

The Linux command line



# Training Contents (1)

# Shells, filesystem and file handling

* Everything is a file
* GNU / Linux filesystem structure
* Command line interpreters
* Handling files and directories
* Displaying, scanning and sorting files
* Symbolic and hard link
* File access rights

**Training contents (2)**

Standard I/O, redirections, pipes

* Standard input and output, redirecting to files
* Pipes: redirecting standard output to other com mands
* Standard error

**Training Contents (3)**

Task control

* + Full control on tasks
  + Executing in background, suspending, resuming and ab orting
  + List of active tasks
  + Killing processes
  + Environment variables
  + PATH environment variables
  + Shell aliases, .bashrc file

**Training contents (4)**

Miscellaneous

* Text editors
* Compressing and archiving
* Printing files
* Comparing files and directories
* Looking for files

**Linux command line**

Linux filesystem

**Everything is a file**

Almost everything in Unix is a file!

* Regular files
* Directories

Directories are just files listing a set of files

* Symbolic links

Files referring to the name of another file

* Devices and peripherals Read and write from devices as with regular files
* Pipes

Used to cascade programs

cat \*.log **|** grep error

* Sockets

Inter process communication

# File names

File name features since the beginning of Unix

* Case sensitive
* No obvious length limit
* Can contain any character (including whitespace, except /).

File types stored in the file (“magic numbers”).

File name extensions not needed and not interpreted. Just used for user convenience.

* File name examples:

README .bashrc Windows Buglist index.htm index.html index.html.old

# File paths

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

* / : *root directory*.

Start of absolute paths for all files on the system (even for files on removable devices or network shared).

# Linux filesystem structure (1)

Not imposed by the system. Can vary from one system to the other, even between two 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

# Linux filesystem structure (2)

/lost+found Corrupt files the system tried to recover

/media Mount points for removable media:

/media/usbdisk,

/media/cdrom

/mnt/ Mount points for temporarily mounted filesystems

/opt/ Specific tools installed by the sysadmin

/usr/local/ often used instead

/proc/ Access to system information

/proc/cpuinfo,

/proc/version ...

/root/ root user home directory

/sbin/ Administrator-only commands

/sys/ System and device controls

(cpu frequency, device power, etc.)

# Linux filesystem structure (3)

/tmp/ Temporary files

/usr/ Regular user tools (not essential to the system)

/usr/bin/, /usr/lib/,

/usr/sbin...

/usr/local/ Specific software installed by the sysadmin

(often preferred to /opt/)

/var/ Data used by the system or system servers

/var/log/, /var/spool/mail

(incoming mail), /var/spool/lpd

(print jobs)...

The Unix filesystem structure is defined

by the Filesystem Hierarchy Standard (FHS): <http://www.pathname.com/fhs/>

# Linux command line

## Shells 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)

Traditional, basic shell found on Unix systems, by Steve Bourne.

* 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.

# 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 ­S (size)

Lists the biggest 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 \*txt

The shell first replaces \*txt by all the file and directory names ending by txt (including .txt), except those starting with ., and then executes the ls command line.

* ls ­d .\*

Lists all the files and directories starting with .

­d tells ls not to display the contents of directories.

* cat ?.log

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

# The cd and pwd commands

* cd <dir>

Changes the current directory to <dir>.

* cd ­

Gets back to the previous current directory.

* pwd

Displays the current directory ("working directory").

**The cp command**

* + 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 confirmation if the target file already exists

* + cp ­r <source\_dir> <target\_dir> (recursive) Copies the whole directory.

**mv and rm commands**

* + - mv <old\_name> <new\_name> (move)

Renames the given file or directory.

* + - mv ­i (interactive)

If the new file already exits, asks for user confirm

* + - rm file1 file2 file3 ... (remove)

Removes the given files.

* + - rm ­i (interactive)

Always ask for user confirm.

* + - rm ­r dir1 dir2 dir3 (recursive)

Removes the given directories with all their contents.

**Creating and removing directories**

* mkdir dir1 dir2 dir3 ... (make dir)

Creates directories with the given names.

* rmdir dir1 dir2 dir3 ... (remove dir)

Removes the given directories

Safe: only works when directories and empty. Alternative: rm ­r (doesn't need empty directories).

**Displaying file contents**

Several ways of displaying the contents of files.

* cat file1 file2 file3 ... (concatenate)

Concatenates and outputs the contents of the given files.

* more file1 file2 file3 ...

After each page, asks the user to hit a key to continue.

Can also jump to the first occurrence of a keyword (/ command).

* less file1 file2 file3 ...

Does more than more with less.

Doesn't read the whole file before starting.

Supports backward movement in the file (? command).

# 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> (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!

**The sed command**

* + sed is a Stream EDitor
  + It parses text files and implements a programming language to apply transformations on the text.
  + One of the most common usage of sed is text replacement, which relies on regular expressions
    - sed ­e 's/abc/def/' testfile will replace every string “abc” by “def” in the file testfile and display the result on the standard output.
    - sed 's/^[ \t]\*//' testfile will remove any tabulation or space at the beginning of a line

 sed 's/^|\([^|]\*\)|\([^|]\*\)|$/\1 ­> \2/' testfile replace lines like |string1|string2| by string1 ­> string2

# sed : regular expressions

* Regular expressions are useful in many Unix tools, not only sed.

They allow to match the input text against an expression.

* + . matches any character
  + [ ] matches any character listed inside the brackets
  + [^ ] matches any character not listed inside the brackets
  + ^ matches the beginning of the line
  + $ matches the end of the line
  + \* matches the previous element zero or more times, + matches the previous element one or more times, ? matches the previous element zero or one time
  + \( \) defines a sub-expression that can be later recalled by using

\n, where n is the number of the sub-expression in the regular expression

* + More at <http://www.regular-expressions.info/>

# 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, there is no impact on the hard link contents.
* The contents are removed when there are no more files (hard links) to them.

**Linux command line**

Command documentation

**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 fstab (for /etc/fstab)

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

**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](http://wikipedia.org/)

Lots of useful definitions in computer science. A real encyclopedia! Open to anyone's contributions.

# Linux command line

## Users and permissions

**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.

**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

* ­rw­r­r­

Readable and writable for file owner, only readable for others

* ­rw­r­

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

* drwx­

Directory only accessible by its owner

* ­r­x

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 u­w : remove write permissions from user.

chmod a­x : (a: all) remove execute permission from all.

**More chmod (1)**

chmod ­R a+rX linux/

Makes linux and everything in it available to everyone!

* R: apply changes recursively
* X: x, but only for directories and files already executable

Very useful to open recursive access to directories, without adding execution rights to all files.

**More chmod (2)**

chmod a+t /tmp

* t: (sticky). Special permission for directories, allowing only the directory and file owner to delete a file in a directory.
* Useful for directories with write access to anyone, like /tmp.
* Displayed by ls ­l with a t character.

**File ownership**

Particularly useful in (embedded) system development when you create files for another system.

* chown ­R sco /home/linux/src (­R : recursive)

Makes user sco the new owner of all the files in

/home/linux/src.

* chgrp ­R empire /home/askywalker

Makes empire the new group of everything in

/home/askywalker.

* chown ­R borg:aliens usss\_entreprise/

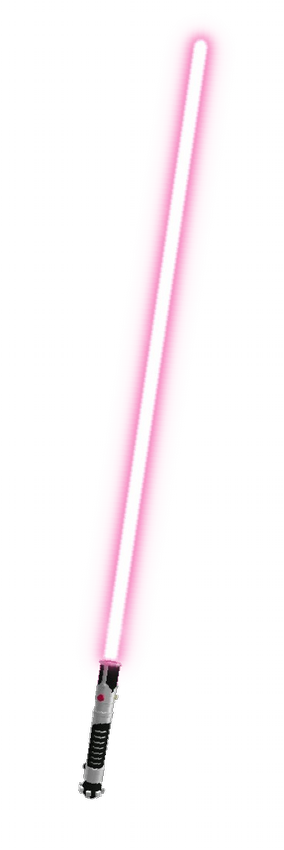
chown can be used to change the owner and group at the same time.

# Beware of the dark side of root

* + root user privileges are only needed for very specific tasks with security risks: mounting, creating device files, loading drivers, starting networking,

changing file ownership, package upgrades...

* + Even if you have the root password, your regular account should be sufficient for 99.9 % of your tasks (unless you are a system administrator).



* + In a training session, it is acceptable to use root.

In real life, you may not even have access to this account, or put your systems and data at risk if you do.

# Using the root account

In case you really want to use root...

* If you have the root password:

su ­ (**s**witch **u**ser)

* In modern distributions, the sudo command gives you access to some

root privileges with your own user password. Example: sudo mount /dev/hda4 /home

# Linux command line

## 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 commands, 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 a command to the standard input of another one.
    - 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

# Special devices (1)

Device files with a special behavior or contents

* /dev/null

The data sink! Discards all data written to this file. Useful to get rid of unwanted output, typically log information:

mplayer black\_adder\_4th.avi &> /dev/null

* /dev/zero

Reads from this file always return \0 characters Useful to create a file filled with zeros:

dd if=/dev/zero of=disk.img bs=1k count=2048

## See man null or man zero for details

**Special devices (2)**

* /dev/random

Returns random bytes when read. Mainly used by cryptographic programs. Uses interrupts from some device drivers as sources of true randomness (“entropy”).

Reads can be blocked until enough entropy is gathered.

* /dev/urandom

For programs for which pseudo random numbers are fine. Always generates random bytes, even if not enough entropy is available (in which case it is possible, though still difficult, to predict future byte sequences from past ones).

See man random for details.

# Linux command line

## 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.

**Processes**

“Everything in Unix is a file

Everything in Unix that is not a file is a process” Processes

Instances of a running programs

Several instances of the same program can run at the same time

 Data associated to processes:

Open files, allocated memory, stack, process id, parent, priority, state...

# 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)...

# 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,  Tasks: 81 total, 5 running, 76 sleeping, 0 stopped,  Cpu(s): 92.7% us, 5.3% sy, 0.0% ni, 0.0% id, 1.7% wa, | | | | 0.59  0 zombie  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.

# Killing processes (1)

 kill <pids>

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

* kill ­9 ­1

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

**Killing processes (2)**

* + killall [­<signal>] <command>

Kills all the jobs running <command>. Example:

killall bash

* + xkill

Lets you kill a graphical application by clicking on it!

Very quick! Convenient when you don't know the application command name.

**Recovering from stuck graphics**

* If your graphical session is stuck and you can no longer type in your terminals, don't reboot!
* It is very likely that your system is still fine. Try to access a text console by pressing the [Ctrl][Alt][F1] keys

(or [F2],[F3] for more text consoles)

* In the text console, you can try to kill the guilty application.
* Once this is done, you can go back to the graphic session by pressing [Ctrl][Alt][F5] or [Ctrl][Alt][F7] (depending on your distribution)
* If you can't identify the stuck program, you can also kill all your processes: kill ­9 ­1

You are then brought back to the login screen.

# Sequential commands

* + Can type the next command in your terminal even when the current one is not over.
  + Can separate commands with the ; symbol:

echo “I love thee”; sleep 10; echo “ not”

* + Conditionals: use || (or) or && (and):

more God || echo “Sorry, God doesn't exist”

Runs echo only if the first command fails

ls ~sd6 && cat ~sd6/\* > ~sydney/recipes.txt Only cats the directory contents if the ls command succeeds (means read access).

**Quoting (1)**

Double (") quotes can be used to prevent the shell from interpreting spaces as argument separators, as well as to prevent file name pattern expansion.

* echo "Hello World" Hello World
* echo "You are logged as $USER" You are logged as bgates
* echo \*.log

find\_prince\_charming.log cosmetic\_buys.log

* echo "\*.log"

\*.log

# Quoting (2)

Single quotes bring a similar functionality, but what is between quotes is never substituted

* echo 'You are logged as $USER' You are logged as $USER

Back quotes (`) can be used to call a command within another

* cd /lib/modules/`uname ­r`; pwd

/lib/modules/2.6.9­1.6\_FC2

Back quotes can be used within double quotes

* echo "You are using Linux `uname ­r`" You are using Linux 2.6.9­1.6\_FC2

# Environment variables

* + Shells let the user define *variables*.

### They can be reused in shell commands. Convention: lower case names

* + You can also define *environment variables*: variables that are also visible 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 type

* USER

Current user name

# PATH environment variables

 PATH

Specifies the shell search order for commands

/

home/acox/bin:/usr/local/bin:/usr/kerberos/bi n:/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 directories. 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 its list of 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 confirmation.

* alias frd='find\_rambaldi\_device ­asap ­risky'

Useful to replace very long and frequent commands.

* alias cia='. /home/sydney/env/cia.sh'

Useful to set an environment in a quick way

(. is a shell command to execute the content of a shell script).

# The which command

Before you run a command, which tells you where it is found

* bash> which ls

alias ls='ls ­color=tty'

/bin/ls

* tcsh> which ls

ls: aliased to ls ­color=tty

* bash> which alias

/usr/bin/which: no alias in (/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin)

* tcsh> which alias

alias: shell built­in command.

### ~/.bashrc

**~/.bashrc file**

Shell script read each time a bash shell is started

* + 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.

# Command editing

## You can use the left and right arrow keys to move the cursor in the current command.

* + You can use [Ctrl][a] to go to the beginning of the line, and [Ctrl][e] to go to the end.
  + You can use the up and down arrows to select earlier commands.
  + You can use [Ctrl][r] to search inside the history of previous commands.

**Command history (1)**

* + - history

Displays the latest commands that you ran and their number. You can copy and paste command strings.

* + - You can recall the latest command:

!!

* + - You can recall a command by its number

!1003

* + - You can recall the latest command matching a starting string:

!cat

**Command history (2)**

* + - You can make substitutions on the latest command:

^more^less

* + - You can run another command with the same arguments:

more !\*

**Linux command line**

Miscellaneous

### Text editors

**Text editors**

Graphical text editors Fine for most needs

* nedit
* Emacs, Xemacs
* Kate, Gedit Text-only text editors

Often needed for sysadmins and great for power users

* vi, vim
* nano

**The nedit text editor**

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

Best text editor for non vi or emacs experts

 Feature highlights:

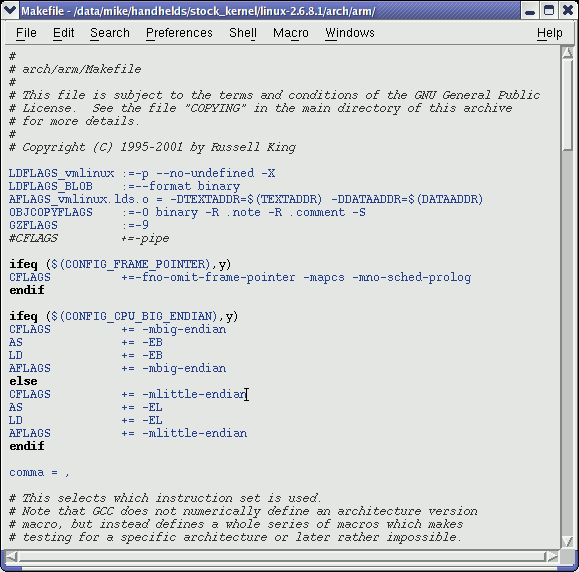
Very easy text selection and moving Syntax highlighting for most languages and

formats. Can be tailored for your own log files, to highlight particular errors and warnings.

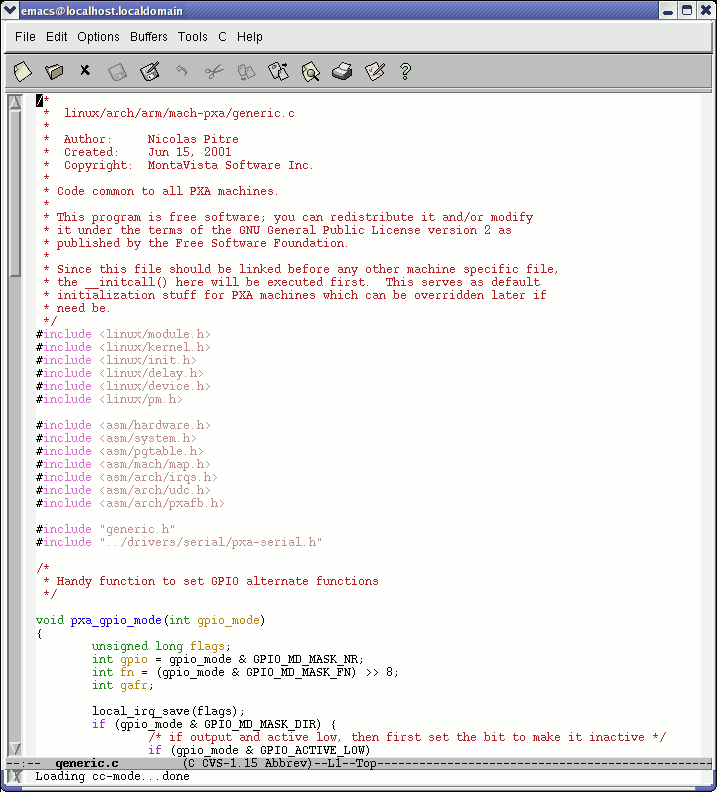
Easy to customize through menus

* Not installed by default by all distributions

**nedit screenshot**



**Emacs / Xemacs**

Emacs and Xemacs are pretty similar (up to your preference)

Extremely powerful text editor features

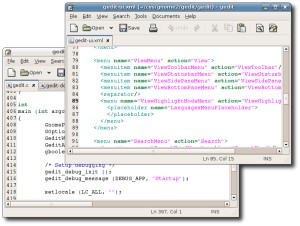
Great for power users Less ergonomic than nedit Non standard shortcuts

Much more than a text editor (games, e-mail, shell, browser).

Some power commands have to be learnt.

# Kate and gedit

## Kate is a powerful text editor dedicated to programming activities, for KDE

 [http://kate.kde.org](http://kate.kde.org/)

* Gedit is a text editor for the Gnome environment

**vi**

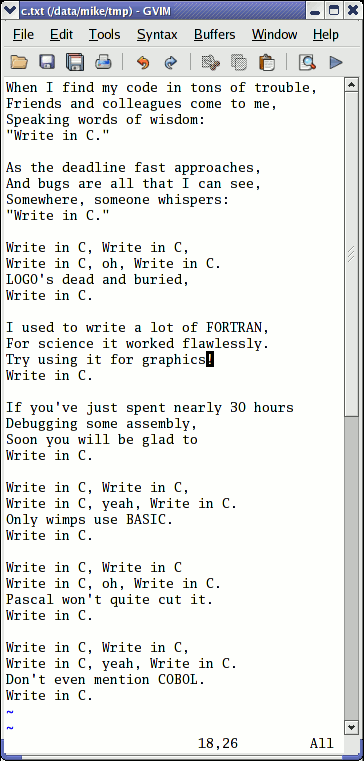
Text-mode text editor available in all Unix systems. Created before computers with mice appeared.

Difficult to learn for beginners used to 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.

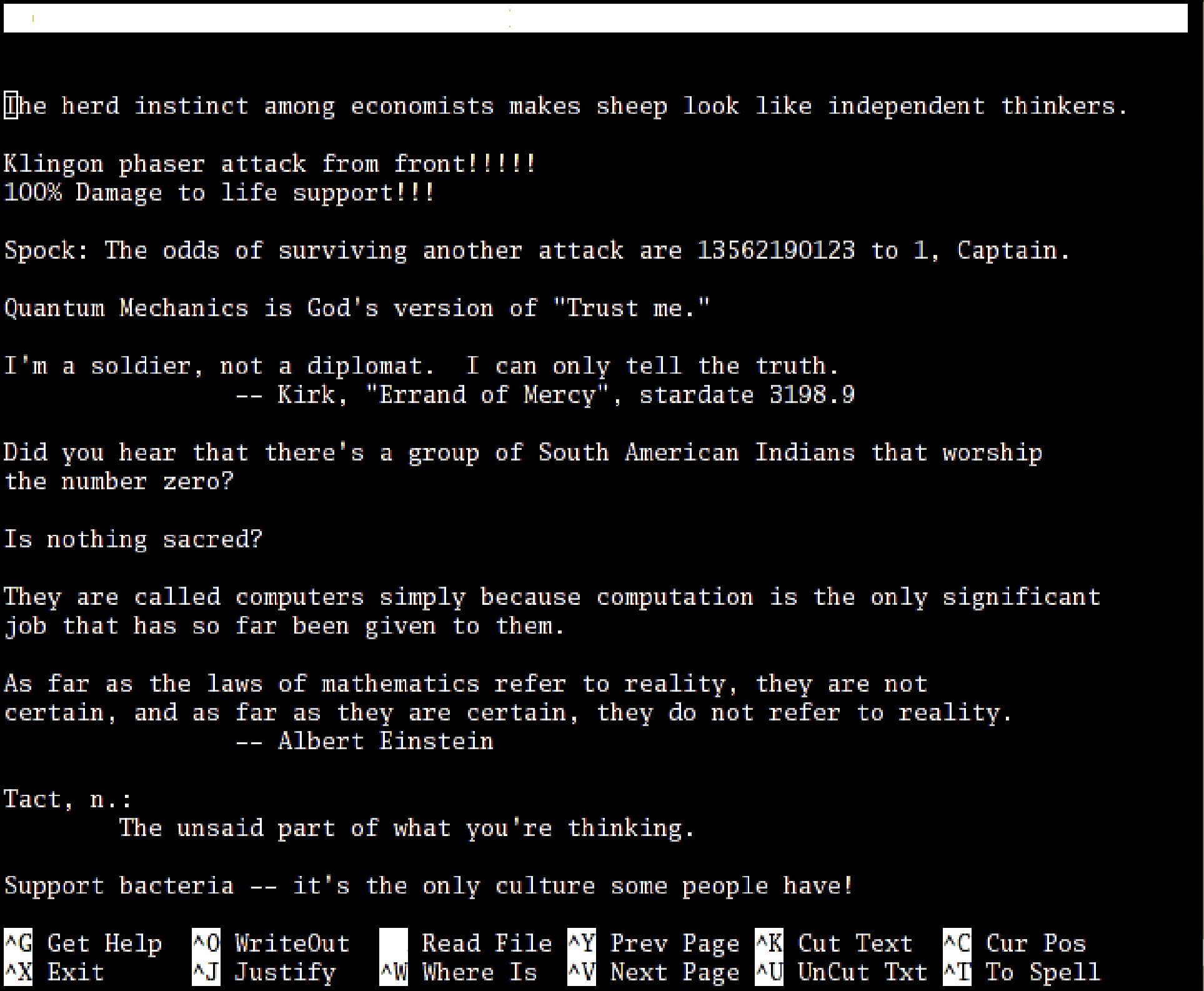
**vim - vi improved**

* 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)

# 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.



GNU nano 1.2.3

File: fortune.txt

^R

**Linux command line**

Miscellaneous

### Compressing and archiving

**Measuring disk usage**

Caution: different from file size!

* du ­h <file> (disk usage)

­h : returns size on disk of the given file, in human readable format: K (kilobytes), M (megabytes) or G (gigabytes), . Without ­h , du returns the raw number of disk blocks used by the file (hard to read).

Note that the ­h option only exists in GNU du.

* du ­sh <dir>

­s : returns the sum of disk usage of all the files in the given directory.

**Measuring disk space**

* df ­h <dir>

Returns disk usage and free space for the filesystem containing the given directory.

Similarly, the ­h option only exists in GNU df.

* Example:
* df ­h .

Filesystem Size Used Avail Use% Mounted on

/dev/hda5 9.2G 7.1G 1.8G 81% /

* df ­h

Returns disk space information for all filesystems available in the system. When errors happen, useful to look for full filesystems.

**Compressing and decompressing**

Very useful for shrinking huge files and saving space

* g[un]zip <file>

GNU zip compression utility. Creates .gz files. Ordinary performance (similar to Zip).

* b[un]zip2 <file>

More recent and effective compression utility. Creates .bz2 files. Usually 20-25% better than gzip.

* [un]lzma <file>

Much better compression ratio than bzip2 (up to 10 to 20%). Compatible command line options.

# 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
* ­lzma option: [un]compresses on the fly with lzma
* Examples (which one will you remember?) 
  + gtar jcvf bills\_bugs.tar.bz2 bills\_bugs
  + tar cvf ­bills\_bugs | bzip2 > bills\_bugs.tar.bz2

# Checking file integrity

Very low cost solution to check file integrity

* + md5sum FC3­i386­disk\*.iso > MD5SUM

Computes a MD5 (Message Digest Algorithm 5) 128 bit checksum of the given files. Usually redirected to a file.

* + Example output:

db8c7254beeb4f6b891d1ed3f689b412 FC3­i386­disc1.iso 2c11674cf429fe570445afd9d5ff564e FC3­i386­disc2.iso f88f6ab5947ca41f3cf31db04487279b FC3­i386­disc3.iso 6331c00aa3e8c088cc365eeb7ef230ea FC3­i386­disc4.iso

* + md5sum ­c MD5SUM

Checks the integrity of the files in MD5SUM by comparing their actual MD5 checksum with their original one.

# Linux command line

## Miscellaneous

### 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

Sets 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 the output 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 PostScript with pdf2ps:

pdf2ps rambaldi\_artifacts\_for\_dummies.pdf lpr rambaldi\_artifacts\_for\_dummies.ps

# Linux command line

## Miscellaneous

### Comparing files and directories

**Comparing files and directories**

* diff file1 file2

Reports the differences between 2 files, or nothing if the files are identical.

* diff ­r dir1/ dir2/

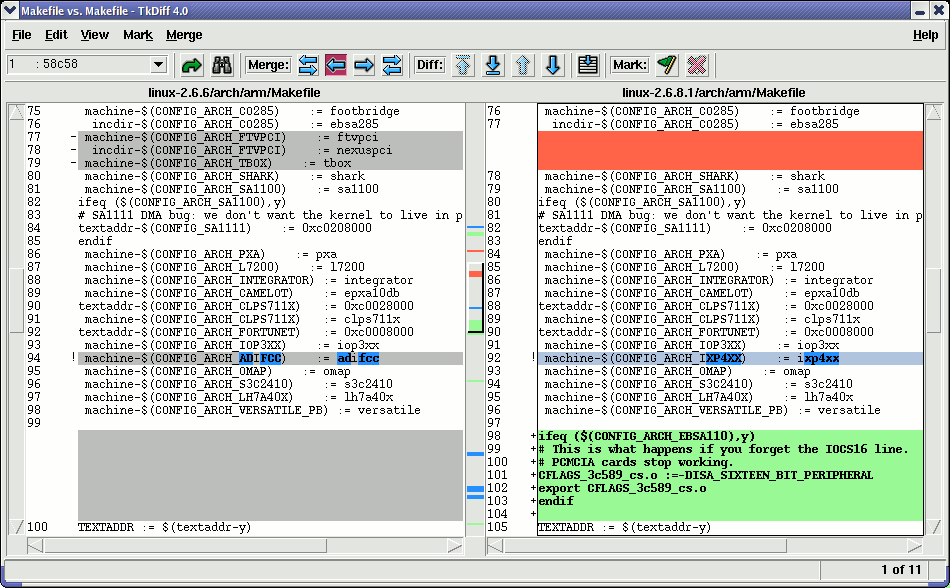
Reports all the differences between files with the same name in the 2 directories.

* These differences can be saved in a file using the redirection, and then later re-applied using the patch command.
* To investigate differences in detail, better use graphical tools!

**tkdiff**

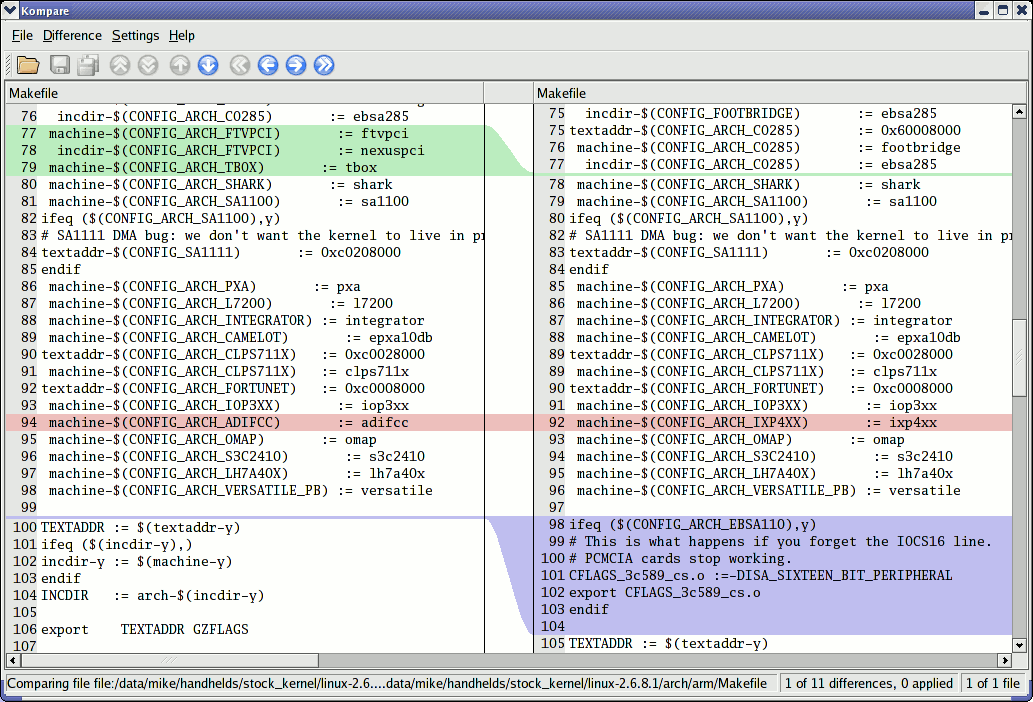
<http://tkdiff.sourceforge.net/>

Useful tool to compare files and merge differences



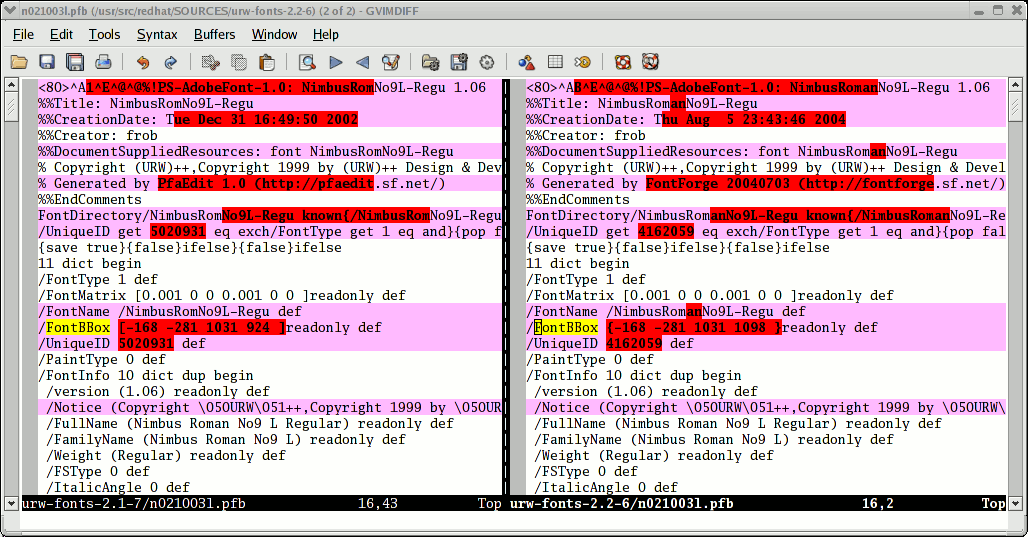
# kompare

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



# gvimdiff

## Another nice tool to view differences in files

Available in most distributions with gvim Apparently not using diff.

No issue with files with

binary sections!

# Linux command line

## Miscellaneous

### Looking for files

**The find command**

Better explained by a few examples!

* find . ­name “\*.pdf”

Lists all the \*.pdf files in the current (.) directory or subdirectories. You need the double quotes to prevent the shell from expanding the \* character.

* 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.

**The locate command**

Much faster regular expression search alternative to find

* locate keys

Lists all the files on your system with keys in their name.

* locate “\*.pdf”

Lists all the \*.pdf files available on the whole machine

* locate “/home/fridge/\*beer\*”

Lists all the \*beer\* files in the given directory (absolute path)

* locate is much faster because it indexes all files in a dedicated database, which is updated on a regular basis.
* find is better to search through recently created files.

# Linux command line

## Miscellaneous

### Various commands

**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).

# Changing users

## You do not have to log out to log on another user account!

* su hyde

(Rare) Change to the hyde account, but keeping the environment variable settings of the original user.

* su ­jekyll

(More frequent) Log on the jekyll account, with exactly the same settings as this new user.

* su ­

When no argument is given, it means the root

user.

**The wget command**

Instead of downloading files from your browser, just copy and paste their URL and download them with wget!

wget main features

* http and ftp support
* Can resume interrupted downloads
* Can download entire sites or at least check for bad links
* Very useful in scripts or when no graphics are available (system administration, embedded systems)
* Proxy support (http\_proxy and ftp\_proxy env. variables)

# wget examples

* wget ­c \<http://microsoft.com/customers/dogs/winxp4dogs.zip> Continues an interrupted download.
* wget ­m <http://lwn.net/>

Mirrors a site.

* wget ­r ­np <http://www.xml.com/ldd/chapter/book/>

Recursively downloads an on-line book for off-line access.

­np : "no-parent". Only follows links in the current directory.

# Misc commands (1)

### sleep 60

Waits for 60 seconds

(doesn't consume system resources).

* wc report.txt (word count)

438 2115 18302 report.txt

Counts the number of lines, words and characters in a file or in standard input.

**Misc commands (2)**

* bc ("basic calculator?")

bc is a handy but full-featured calculator. Even includes a programming language! Use the ­l option to have floating point support.

* date

Returns the current date. Useful in scripts to record when commands started or completed.