# Introduction to Unix/Linux

Mihaela Martis

BILS/IKE
Linköping University

August 27, 2015

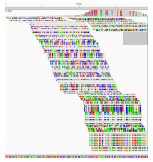
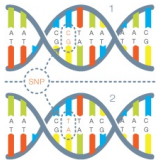
# Goals

- Become familiar with the Unix/Linux operating system.
- Get comfortable with the command-line environment
- Learn powerful commands to process/explore your data.
- Be able to find documentation about individual commands.

# What is Unix?

- is an operating system
- developed in the 1960's by MIT and AT&T Bell Laboratory
- it is fast, stable, secure, multi-user and multi-tasking friendly
- different types of UNIX:
  - Solaris
  - MacOS X
  - Linux (free)



# The operating system

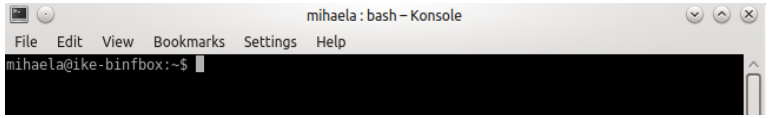
- operating system → is the main software that operates on a computer and that manages the allocation and use of hardware resources
  - kernel → has complete control over everything that occurs in the system and communicates directly with the hardware
  - utilities → small programs used to help manage the system and hardware
  - user interfaces → are wrapper to the kernel that enables users to interact with computers (shell, graphical user interface (GUI))
- e.x. Windows, Linux, OS X

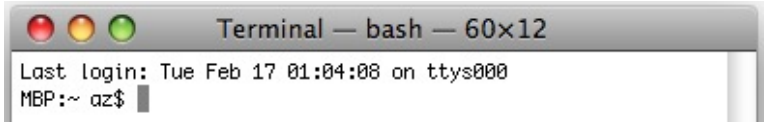
# The shell

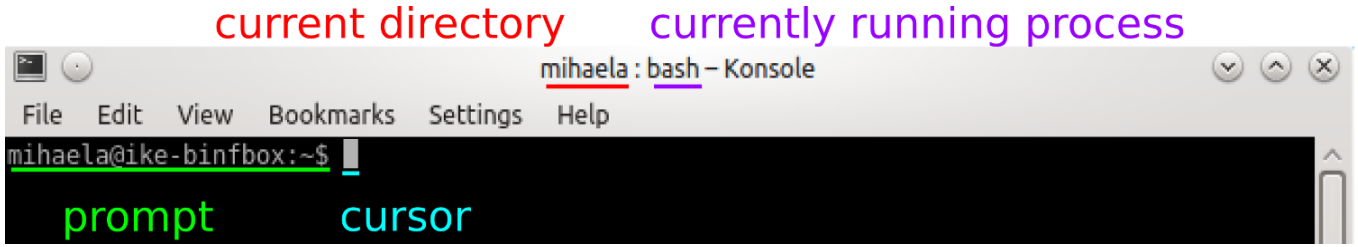
- command → is an instruction given by a human to tell a computer to do something
- shell → is a text-based interface which mediates between the user and the operating systems (command line interpreter).
  - it accept text-based commands, interpret them and invoke the corresponding programs
  - types of shell: bash (Bourne-Again Shell, default on Linux and OS X), csh (C shell), ksh (Korn shell)
  - it runs in a terminal emulator

# The terminal window

- is a text-only window in a GUI that emulates a console
- e.x. in Linux and Mac OS X:



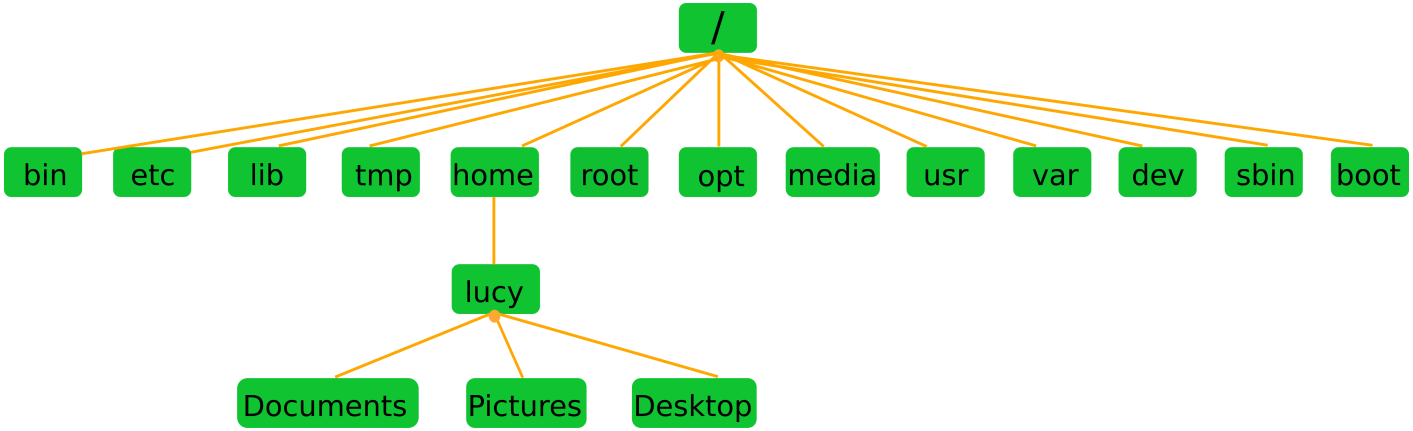



# Files and processes

- everything in UNIX is either a file or a process
- process $\rightarrow$ is an executing program identified by a unique process identifier (PID).
- file $\rightarrow$ is a collection of data created by users using text editors, running programs ...

# The directory structure (I)

- all files are grouped together in the directory structure
- the file-system is arranged in a Hierarchical structure, like an inverted tree
- the top of the hierarchy is called root → displayed as '/' (slash)



# The directory structure (II)

- home → user's home directory
- media → mounted file systems
- boot → kernel and other boot files
- sbin → system administration programs
- bin → common programs
- etc → configuration files
- lib → shared libraries
- tmp → temporary files
- var → log files, dynamic files

# Environmental variables

→ are a class of variables that tell the shell how to behave as the user works at the command line or with shell script
→ use env to list all variables and their values
→ use echo $VARIABLE_NAME to print the content of an individual variable

- HOME → set the home directory
- SHELL → tells where the program that represents your shell is to be found
- PATH → specify a set of directories where executable programs are located

  – the list of directories are separated by colon ':' characters

  – add new directory to the PATH variable:
  PATH=$PATH:/newdirectory

# Paths

- path → is a unique location to a file or a folder
- absolute path → is the list of all directories starting from the root that lead to the current directory

  /var/www/html/
  /home/lucy/Desktop/Pictures/party.png

- relative path → is a way to specify the location of a directory relative to another directory

  Pictures/party.png

- ~ → shortcut to your home directory
- . → the current directory
- .. → the parent of the current directory

# pwd and echo

- *print working directory* – get the absolute path of the working directory:

```
lucy@host:~$ pwd
/home/lucy/Desktop
```

- *echo* – display a line of text/string on standard output or a file

```
lucy@host:~$ echo Hello World!
Hello World!
```

- get the absolute path of your home directory using echo:

```
lucy@host:~$ echo $HOME
/home/lucy
lucy@host:~$ echo ~
/home/lucy
```

# Viewing user and host

- show user name

```
lucy@host:~$ echo $USER
lucy
lucy@host:~$ whoami
lucy
```

- show on which machine you are

```
lucy@host:~$ echo $HOSTNAME
host
lucy@host:~$ hostname
host
```

# ls

- *list* files and directories contained within a specified directory or the current working directory:

```
lucy@host:~$ ls
Desktop  Downloads  Documents
lucy@host:~$ ls Desktop
workflow.pdf  Pictures
```

- include the hidden files as well (.name):

```
lucy@host:~$ ls -a Desktop
.  ..  .directory workflow.pdf Pictures
```

# cd

- *change directory* – move from the current working directory to a new directory

```
lucy@host:~$ cd Desktop
lucy@host:~/Desktop$
lucy@host:~/Desktop$ cd ~/Documents
lucy@host:~/Documents$
```

- move back towards the root in the directory structure

```
lucy@host:~/Desktop$ cd ../Documents
lucy@host:~/Documents$
lucy@host:~/Desktop$ cd ..
lucy@host:~$
```

# Adding and removing directories

- mkdir – **ma**ke **dir**ectory

```
lucy@host:~$ mkdir unix_introduction
lucy@host:~$ ls
Desktop  Downloads  Documents unix_introduction
```

- rmdir – **rem**ove **dir**ectory

```
lucy@host:~$ rmdir unix_introduction
lucy@host:~$ ls
Desktop  Downloads  Documents
```

→ removes only empty directories

→ deleted directories cannot be recovered

# rm

- remove files or directories and their content
- common options:

| | |
|---|---|
| -i | asks for every deletion to be confirmed |
| -r | removes directories |
| -f | force |

```
lucy@host:~/unix_introduction$ ls
first_class myIDs.txt mySeq.fa
lucy@host:~/unix_introduction$ rm myIDs.txt
lucy@host:~/unix_introduction$ ls
first_class mySeq.fa
lucy@host:~/unix_introduction$ rm -r first_class
lucy@host:~/unix_introduction$ ls
mySeq.fa
lucy@host:~/unix_introduction$ rm -i mySeq.fa
rm: remove regular file 'mySeq.fa'?
```

# cp

- copy a file within the same directory to a new file

```
lucy@host:~$ cp original.txt duplicates.txt
lucy@host:~$ ls
duplicates.txt original.txt
```

- copy a file to another directory

```
lucy@host:~$ cp duplicates.txt unix_introduction/
lucy@host:~$ cd unix_introduction/
lucy@host:~/unix_introduction$ ls
duplicates.txt
```

- copy a file from another directory to the current directory

```
lucy@host:~$ cp ~/unix_introduction/duplicates.txt .
lucy@host:~$ ls
duplicates.txt
```

# mv

- move a file to a new location

```
lucy@host:~$ ls
mySeq.fa
lucy@host:~$ mv mySeq.fa ~/unix_introduction/
lucy@host:~$ cd ~/unix_introduction/
lucy@host:~/unix_introduction$ ls
mySeq.fa
```

- rename a file

```
lucy@host:~$ mv mySeq.fa mySeq_renamed.fa
lucy@host:~$ ls
mySeq_renamed.fa
```

# du and touch

- du – show directory size of current directory
  - -s – sum
  - -h – human readable form, in binary units
  - --si – human readable form, in metric units

```
lucy@host:~$ du -sh
4,3M .
lucy@host:~$ du --si
115k    ./tmp
8,2k    ./plots
4,1k    ./test
37k     ./figures/clipart
2,2M    ./figures
4,5M    .
```

- touch – create a new file

```
lucy@host:~$ touch new_file.txt
lucy@host:~$ ls
new_file.txt
```

# less

- display the contents of text files
- loads only the part of a file into memory, which is displaying

```
lucy@host:~$ less mySeq.fa
```

- useful keyboard shortcuts to navigate in *less*

```
q     quit viewing       /abc            search for text 'abc'
space next page          n               find next occurrence of 'abc'
b     back a page        ?               find previous occurrence of 'abc'
3g    go to line 3       h               show help for less
G     go to the end      up/down arrows  move up or down a line
```

# Unix Help

- man (manual page) – view help files for a Unix command at the command line
- press the 'q' key to exit the manual

```
lucy@host:~$ man
What manual page do you want?
lucy@host:~$ man rm
lucy@host:~$ rm --help
```

# Command line shortcuts

- *cut/paste* in a command line environment

```
Ctrl+w (cut last word)
Ctrl+y (paste)
```

- *copy/cut/paste* in a non-command line environment

```
Ctrl+c (copy)
Ctrl+x (cut)
Ctrl+v (paste)
```

- copy text out of the command line and into the desktop

```
Shift+Ctrl+c or Apple+c
```

- paste text from the desktop into the command line

```
Shift+Ctrl+v or Apple+v
```

# Command line shortcuts (II)

- ↑ key – moves back through your previous command history
- ← and → keys – move the cursor back or forth along the current command line
- *tab* key – auto-completion button for the command line
- taking control over the cursor

```
Ctrl+a  -- cursor to beginning of command line
Ctrl+e  -- cursor to end of command line
Ctrl-w  -- cut last word
Ctrl+k  -- cut to the end of the line
Ctrl+y  -- paste content that was cut earlier
```

# Wildcards

- are symbols used to replace of represent one or more characters
- basic wildcards:
  - * – represents zero or more characters
  - ? – represents a single character
  - [] – represents a range of characters
  - \– used as an 'escape' character

```
lucy@host:~$ ls my*
mySeq.fa myIDs.txt
lucy@host:~$ ls *.txt
myIDs.txt
lucy@host:~$ ls ?i*
video.mpeg first.txt
lucy@host:~$ ls *[0-9]*
class_1.txt class_2.txt
```

# Finding files and directories

- search for 'pattern' in and below current directory

```
lucy@host:~$ find -name "*Seq*"
./mySeq.fa
```

- find file names containing the 'pattern' in specified directory

```
lucy@host:~$ find ~/unix_introduction -name "*Seq*"
/home/lucy/unix_introduction/mySeq.fa
```

- find file names containing the 'pattern', but case insensitive

```
lucy@host:~$ find ~/unix_introduction -iname "*seq*"
/home/lucy/unix_introduction/mySeq.fa
```

# Finding applications

- find all files which have been modified in the last two days

```
lucy@host:~$ find ~ -type f -mtime -2
```

- whereis → locate the binary and man page files for a command

```
lucy@host:~$ whereis python
python: /usr/bin/python /usr/bin/python2.7 /usr/bin/python3.4
```

- which → show full path of a command

```
lucy@host:~$ which python
/usr/bin/python
```

# Permissions and ownership

- each file has specific permissions

```
lucy@host:~$ ls -al
drwxrwxr-x 2 lucy family 4096 aug 6 .
drwxrwxr-x 2 lucy family 4096 aug 6 ..
-rw-rw-r-- 1 lucy family  170 aug 6 mySeq.fa
-rw-rw-r-- 1 lucy family    0 aug 6 myIDs.txt
```

- d – directory
- rwx – read write execute
- first triplet – user permissions (u)
- second triplet – group permissions (g)
- third triplet – world permissions (o)

# Change permissions – chmod

- is used to change the permissions of files or directories

```
lucy@host:~$ chmod u=rwx,g=rx,o=r mySeq.fa
lucy@host:~$ ls -l mySeq.fa
-rwxr-xr-- 1 lucy family 170 aug 6 mySeq.fa
```

- use digits to represent the permissions → each digit is a combination of the numbers 4 (read), 2 (write), 1 (execute), and 0 (no permission)

```
lucy@host:~$ chmod 754 mySeq.fa
lucy@host:~$ ls -l mySeq.fa
-rwxr-xr-- 1 andy friends 170 aug 6 mySeq.fa
```

  - 7: 4+2+1 (read, write, execute)
  - 5: 4+0+1 (read, execute)
  - 4: 4+0+0 (read)

# Change ownership – chown, chgrp

- change the owner of a file

```
lucy@host:~$ chown andy mySeq.fa
lucy@host:~$ ls -l mySeq.fa
-rw-rw-r-- 1 andy family 170 aug 6 mySeq.fa
```

- change the group of a file

```
lucy@host:~$ chown :friends mySeq.fa
lucy@host:~$ ls -l mySeq.fa
-rw-rw-r-- 1 andy friends 170 aug 6 mySeq.fa
lucy@host:~$ chgrp pet mySeq.fa
lucy@host:~$ ls -l mySeq.fa
-rw-rw-r-- 1 andy pet 170 aug 6 mySeq.fa
```

- change both owner and the group

```
lucy@host:~$ chown lucy:friends mySeq.fa
lucy@host:~$ ls -l mySeq.fa
-rw-rw-r-- 1 lucy friends 170 aug 6 mySeq.fa
```

# Editing text files

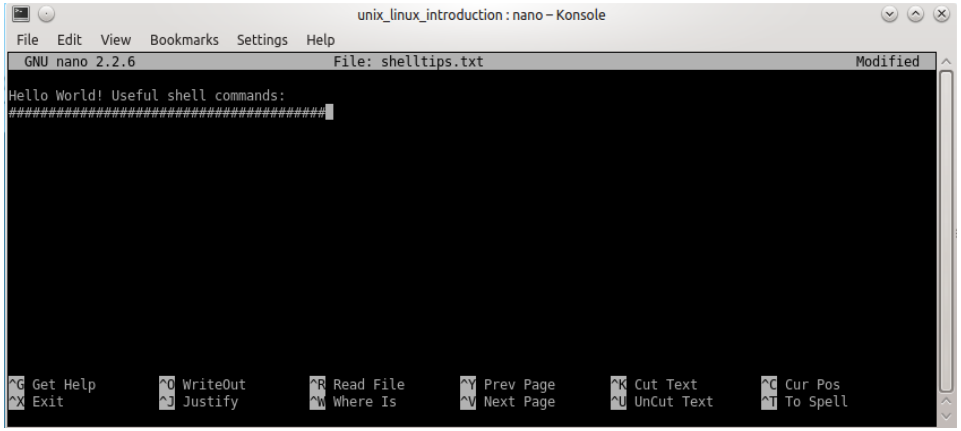
- use text-mode editors to create and modify a file at the command line
- are light-weight, fast, and don't require a lot of overhead to run
- available editors: vi, emacs, gedit, nano
- each has its own set of unique keystrokes for performing similar functions

# nano

- is a keyboard-oriented text editor controlled with control keys
- it displays the options in a menu-like array at the bottom of the screen
- create a new blank file or open an existing one

```
lucy@host:~$ nano
lucy@host:~$ nano shelltips.txt
```

# nano – options




- editor control → hold down the ctrl key while pressing the relevant character:

| | |
|---|---|
| ctrl X | exit nano |
| ctrl Y | scroll up |
| ctrl V | scroll down |
| ctrl O | save the file without closing it |
| ctrl G | get help |

# head and tail

- head – lists the beginning of a file (default: the first 10 lines)
- tail – lists the last few lines of a file (default: the last 10 lines)
- -n – outputs the first/last N lines of the file
- -f – view the last few lines of a growing file, updated continuously

```
lucy@host:~$ head mySeq.fa
lucy@host:~$ head -n 25 mySeq.fa
lucy@host:~$ tail mySeq.fa
lucy@host:~$ tail -n 25 mySeq.fa
lucy@host:~$ tail -f mySeq.fa
```

# file

- reports the type of a file

```
lucy@host:~$ file mySeq.fa
mySeq.fa: ASCII text
lucy@host:~$ file unix_introduction.tex
unix_introduction.tex: LaTeX 2e document, UTF-8 Unicode text
```

# cat

- convenient tool to view the contents of files or join several files
- the output is directly displayed to the screen
- ctrl C – kill cat and get back to the command line

```
lucy@host:~$ cat wishlist1.txt
more money
lucy@host:~$ cat wishlist2.txt
less work
lucy@host:~$ cat wishlist1.txt wishlist2.txt
more money
less work
```

# Input/Output redirection

- most commands read input and write output
- the input is given with the keyboard → stdin, 0
- the output is displayed on the screen → stdout, 1
- error messages are displayed, too → stderr, 2
- use redirection operators to send the output of the commands to files, programs, and even to the input of other commands: »<, >, |

# Standard output

- send the stout to a file using the '>' operator

```
lucy@host:~$ cat wishlist1.txt wishlist2.txt > combined_wishlist.txt
lucy@host:~$ cat wish*.txt > combined_wishlist.txt
lucy@host:~$ cat wishlist.txt
more money
less work
```

- use '»' to append the output to a file

```
lucy@host:~$ cat combined_wishlist.txt
more money
less work
lucy@host:~$ date >> combined_wishlist.txt
lucy@host:~$ cat wishlist.txt
more money
less work
Fre  7 aug 2015 15:07:30 CEST
```

# Standard input

- redirect stdin from a file instead of the keyboard by using '<'

```
lucy@host:~$ cat numbers.txt
20
1
13
2
15
4
3
16
lucy@host:~$ sort -n < numbers.txt
1
2
3
4
13
15
16
20
lucy@host:~$ sort -n < numbers.txt > sorted_numbers.txt
```

# sort

- rearrange the lines in a text file so that they are sorted, numerically and alphabetically
- default sorting rules:
  - lines starting with a number will appear before lines starting with a letter
  - lines starting with a lowercase letter will appear before lines starting with the same letter in uppercase

```
lucy@host:~$ cat toSort.txt
potato
cucumber
ananas
1
Apple
lucy@host:~$ sort toSort.txt
1
ananas
Apple
cucumber
potato
```

# Numerical sorting

- forward and reverse sorting

```
lucy@host:~$ sort -n numbers.txt
1
2
4
8
10
lucy@host:~$ sort -rn numbers.txt
10
8
4
2
1
```

- sort and remove duplicates

```
lucy@host:~$ cat numbers.txt
9
2
3
2
lucy@host:~$ sort -u numbers.txt
2
3
9
```

# Column based sorting

- sort a file on the basis of a column

```
lucy@host:~$ cat age.txt
Fred 22
Mia  30
Jane 8
lucy@host:~$ sort -nk2 age.txt
Jane 8
Fred 22
Mia 30
```

- sort a file based upon more than one column

```
lucy@host:~$ cat employess.txt
Mia  30 2000 1985 apple
Lucy  3    1 2014 banana
Fred 13   15 2002 tomato
lucy@host:~$ sort -nk2,3 -k5 employees.txt
Lucy  3    1 2014 banana
Fred 13   15 2002 tomato
Mia  30 2000 1985 apple
```

# grep – **g**lobal **re**gular expression **p**rint

- a multi-purpose file search tool that uses regular expressions
- extract particular rows from a file

```
lucy@host:~$ cat countries.txt
Sweden, King Carl XVI Gustaf, population: 9,723,809
Denmark, Queen Margrethe II, population: 5,569,077
Ecuador, Rafael Correa, population: 15,654,411
lucy@host:~$ grep "Queen" countries.txt
Denmark, Queen Margrethe II, population: 5,569,077
```

- -i – ignore the case of letters in search matters

```
lucy@host:~$ grep "kiNg" -i countries.txt
Sweden, King Carl XVI Gustaf, population: 9,723,809
```

- -v – return all the lines that don't match the search expression

```
lucy@host:~$ grep "King" -v countries.txt
Denmark, Queen Margrethe II, population: 5,569,077
Ecuador, Rafael Correa, population: 15,654,411
```

# Searching across multiple files with **grep**

```
lucy@host:~$ cat *.seq | grep ">"
>FE_MM1_01A01
>FE_MM1_01A02
>FE_MM1_01A03
>FE_MM1_01A04
>FE_MM1_01A05
>FE_MM1_01A06
```

```
lucy@host:~$ grep ">" *.seq
FEC00001_1.seq:>FE_MM1_01A01
FEC00002_1.seq:>FE_MM1_01A02
FEC00003_1.seq:>FE_MM1_01A03
FEC00004_1.seq:>FE_MM1_01A04
FEC00004_2.seq:>FE_MM1_01A05
FEC00005_1.seq:>FE_MM1_01A06
```

# Further **grep** options

| | |
|---|---|
| -c | show only a count of the results in the file |
| -E | use regular expression syntax with the exception of wildcards |
| -l | list only the file names containing matches |
| -n | show the line numbers of the match |
| -h | hide the file names in the output |

# Pipes

- redirect output from one command to another command with '|'

```
lucy@host:~$ ls -la | less
lucy@host:~$ ls -la | grep wish
-rw-rw-r-- 1 lucy family 11 aug 7 15:18 wishlist1.txt
-rw-rw-r-- 1 lucy family 10 aug 7 15:18 wishlist2.txt
-rw-rw-r-- 1 lucy family 21 aug 7 15:18 wishlist.txt
```

# wc – word count

| options: | |
|---|---|
| -w | gives only the word count |
| -l | gives only the line count |
| -c | gives only the byte count |
| -m | gives only the character count |
| -L | gives only the length of the longest line |

- count how many .txt files are in the current working directory

```
lucy@host:~$ ls *.txt | wc -l
4
```

# Retrieving Web content

- wget – is a tool for non-interactive download of files from the Web (HTTP, HTTPS, FTP)

```
lucy@host:~$ wget "http://www.rcsb.org/pdb/files/1ema.pdb"
--2015-08-14 17:50:59--  http://www.rcsb.org/pdb/files/1ema.pdb
Resolving www.rcsb.org (www.rcsb.org)... 128.6.70.10
Connecting to www.rcsb.org (www.rcsb.org)|128.6.70.10|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: 1ema.pdb.1
    [   <=>  ] 191 403      389KB/s   in 0,5s
2015-08-14 17:50:59 (389 KB/s) - 1ema.pdb.1 saved [191403]
```

- curl – is a tool to transfer data from or to a server (HTTP, HTTPS, FTP, IMAP, POP3 ...)

  – data is downloaded directly to the screen

```
lucy@host:~$ curl "http://www.rcsb.org/pdb/files/1ema.pdb"
% Total % Received % Xferd  Average Speed Time Time Time  Current Dload Upload
      Total Spent Left Speed
0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:--  0HEADER FLUORESCENT PROTEIN  01-AUG
      -96   1EMA
TITLE     GREEN FLUORESCENT PROTEIN FROM AEQUOREA VICTORIA
COMPND    MOL_ID: 1;
...
```

# curl

- save the information to a file

```
lucy@host:~$ curl "http://www.rcsb.org/pdb/files/1ema.pdb" > 1ema.pdb
lucy@host:~$ curl "http://www.rcsb.org/pdb/files/1ema.pdb" -o 1ema.pdb
```

- retrieve a set of files at once

```
lucy@host:~$ curl "http://www.rcsb.org/pdb/files/{1ema,1gfl,1g7k,1xmz}.pdb" -o
     '#1'.pdb
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  186k    0  186k    0     0   258k      0 --:--:-- --:--:-- --:--:--  258k

[2/4]: http://www.rcsb.org/pdb/files/1gfl.pdb --> 1gfl.pdb
100  353k    0  353k    0     0   767k      0 --:--:-- --:--:-- --:--:-- 1035k

[3/4]: http://www.rcsb.org/pdb/files/1g7k.pdb --> 1g7k.pdb
100  691k    0  691k    0     0  1183k      0 --:--:-- --:--:-- --:--:-- 1183k

[4/4]: http://www.rcsb.org/pdb/files/1xmz.pdb --> 1xmz.pdb
100  737k    0  737k    0     0  1540k      0 --:--:-- --:--:-- --:--:-- 2076k
```

# Creating an archive using tar

- create an uncompressed tar archive

```
lucy@host:~$ tar cvf archive_name.tar dirname/
```

- create a tar archive with gzip or bzip2 compression

```
lucy@host:~$ tar czvf archive_name.tar.gz dirname/
lucy@host:~$ tar cvfj archive_name.tar.bz2 dirname/
```

c – create a new archive

v – verbosely list files which are processed

f – following is the archive file name

z – filter the archive through gzip

j – filter the archive through bzip2

# Extracting an archive using tar

- extract an uncompressed tar archive

```
lucy@host:~$ tar xvf archive_name.tar
```

- extract a tar archive with gzip or bzip2 compression

```
lucy@host:~$ tar xzvf archive_name.tar.gz
lucy@host:~$ tar xjvf archive_name.tar.bz2
```

x – extract files from archive

# Listing an archive using tar

- view the tar archive file content without extracting the content

```
lucy@host:~$ tar tvf archive_name.tar
lucy@host:~$ tar tzvf archive_name.tar.gz
lucy@host:~$ tar tjvf archive_name.tar.bz2
```

# Adding a file to an existing archive

- r – add additional files to an existing, uncompressed tar archive

```
lucy@host:~$ tar rvf archive_name.tar new_file.txt
lucy@host:~$ tar rvf archive_name.tar newDir/
```

- estimate the tar archive size (in kb) before you create the tar file

```
lucy@host:~$ tar -cf - path_to_directory/ | wc -c
lucy@host:~$ tar -czf - path_to_directory/ | wc -c
lucy@host:~$ tar -cjf - path_to_directory/ | wc -c
```

# Compress/decompress with gzip and bzip2

- compress a file

```
lucy@host:~$ gzip file_name.txt
lucy@host:~$ bzip2 file_name2.txt
lucy@host:~$ ls
file_name.txt.gz file_name2.txt.bz2
```

- decompress a file

```
lucy@host:~$ gzip -d file_name.txt.gz
lucy@host:~$ bzip2 -d file_name2.txt.bz2
```

# The shell script

- is a text file that contains a sequence of shell commands and which can be invoked as a program
- the script can be saved using the extension '.sh'
- save lots of time by automating tasks
- examples:
  - monitoring your system
  - data backup
  - run various programs sequentially
  - rename a set of files or copy them to a different directory

# How to write shell script

1. open a file

```
lucy@host:~$ nano dir.sh
```

2. tell the operating system which program it should use to interpret the commands

```
#!/bin/bash
```

3. type the commands and save the file

```
ls -la
echo "Above␣are␣the␣directory␣listings␣for␣this␣folder:"
pwd
echo "Right␣now␣it␣is:"
date
```

# How to write shell script (II)

4. give the script execute permissions

```
lucy@host:~$ chmod u+x dir.sh
lucy@host:~$ chmod 755 dir.sh
```

5. execute your script

```
lucy@host:~$ bash dir.sh
lucy@host:~$ sh dir.sh
lucy@host:~$ ./dir.sh
```

# Variables in shell

- system variables → created and maintained by the operating system itself

  – are defined in CAPITAL LETTERS

  – examples: BASH=/bin/bash, HOME=/home/lucy, PATH=/usr/bin:/sbin

- user defined variables (UDV) → created and maintained by the user

  – are defined in lowercase letters

# User defined variables

- syntax → variable_name=value
- rules:
  - variable name must begin with Alphanumeric character or underscore character (_)
  - don't put spaces on either side of the equal sign when assigning value to variable
  - variables are case-sensitive
  - don't use ? or * to name your variables

# How to access UDV

- syntax: $variablename
- print the content of the UDV

```
#!/bin/bash

number=10
vech=bus

echo $vech $number
```

```
lucy@host:~$ ./test.sh
bus 10
```

# Shell arithmetic

- syntax → expr number1 math-operator number2

```
#!/bin/bash

nr1=13
nr2=4

echo "$nr1 + $nr2 ="
expr $nr1 + $nr2

echo "$nr1 * $nr2 = "
expr $nr1 \* $nr2

echo "$nr1 - $nr2 = "
expr $nr1 - $nr2
```

```
lucy@host:~$ ./math.sh
13 + 4 =
17
13 * 4 =
52
13 - 4 =
9
```