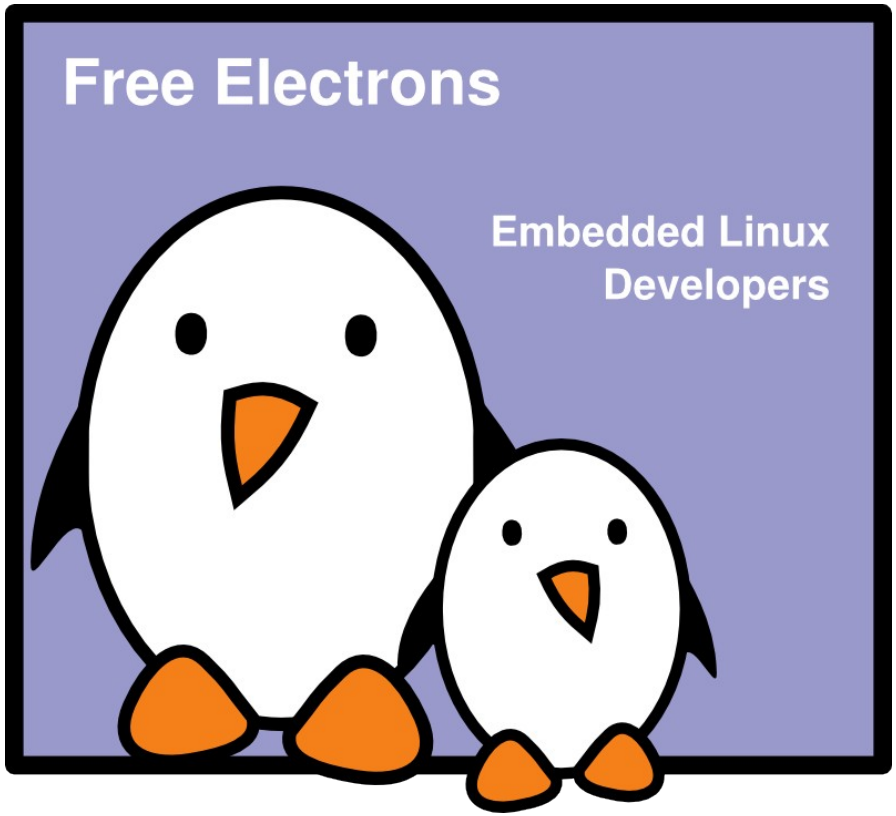
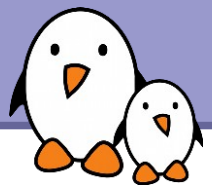
The Unix and GNU/Linux command line



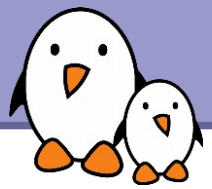
Michael Opdenacker
Thomas Petazzoni
Free Electrons

Abridged for ELE209 Lab 1
By Tim Toolan

Latest update: Feb 4, 2013,
Document sources, updates and translations:
http://free-electrons.com/docs/command-line
Corrections, suggestions, contributions and translations are welcome!



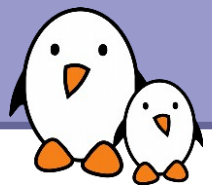
Unix filesystem



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

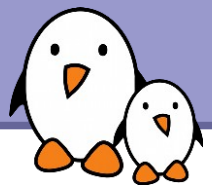


File names

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:

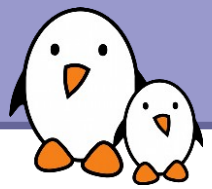
```
README          .bashrc          Windows Buglist
index.htm       index.html       index.html.old
```



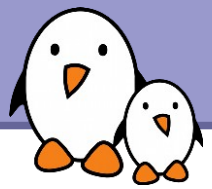
File paths

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).

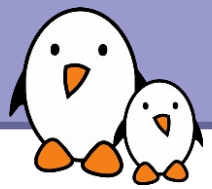


Shells and file handling



Command line interpreters

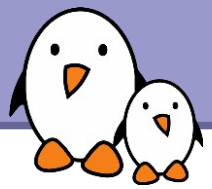
- Shells: tools to execute user commands
- Called “shells” because they hide the details on the underlying operating system under the shell's surface.
- Commands are input in a text terminal, either a window in a graphical environment or a text-only console.
- Results are also displayed on the terminal. 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

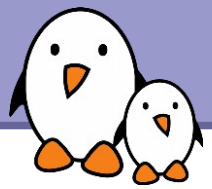
- `sh`: The Bourne shell (obsolete)
  Traditional, basic shell found on Unix systems, by Steve Bourne.
- `csh`: The C shell (obsolete)
  Once popular shell with a C-like syntax
- `tcsh`: The TC shell (still very popular)
  A C shell compatible implementation with evolved features (command completion, 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.

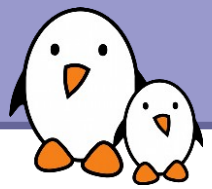
- `ls -a` (all)
  Lists all the files (including `.*` files)
- `ls -l` (long)
  Long listing (type, date, size, owner, permissions)
- `ls -t` (time)
  Lists the most recent files first
- `ls -S` (size)
  Lists the biggest files first
- `ls -r` (reverse)
  Reverses the sort order
- `ls -ltr` (options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



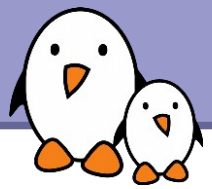
Special directories (1)

`./`

- 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 (enclosing) directory. Always belongs to the `.` directory (see `ls -a`). Only reference to the parent directory.
- Typical usage:
  `cd ..`



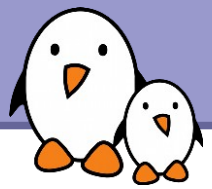
Special directories (2)

`~/`

- 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.

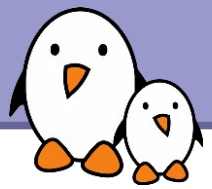
`~sydney/`

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



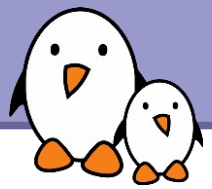
The cd and pwd commands

- `cd <dir>`
  Changes the current directory to `<dir>`.
- `cd -`
  Gets back to the previous current directory.
- `pwd`
  Displays the current directory ("working directory").



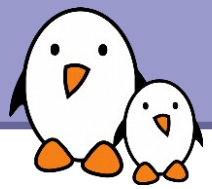
The cp command

- `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.



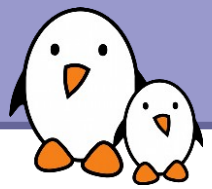
mv and rm commands

- `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

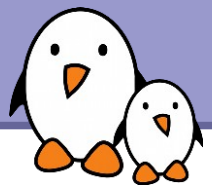
- `mkdir dir1 dir2 dir3 ...` (make dir)
  Creates directories with the given names.
- `rmdir dir1 dir2 dir3 ...` (remove dir)
  Removes the given directories
  Safe: only works when directories and empty.
  Alternative: `rm -r` (doesn't need empty directories).



Displaying file contents

Several ways of displaying the contents of files.

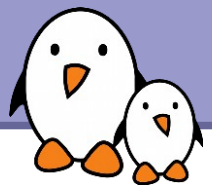
- `cat file1 file2 file3 ...` (concatenate)
  Concatenates and outputs the contents of the given files.
- `more file1 file2 file3 ...`
  After each page, asks the user to hit a key to continue.
  Can also jump to the first occurrence of a keyword (`/` command).
- `less file1 file2 file3 ...`
  Does more than `more` with less.
  Doesn't read the whole file before starting.
  Supports backward movement in the file (`?` command).



Symbolic links

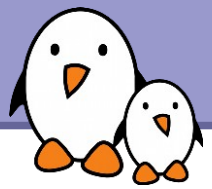
A symbolic link is a special file which is just a reference to the name of another one (file or directory):

- 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

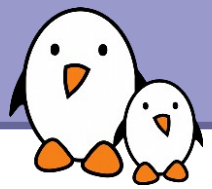
- To create a symbolic link (same order as in `cp`):
  ```
  ln -s file_name link_name
  ```
- To create a link with to a file in another directory, with the same name:
  ```
  ln -s ../README.txt
  ```
- To create multiple links at once in a given directory:
  ```
  ln -s file1 file2 file3 ... dir
  ```
- 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.

Command documentation

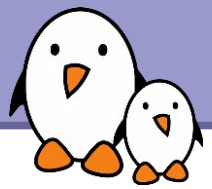


Command help

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>`

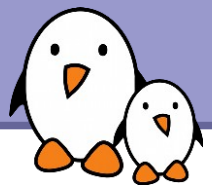
Displays one or several manual pages for `<keyword>`

- `man man`

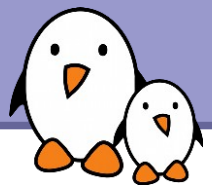
Most available manual pages are about Unix commands, but some are also about C functions, headers or data structures, or even about system configuration files!

- `man stdio.h`
- `man fstab` (for `/etc/fstab`)

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



Users and permissions



File access rights

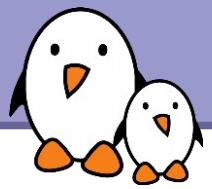
Use `ls -l` to check file access rights

3 types of access rights

- Read access (`r`)
- Write access (`w`)
- Execute rights (`x`)

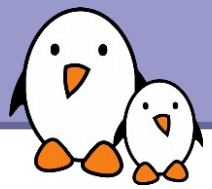
3 types of access levels

- 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



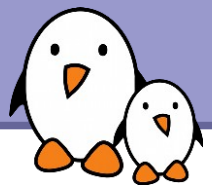
Access right constraints

- `x` is sufficient to execute binaries
  Both `x` and `r` and required for shell scripts.
- Both `r` and `x` permissions needed in practice for directories:
  `r` to list the contents, `x` to access the 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.



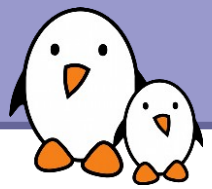
Access rights examples

- `-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`: (`a`: all) remove execute permission from all.

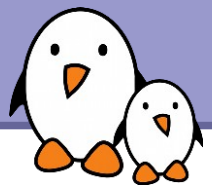


More chmod (1)

```
chmod -R a+rX linux/
```

Makes `linux` and everything in it available to everyone!

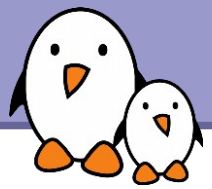
- `R`: apply changes recursively
- `X`: `x`, but only for directories and files already executable
  Very useful to open recursive access to directories, without adding execution rights to all files.



More chmod (2)

```
chmod a+t /tmp
```

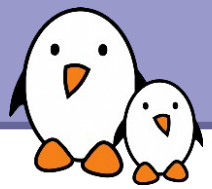
- `t`: (sticky). Special permission for directories, allowing only the directory and file owner to delete a file in a directory.
- Useful for directories with write access to anyone, like `/tmp`.
- Displayed by `ls -l` with a `t` character.



Quoting (1)

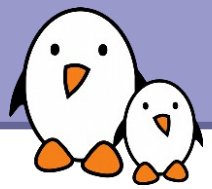
Double (") quotes can be used to prevent the shell from interpreting spaces as argument separators, as well as to prevent file name pattern expansion.

```
> 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
```



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 (tcsh)

Shell variables (tcsh)

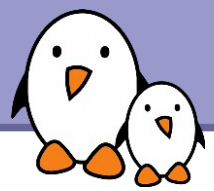
- `setenv projdir /home/marshall/coolstuff`
  `ls -la $projdir; cd $projdir`

Environment variables (tcsh)

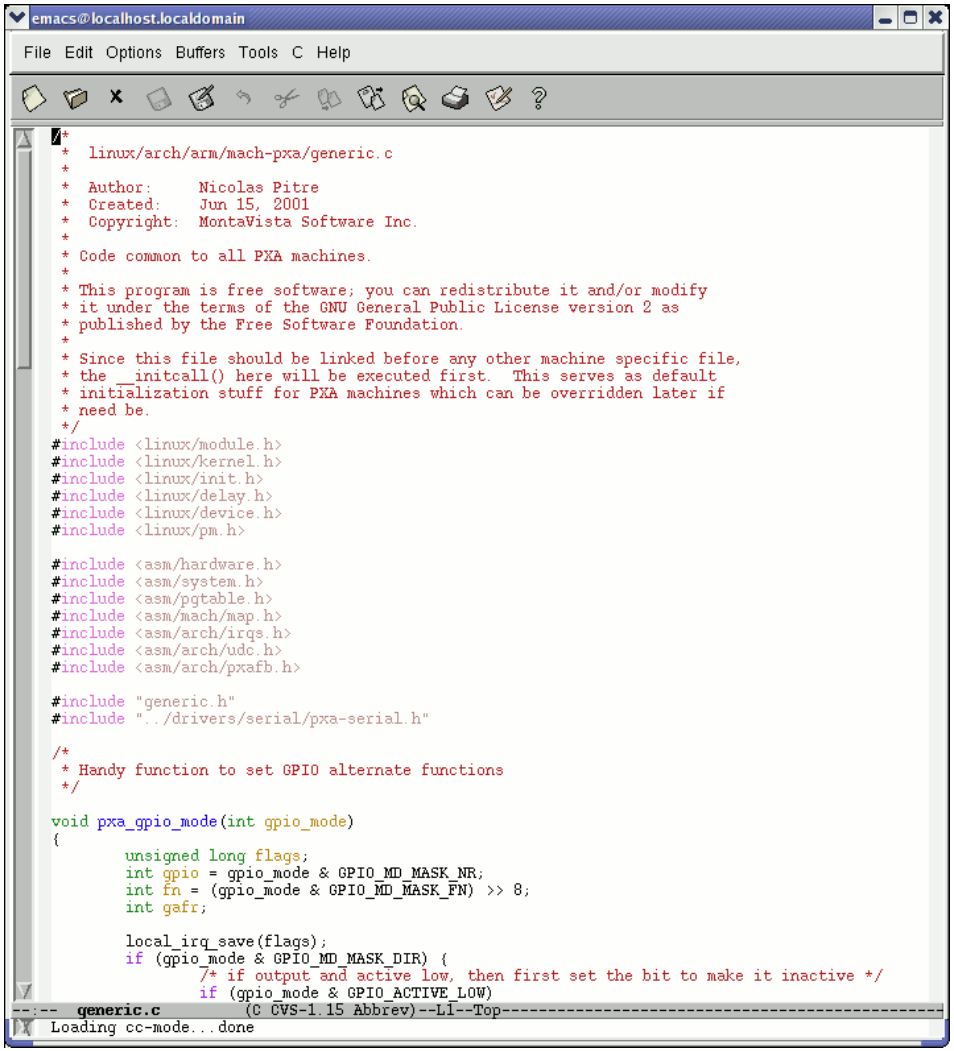
- `cd $HOME`
- `setenv DEBUG 1`
  `./find_extraterrestrial_life`
  (displays debug information if `DEBUG` is set)

Miscellaneous

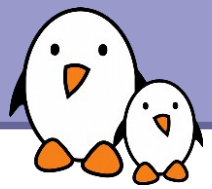
Text editors



Emacs / Xemacs



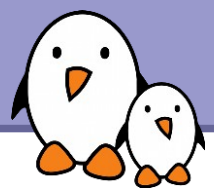
- Emacs and Xemacs are pretty similar (up to your preference)
- Extremely powerful text editor features
- Great for power users
- Less ergonomic than nedit
- Non standard shortcuts
- Much more than a text editor (games, e-mail, shell, browser).
- Some power commands have to be learnt.



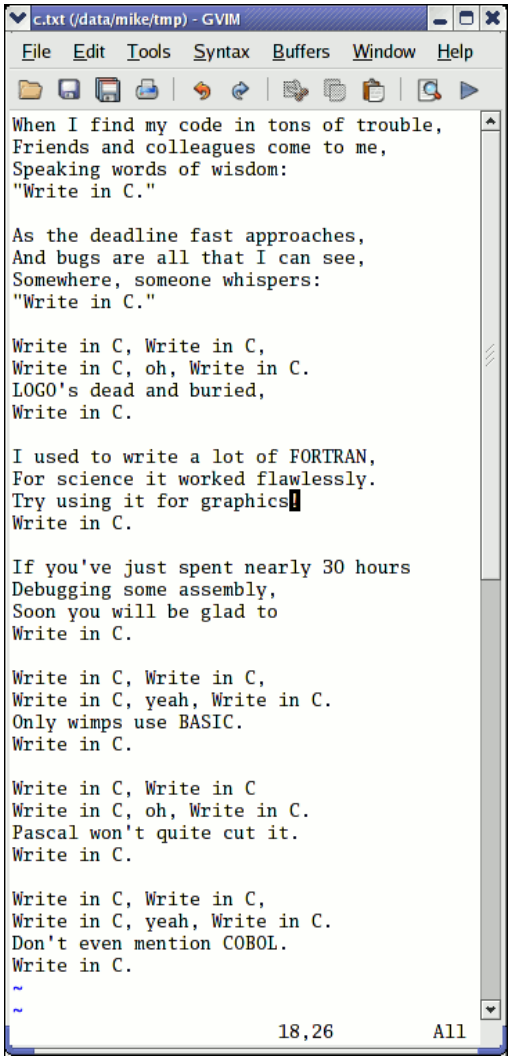
vi

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.



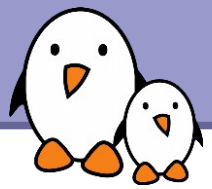
vim - vi improved



- `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)

Miscellaneous

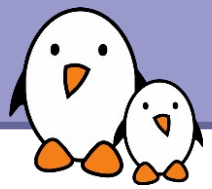
Looking for files



The find command

Better explained by a few examples!

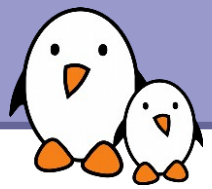
- `find . -name “*.pdf”`
  Lists all the `*.pdf` files in the current (`.`) directory or subdirectories. You need the double quotes to prevent the shell from expanding the `*` character.
- `find docs -name "*.pdf" -exec xpdf {} ';'`
  Finds all the `*.pdf` files in the `docs` directory and displays one after the other.
- Many more possibilities available! However, the above 2 examples cover most needs.



The locate command

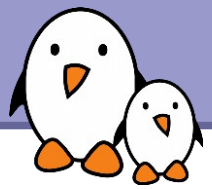
Much faster regular expression search alternative to `find`

- `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.



Miscellaneous

Various commands

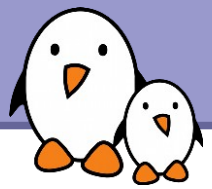


The wget command

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

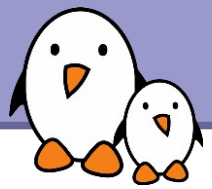
`wget` main features

- 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.