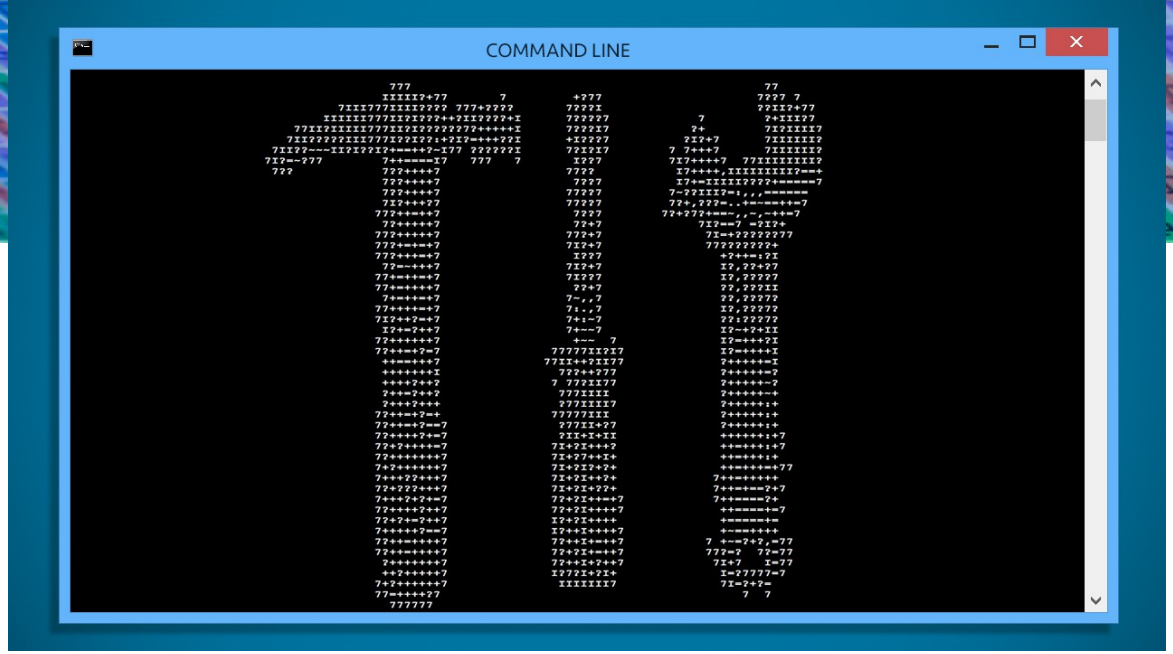


THE UNIX COMMAND LINE

Lecture 1

Summer school Malta 12 September 2016

Train MALTA Horizon 2020

UNIVERSITY OF CAMBRIDGE

Luigi Grassi

Department of Haematology

Long Road Cambridge, CB2 0PT UK

lg490@medschl.cam.ac.uk

UNIVERSITY OF CAMBRIDGE

Course Contents (I)

It is a unix course and has to be as much interactive as possible: please let try the commands as we are going to describe them!

- Unix files and structure
- Shells and file handling
- Unix Tools
- Users and permissions
- Coffee break
- Standard input/output, redirections, pipes

Course Contents (II)

- Task control
- Environment variables & bashrc
- Text editor (nano)
- Compressing and archiving
- Locating files

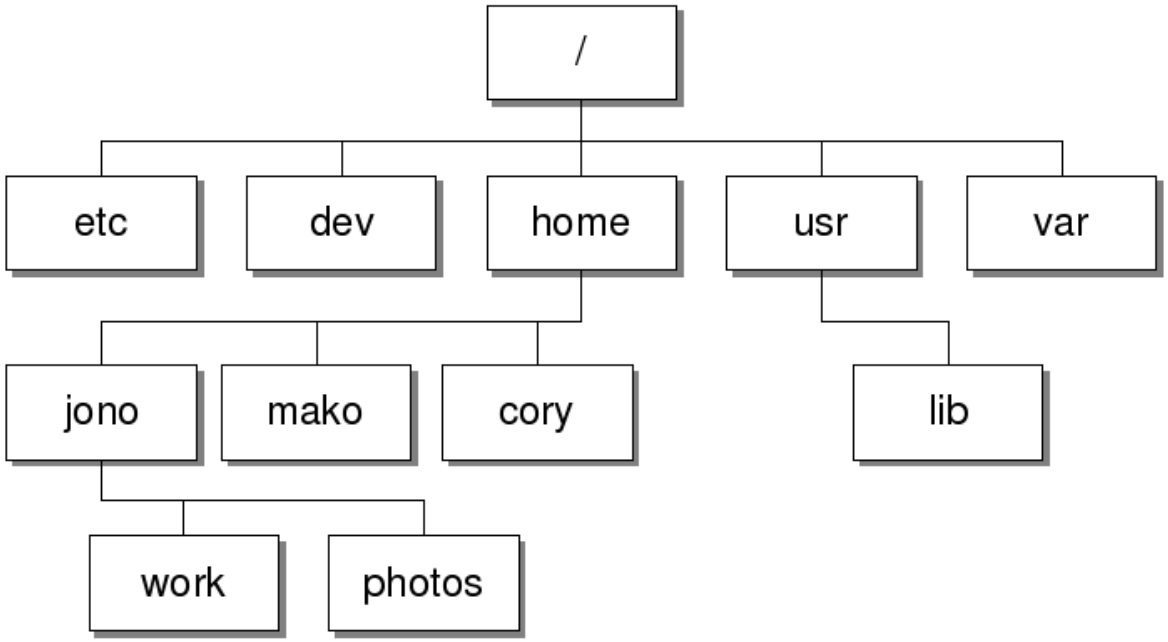
Unix files and structure

UNIX FILES

Unix is made of files!

- Regular files
- Directories
  Directories are just files
  listing a set of files

Filesystem structure



/	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

File names

- Case sensitive
- No length limit
- Can contain any character (including whitespace, except /).
  File types stored in the file.
  File name extensions not needed and not interpreted. Just used for user convenience.
- File name examples:
  README .bashrc
  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/documents/fun/microsoft_jokes.html

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.

CONSOLE

Open the web browser a go to the address

http://ec2-52-209-201-139.eu-west-1.compute.amazonaws.com:3000/connect

Use your credentials to login

Command help

Commands take “arguments” which tell them what exactly to do (most of them start with - or --)

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.

TIPS AND TRICKS

1)The tab key is a shortcut for auto complete your command

2)Unix saves your commands in a history that you can trace back whenever you need (try to type “history” in your terminal and you'll see all last 500-1000 launched commands)

3)using arrows UP and DOWN you can navigate across you history

4)ctrl + a → moves the cursor at the beginning of the line

5)ctrl + e → moves the cursor at the end of the line

TIPS AND TRICKS

6)ctrl + b → moves the cursor one character back

7)ctrl + f → moves the cursor one character forward

8) You can recall a command by its number
!100

9) You can recall the latest command:
!!

Moving between directories

- pwd
  Displays the current directory ("working directory")

- cd <dir>
  Changes the current directory to <dir>

```
$ cd ~/Unix_course/MALTA_2016

$ pwd
```

Moving between directories

- cd ..
  Gets back to the previous directory
- cd ../..
  Gets back to the previous two directories
- cd (cd ~)
  Goes directly to the home directory

```
$ cd ..

$ pwd

$ cd MALTA_2016
```

ls command

Lists the files in the current directory (by default) , in alphanumeric order (if no other option is specified), 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

```
$ ls -ltrh

$ ls -lSrh
```

Wildcards

Commands can use wildcards to perform actions on more than one file at a time, or to find part of a phrase in a text file.

? (question mark)

this can represent any single character.

* (asterisk)

this can represent any number of characters (including zero, in other words, zero or more characters).

```
$ ls -ltrh *txt

$ ls -ltrh  ?.txt
```

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 -r <source_dir> <target_dir> (recursive)
  Copies the whole directory.

```
$ cd ~/Unix_course/

$ cp -r MALTA_2016 MALTA_2016_copy
```

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 -r dir1 dir2 dir3 (recursive)
  Removes the given directories with all their contents.

```
$ cd MALTA_2016_copy

$ ls -ltrh

$ cd ..

$ rm -rf MALTA_2016_copy
```

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).

```
$ mkir TEST1
$ cd TEST1
$ ls
$ cd ..
$ rmdir TEST1
```

The shell has plenty of tools ready to use

Unix tools

Unix tools are robust and fast programs designed with the “one thing well” philosophy

Displaying file contents

Several ways of displaying the contents of files.

- less <file>
  visualize a text file, you can also look for specific words in the file press “q” to exit
- less -S <file>
  does not split the lines
- more <file>
  visualize the file (less interactive).

```
$ cd ~/Unix_course/MALTA_2016

$ less -S 6.txt

$ /

$ 3
```

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)

```
$ cd MALTA_2016

$ head -n1 6.txt

$ tail -n1 6.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 -F “error” .
  Interpret pattern as a set of fixed strings
- grep -v info *.log
  Outputs all the lines in the files except those containing info.

```
$ cd ~/Unix_course/MALTA_2016

$ grep 1 *.txt

$ grep 6 *.txt
```

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.

```
$ cd ~/Unix_course/MALTA_2016

$ sort 6 .txt

$ sort -r 6 .txt
```

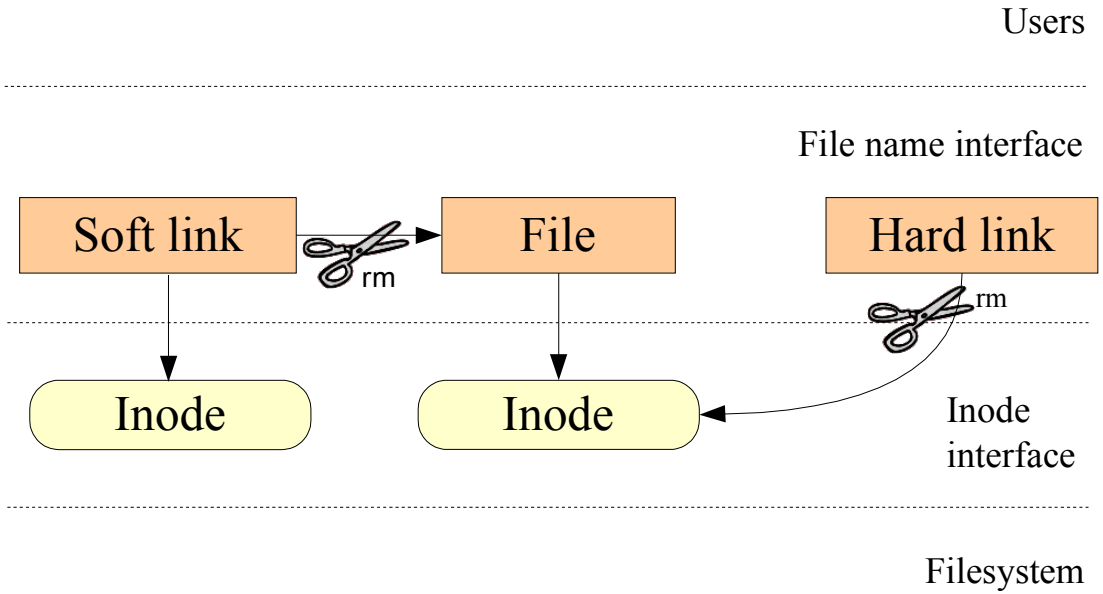
The wc command

- wc -l <file>
  counts the line number in the file.
- wc -w <file>
  counts the word number in the file.
- wc -m <file>
  counts the character number in the file.

```
$ cd ~/Unix_course/MALTA_2016

$ wc -l *txt
```

File 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.
  - 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.

Files names and inodes

```
$ cd ~/Unix_course/MALTA_2016
$ ls -ltrh
$ ln -s 6.txt sl6.txt
$ ls -ltrh
$ ln  6.txt hl6.txt
$ rm  6.txt
$ less  sl6.txt
$ less  hl6.txt
$ mv hl6.txt 6.txt
$ less  sl6.txt
```

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)

awk

- Scripting language on it’s own
- I mostly use it to change values in a certain column of a tab/comma-separated file
- Much more powerful than just that
- Worth digging deeper (not covered here)

awk

- awk ‘<code>’ <files>
  - Conditional printing
    - awk -F”\t” '($1==”yes”) {print $2}' file.txt
  - We can print only the second and third column

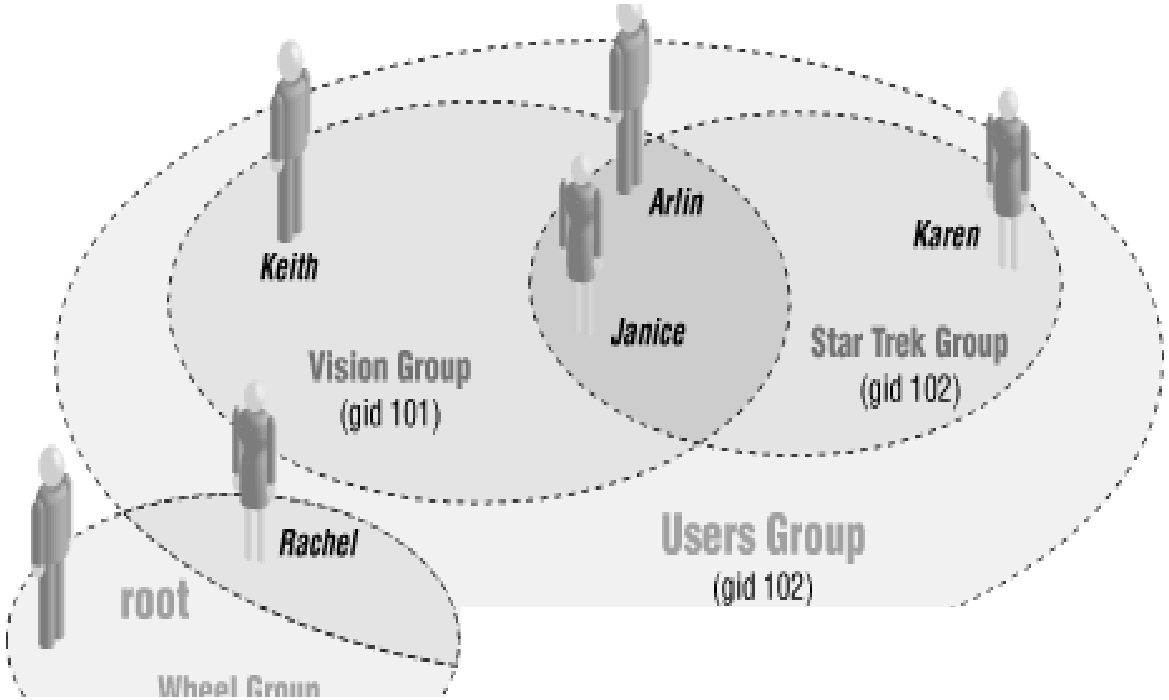
    awk -F”\t” '{print $2,$3}' file.txt
  - We can change change numeric values by

    awk -F"\t" '{$2=$2+1000;print$0}' awk_example.tsv

```
$ cd ~/Unix_course/MALTA_2016
$ less -S awk_example.tsv
$ less -S -x15 awk_example.tsv
$ awk -F"\t" ’($1=="yes") {$2=$2+1000;print$0}’ awk_example.tsv
```

Users and permissions

Users and permissions



Every UNIX user has a user name to define an account

The user is defined by his group and privileges

File access rights

```
$ cd ~/Unix_course/MALTA_2016

$ ls -ltrh
```

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 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:

- 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.

```
$ cd ~/Unix_course/MALTA_2016

$ ls -ltrh

$ chmod g+w awk_example.tsv

$ ls -ltrh

$ chmod g-w awk_example.tsv
```

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...

- Your regular account should be sufficient for 99.9 % of your tasks (unless you are a system administrator).

- 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

Exercise 1

In the directory Unix_course there is a directory named Exercise_1

1) How many files are in it ? Which owner and group do they have ?

2) What are they ? (Visualize the content)

3) Look for the word Malta in each of it

4) How many lines are in each file?

5) How many words are in each file?

6) Create a folder named “Res_1” outside the Exercise_1 path and create in it an hard and a soft link of one file

Coffee break (15 min)

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

  It creates a new file (or replaces it) where saves the output
- Standard output can be appended to an existing file using the >> symbol

  It creates a new file if not already present, otherwise appends the output to the existing file

Standard output

```
$ cd ~/Unix_course/MALTA_2016

$ less 5.txt

$ less 5.txt  > redirect.txt

$ less redirect.txt

$ ls -ltrh

$ less 6.txt >> redirect.txt

$ less redirect.txt

$ ls -ltrh

$ less 1.txt  > redirect.txt

$ less redirect.txt

$ ls -ltrh

$ rm  redirect.txt
```

Concatenate multiple files in one

The cat tool reads files sequentially, writing them to the standard output.

cat <file1> <file 2> <etc.>

```
$ cd ~/Unix_course/MALTA_2016

$ cat 1.txt 2.txt 3.txt >1_2_3.txt

$ less -S 1_2_3.txt
```

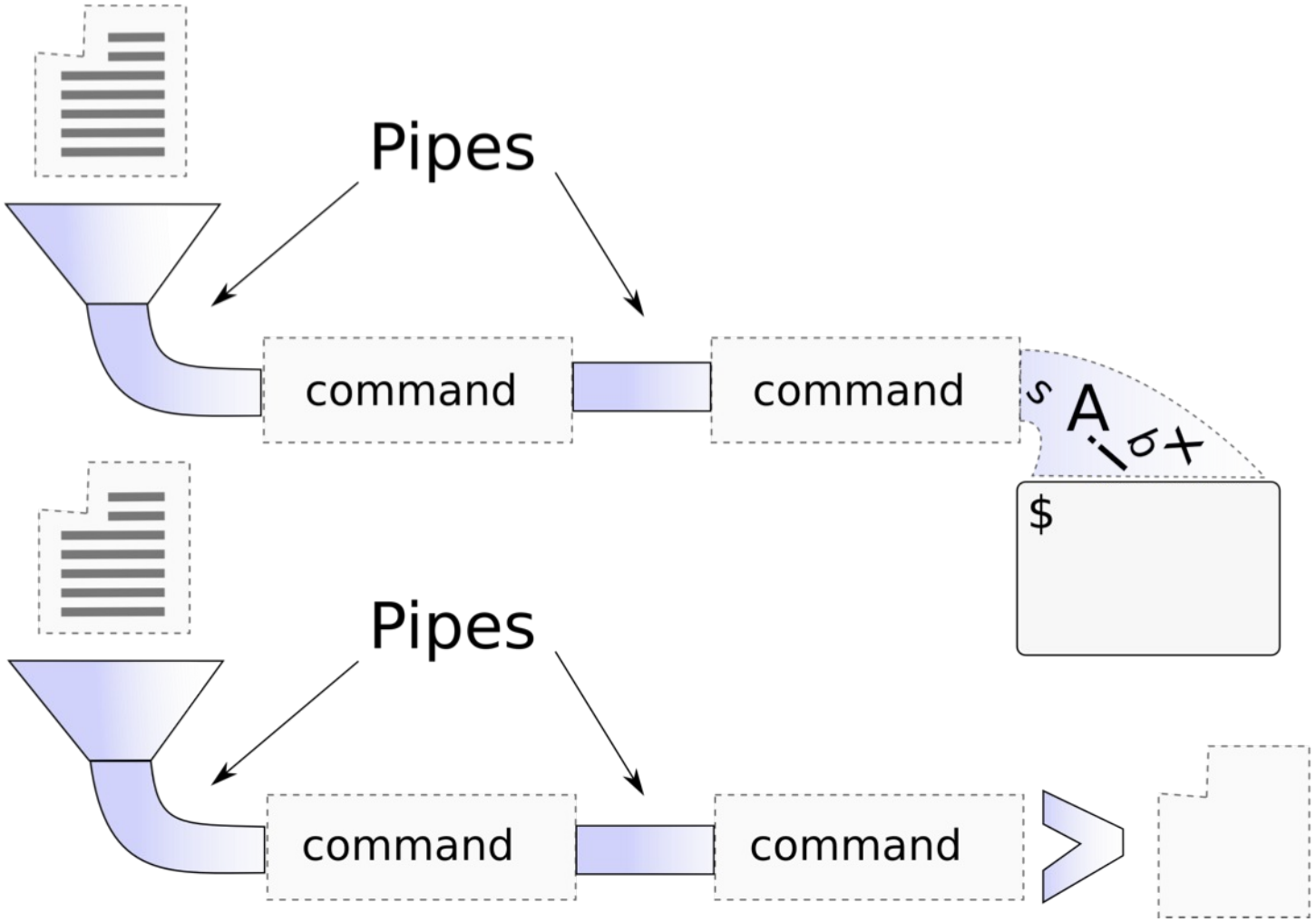
Standard input

More about command input

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



Pipes

- Examples
  - cat *.txt | grep 1 | sort
- This one of the most powerful features in Unix shells!

```
$ cat  ~/Unix_course/Exercise_1/*.txt |grep -F -a "Malta"|sort
```

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

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

- 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...

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_results --interesting --clever --rich &
```

Background job control

- An alternative way to put a job in background is to stop it with [Ctrl] z and than digit bg
- jobs
  Returns the list of background jobs from the same shell
- fg %<n>
  Puts the last / nth background job in foreground mode
- kill %<n>
  Aborts the nth job.

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
- 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.

```
$ top
$ M
$ P
$ q
```

Killing processes

- 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.

- killall [-<signal>] <command>
  Kills all the jobs running <command>. Example: killall bash

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 Goodresults || echo “Sorry, Goodresults don'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).

Exercise 2

In the directory Unix Exercise_2 there is a bash script

1) Which permissions it has? (Can you visualize the content ?)

2) Run it in a normal mode (./kind_job) redirecting its output to a file named ex2_res.out

3) stop it and put in background

4) how can you see that it still working?

5)kill the job

6)visualize the results

7)remove the results

Environment variables

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.

```
$ env
```

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

VISUALIZE A VARIABLE

The **echo** utility writes any specified operands to the standard output.

echo "you are the best unix user" will print to the screen the sentence "you are the best unix user";

You can declare a variable in the shell and print it

```
mytestvar='you'
echo “$mytestvar are the best unix user”
```

At the same manner you can also print system variables as user, shell, etc.

```
$ echo $USER

$ echo $SHELL

$ echo $PRINTER
```

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 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).

~/.bashrc file

- ~/.bashrc

  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.

```
$ less -S ~/.bash_profile
```

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/2.6.9-1.6_FC2
```

Back quotes can be used within double quotes

```
> echo "You are using Linux `uname -r`"
You are using Linux 2.6.9-1.6_FC2
```

Text editors

Graphical text editors

Fine for most needs

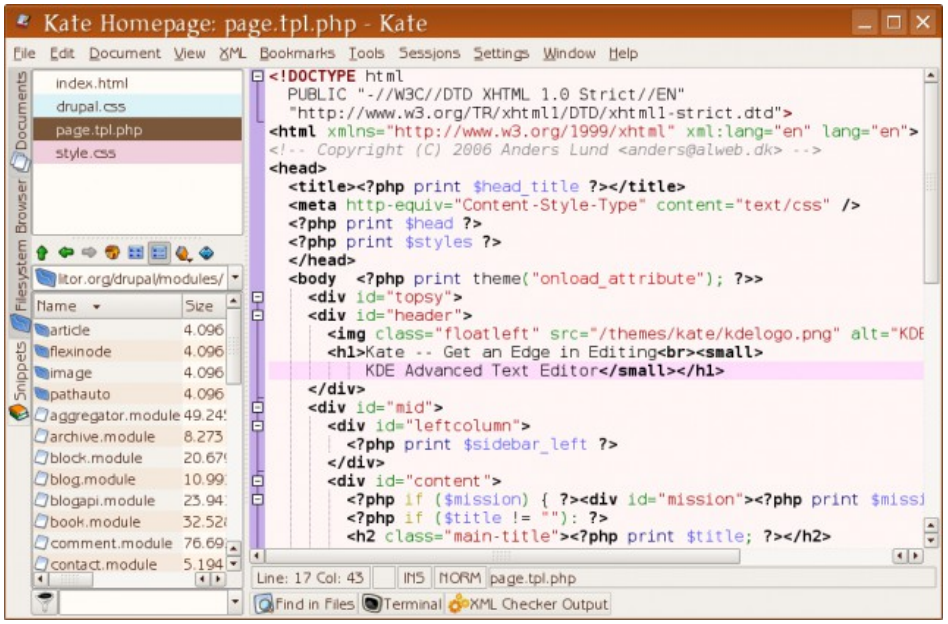
- nedit
- Emacs, Xemacs
- Kate, Gedit

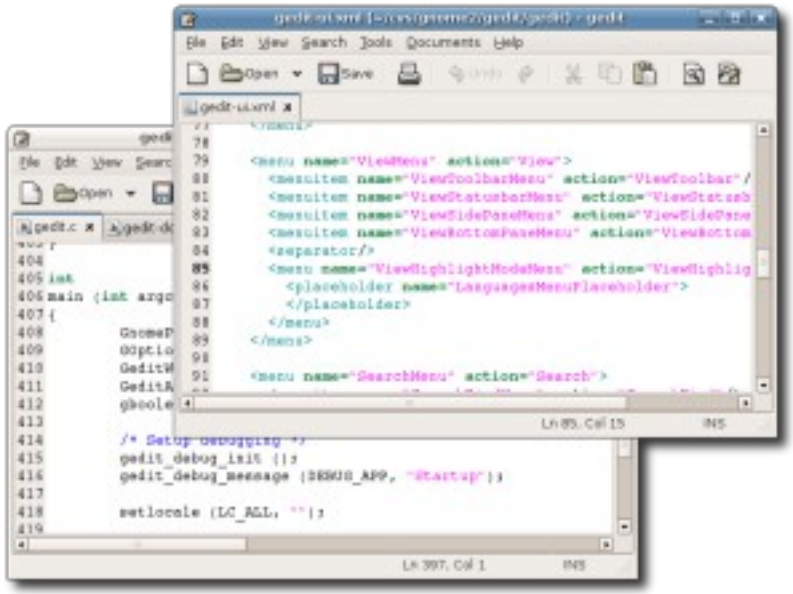
Text-only text editors

Often needed for sysadmins and great for power users

- vi, vim
- nano

Kate and gedit





- Kate is a powerful text editor dedicated to programming activities, for KDE
  - http://kate.kde.org

- Gedit is a text editor for the Gnome environment
  - http://projects.gnome.org/gedit/

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.

It is extremely powerful, its main 30 commands are easy to learn and are sufficient for 99% of everyone's needs!

You can also take the quick tutorial by running vimtutor.

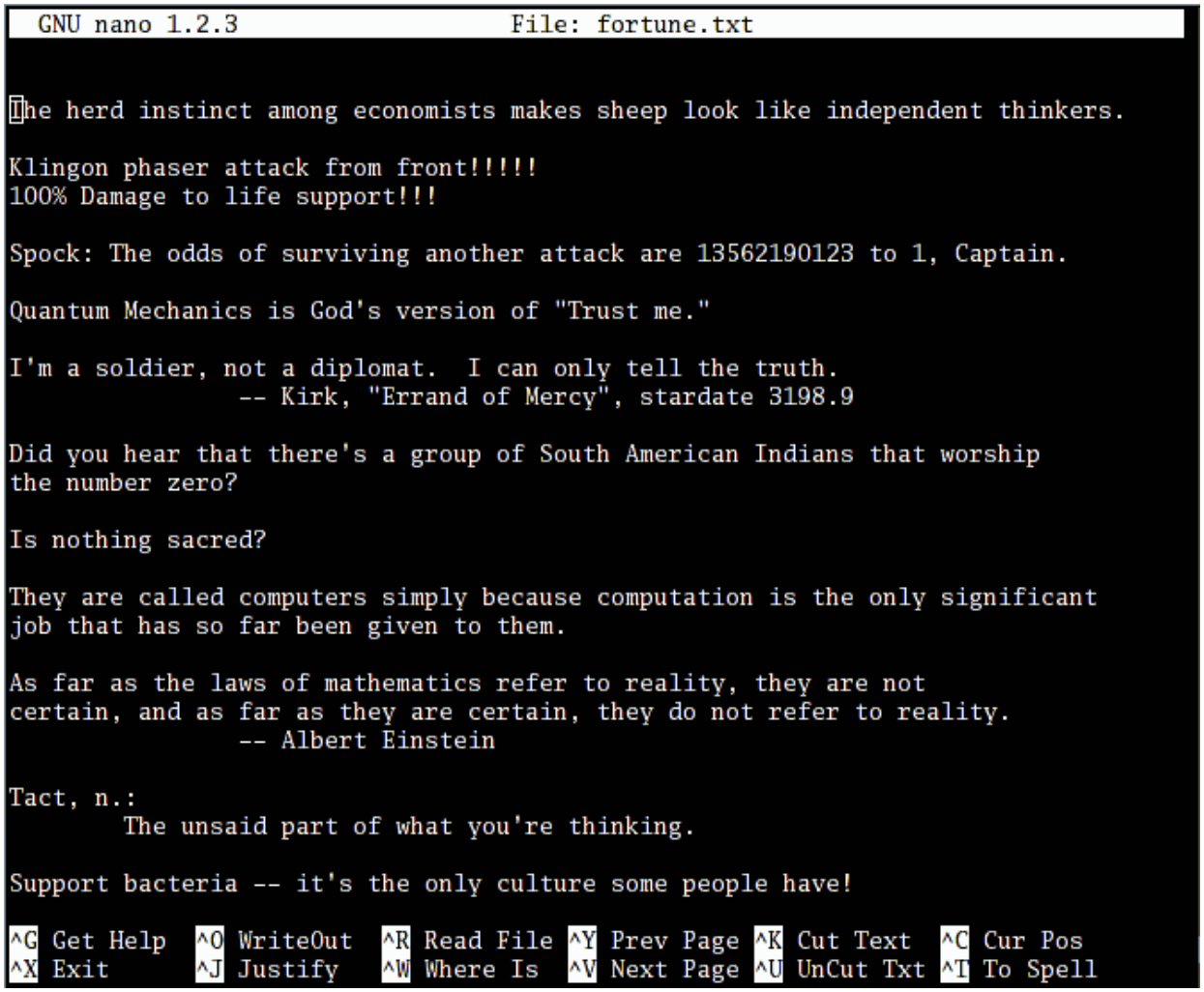
A vi “Cheat Sheet” is present in the handouts folder or online at http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html

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.

GNU nano screenshot



Exercise 3

In the directory Unix Exercise_2 there is a bash script

1) copy it into a different file in the same directory

2) open with nano and edit it add “you are not” before the sentence and remove the sleep command

3) run it redirecting its output to a file in a bakgroundg mode

4) stop it an remove its ouput

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.

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.

Exercise 4

In the directory Exercise_4 there is a the file usconst.txt, it contains the American constitution

1)How many words, lines and bites is made of?

2) Can you count how many times are mentioned the words “Congress” , “State” and c/Citizen ?

3)archive and compress the complete folder

Bonus Exercise

From the gutenberg web page download the 12034-8.txt (http://www.gutenberg.org/files/12034/12034-8.txt)

1)What is it?

2) How many chapter is made of ?

3) How many words is made of ?

4) How many times the word Malta is present in it?

5) Display the content of the 365th line

6) Save in separate files the first 100, the last 150 and all the lines beginning with the word Malta

7)Join all the three files in one and zip it