpenOffice

Evolution

Mr Project

(Planner)

k3b

The GIMP

xmms

xine

mplayer

Don't know enough
Windows tools.



Send us more
correspondences!



Introduction to Unix and GNU / Linux

Going further



Command help

Some Unix commands and most GNU / Linux commands offer at least one help argument:

- `-h`
(`-` is mostly used to introduce 1-character options)
- `--help`
(`--` is always used to introduce the corresponding “long” option name, which makes scripts easier to understand)

You also often get a short summary of options when you input an invalid argument.



Manual pages

`man <keyword>`

Displays one or several manual pages for `<keyword>`

- `man man`

Most available manual pages are about Unix commands, but some are also about C functions, headers or data structures, or even about system configuration files!

- `man stdio.h`

- `man /etc/fstab`

Manual page files are looked for in the directories specified by the `MANPATH` environment variable.



Info pages

- In GNU, man pages are being replaced by info pages. Some manual pages even tell to refer to info pages instead.

`info <command>`

- Info features:
 - Documentation structured in sections (“nodes”) and subsections (“subnodes”)
 - Possibility to navigate in this structure: top, next, prev, up
 - Info pages generated from the same texinfo source has the HTML manual pages



Searching the Internet for resources (1)

Investigating issues

- Most forums and mailing list archives are public, and are indexed on a very frequent basis by **Google**.
- If you investigate an error message, copy it verbatim in the search form, enclosed in double quotes (“error message”). Lots of chances that somebody else already faced the same issue.
- Don't forget to use Google Groups: <http://groups.google.com/>
This site indexes more than 20 years of newsgroups messages.



Searching the Internet for resources (2)

Looking for documentation

- Look for `<tool>` or `<tool>` page to find the tool or project home page and then find the latest documentation resources.
- Look for `<tool>` documentation or `<tool>` manual in your favorite search engine.

Looking for generic technical information

- Wikipedia: <http://wikipedia.org>
Lots of useful definitions in computer science. A real encyclopedia! Open to anyone's contributions.



Introduction to Unix and GNU / Linux

Going further
Using GNU / Linux at home



GNU / Linux at home (1)

GNU / Linux is also a great alternative to Windows for home users

Security

- Virus free
Most viruses are designed to exploit Windows security flaws and have no impact on GNU / Linux
- Virus proof
Even if you executed a Linux compatible virus, it wouldn't have permissions to modify the system.
- Mistake proof
Other family members can't modify the system or somebody else's files either. They can only damage their own files.
- Cracker repellent
Even always connected to the Internet, your system attracts crackers less.



GNU / Linux at home (2)

Privacy

- Your system won't silently collect and transmit information about your movie or web site preferences.

User friendliness

- Your programs are made for users by users. They are more likely to satisfy your needs.
- Developers can easily be contacted to suggest new features.

Freedom

- Data you create are yours forever. They are not tied to a proprietary application through a proprietary (sometimes patented!) format.
- You are free to help your neighbors by sharing your programs with them.
- You are free to use your home programs at work too!



GNU / Linux at home (3)

You can migrate to GNU / Linux for:

- Office work: word processor, spreadsheet, presentations
- Internet navigation: web browsing and e-mail
- Multimedia: video, sound and graphics (including digital cameras)
- Learning about computers and computer programming

If you still have a copy of Windows, you can keep it (double boot) for:

- Gaming. Most consumer games still support Windows or Mac only.
- Using specific proprietary programs or educational cdroms
- Using hardware not supported yet on GNU / Linux



Try GNU / Linux at no risk

Knoppix is a live GNU / Linux cdrom

- Loads GNU / Linux in RAM, nothing is installed on your hard disk.
- Amazing hardware recognition capabilities.
- More than 2 GB of applications available!
- You can access your Windows files, open or even edit them with GNU / Linux applications.
- A great way to try and demonstrate GNU / Linux!
- Offers the possibility to make a permanent install on your hard disk



Using GNU / Linux distributions

GNU / Linux distributions

- Let you install GNU / Linux on free space on your hard disk, and still keep Windows (“double boot”)
- Have a very user-friendly installing interface which can automatically detect most hardware. You don't have any driver to install!
- Let you choose the types of applications to install
- Provide user friendly configuration interface
- Recommended distributions for beginners:
Fedora Core or Mandrake (see the next 2 pages)



Fedora Core

<http://fedora.redhat.com/>



- Stable and secure (access to Red Hat updates)
- Same user friendly interface as in Red Hat, easy to install
- No guaranteed support but access to Red Hat engineers when investigating bugs (same bug db)
- Frequent full releases (2 per year)
- A bit less stable than Red Hat (more innovative)



Mandrake Linux

<http://www.mandrakelinux.com/>



- User friendly
- Easy to install
- Innovative
- Frequent full releases (2 per year)
- Less stable (not enough testing and taking user feedback into account)



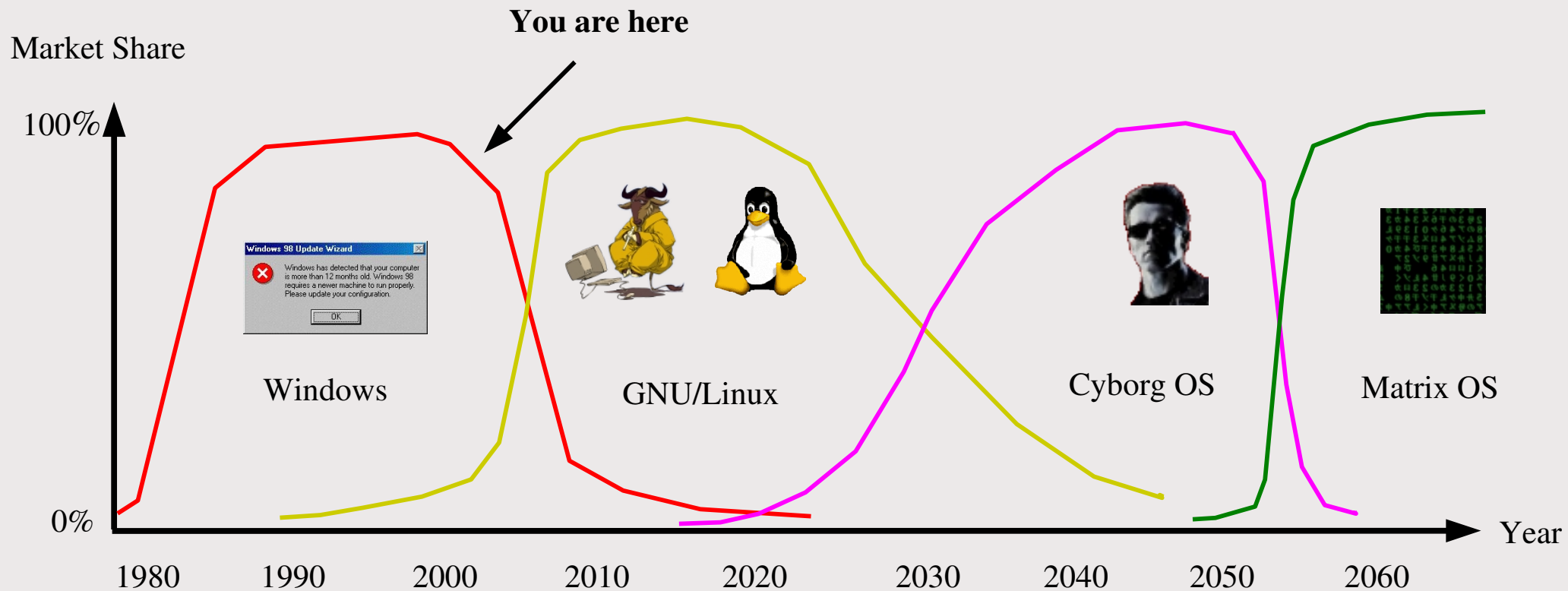
Introduction to Unix and GNU / Linux

Conclusion



Time to jump on the next train!

OS roadmap



Introduction to Unix and GNU / Linux

© Copyright 2004, Michael Opdenacker

GNU Free Documentation License

<http://free-electrons.com>



Related documents

This document belongs to the 500 page materials of an embedded GNU / Linux training from Free Electrons, available under the GNU Free Documentation License.

- Introduction to Unix and GNU / Linux
http://free-electrons.com/training/intro_unix_linux
- Embedded Linux kernel and driver development
<http://free-electrons.com/training/drivers>
- Development tools for embedded Linux systems
<http://free-electrons.com/training/devtools>
- Java in embedded Linux systems
<http://free-electrons.com/articles/java>
- What's new in Linux 2.6?
<http://free-electrons.com/articles/linux26>
- Introduction to uClinux
<http://free-electrons.com/articles/uclinux>
- Linux real-time extensions
<http://free-electrons.com/articles/realtime>



Training and consulting services

This training or presentation is funded by Free Electrons customers sending their people to our training or consulting sessions.

If you are interested in attending training sessions performed by the author of these documents, you are invited to ask your organization to order such sessions.

See <http://free-electrons/training> for more details.

If you just support this work, do not hesitate to speak about it to your friends, colleagues and local Free Software community.