Unix Introduction for Bioinformatics

Genomics Workshop, Pretoria Univeristy

Lieven Sterck
lieven.sterck@psb.vib-ugent.be

2014

Outline

Introduction to the Unix operating system

Introduction to the command line

Unix file system

Unix file utilities

Unix text files processing

Output streams, Unix jobs & processes control

Unix shell

Optional Exercises

Introduction

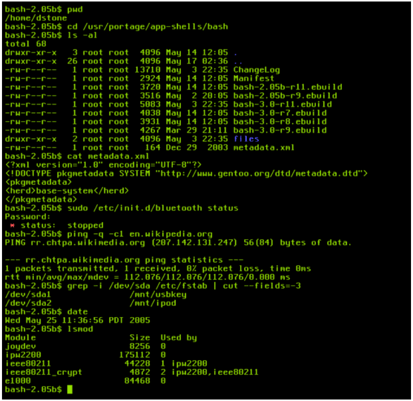
Unix is an operating system (**OS**) that was developed at Bell Labs in 1969.

- **Unix** : **UN**iplexed **I**nformation and **C**omputing **S**ystem
- **Operating systems** coordinate activities in a computer and act as interfaces between users (or programs) and the different hardware components.
- OS examples: **Unix**, **Mac OS**, **Windows**, **Linux**, etc.
- Most computer systems like desktop computers, supercomputers, handheld computers and even video game consoles are using an **OS** of some type.

Unix features: system

- Unix is a **multi-tasking** system: multiple tasks (processes) share common processing resources, like the processor.
- Unix is a **multi-user** system: several users can connect to and use the system, *at the same time*.
- Unix is a **time-sharing** system: the processing power is always shared between applications, so that several tasks can be run at the same time.

Unix features: the command line



```
bash-2.05b$ pwd
/home/dstone
bash-2.05b$ cd /usr/portage/app-shells/bash
bash-2.05b$ ls -al
total 68
drwxr-xr-x   3 root root  4096 May 14 12:05 .
drwxr-xr-x  26 root root  4096 May 17 02:36 ..
-rw-r--r--   1 root root 13710 May  3 22:35 ChangeLog
-rw-r--r--   1 root root  2924 May 14 12:05 Manifest
-rw-r--r--   1 root root  3720 May 14 12:05 bash-2.05b-r11.ebuild
-rw-r--r--   1 root root  3516 May  2 20:05 bash-2.05b-r9.ebuild
-rw-r--r--   1 root root  5083 May  3 22:35 bash-3.0-r11.ebuild
-rw-r--r--   1 root root  4038 May 14 12:05 bash-3.0-r7.ebuild
-rw-r--r--   1 root root  3931 May 14 12:05 bash-3.0-r8.ebuild
-rw-r--r--   1 root root  4267 Mar 29 21:11 bash-3.0-r9.ebuild
drwxr-xr-x   2 root root  4096 May  3 22:35 files
-rw-r--r--   1 root root   164 Dec 29  2003 metadata.xml
bash-2.05b$ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
<pkgmetadata>
<herd>base-system</herd>
</pkgmetadata>
bash-2.05b$ sudo /etc/init.d/bluetooth status
Password:
 * status:  stopped
bash-2.05b$ ping -q -c1 en.wikipedia.org
PING rr.chtpa.wikimedia.org (207.142.131.247) 56(84) bytes of data.

--- rr.chtpa.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 112.076/112.076/112.076/0.000 ms
bash-2.05b$ grep -i /dev/sda /etc/fstab | cut --fields=-3
/dev/sda1               /mnt/usbkey
/dev/sda2               /mnt/ipod
bash-2.05b$ date
Wed May 25 11:36:56 PDT 2005
bash-2.05b$ lsmod
Module                  Size  Used by
joydev                  8256  0
ipw2200               175112  0
ieee80211              44228  1 ipw2200
ieee80211_crypt         4872  2 ipw2200,ieee80211
e1000                  84468  0
bash-2.05b$ █
```

- **Mechanism** to interact with a computer by typing commands to perform specific tasks.
- A **command line interpreter** will analyze the command and its arguments and launch the actions required to perform the task.
- Various command line interpreters are available on Unix systems (**shells**).

Command line advantages and disadvantages

Advantages

- **Simplicity**: simple concept, quite fast learning curve.
- **Flexibility**: easy to combine several commands and create workflows.
- **Powerful**: easy to run programs on remote computers.
- **Batch**: easy to apply a series of commands several times.
- **Fast** development: creating new workflows, combining new programs is quite fast.

Disadvantages

- Less **intuitive** than graphical user interface.
- **Unforgiving**: commands have to be typed with the correct syntax.
- Text based interface, not easy to deal with **graphics**.
- To be efficient, a couple of commands and options have to be **memorized**.
- Command names and options can be **cryptic**.

Unix and bioinformatics

…or **why** are Unix systems so popular in bioinformatics circles?

- Building **workflows** is a typical bioinformatics task: ex. parsing the output of a BLAST search.
- Most of the sequence analysis is dealing with **text-related** data.
- Default Unix programs are well designed and can handle **large amounts** of data.
- By default, multiple **development** tools are available on Unix systems (scripting languages, compilers, shell, etc.).
- Unix computers have robust network features and can be quite easily **clustered** together.
- Unix-like systems are very stable and usually have a very long **uptime**.
- Most Linux distributions are available **free of charge**.

Unix and NGS

. . . or **why** are Unix systems your best choice to process NGS data?

- NGS data files can be really **huge**.
  (eg. the data file(s) associated with the single lane of an NGS machine can often total over 5GB)
- Most of the computation for NGS analysis is going to be done on a **remote computer system**, not the PC/laptop in front of you.
- Many laptops are not that powerful or that fast, compared to server-style computers
- Most NGS research tools are **built for UNIX** operating systems and do not have a graphical interface.

Outline

Introduction to the Unix operating system

Introduction to the command line

Unix file system

Unix file utilities

Unix text files processing

Output streams, Unix jobs & processes control

Unix shell

Optional Exercises

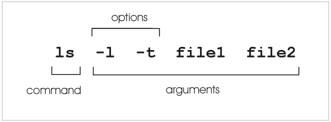
Logging in to a remote server

Connecting to a remote server is done using the SSH-protocol.
PuTTY is a free ssh-client. It's a very small program and does not require any installation. Just download it and save it on your PC.
You will have to configure your connection to the server and the server has to allow remote access!

If you want to connect to a different server, you can simply use the **ssh** protocol. To do so just type "ssh <servername>".

The command line

Typical structure of the Unix command:



options

```
ls  -l  -t  file1  file2
```

command

arguments

- **Whitespaces** separate the command from the options and arguments.
- The number of whitespaces does not matter, but there must be at least one (ls-l does **not** work).
- Commands and arguments are **case-sensitive** (ls $\neq$ LS $\neq$ Ls $\neq$ lS).
- Depending on the command, options and files can be optional.
- The order of the options is in most cases not important (ls -a -l = ls -l -a).
- Hit **[enter]** and the command will be executed.
- Options are prefixed with the - character, followed by **one** letter.
- Long names options are prefixed with a double dash, followed by a **long** name:

  ```
  ls --help
  ```

Command line tricks

Every modern shell has some built-in utilities to edit command lines efficiently.

- Magic **[Tab]** completion: hit [Tab] and the system will try to complete file and directory names.
- **Arrow** keys: Move in the history of previously used commands.
- **[Ctrl]** - **e** Move to the end of the line.
- **[Ctrl]** - **a** Move to the beginning of the line.
- **[Ctrl]** - **l** Clear the screen (same as the command **clear**).
- **[Ctrl]** - **u** Clear the line from the cursor to the beginning of the line.

Your first Unix command!

The **echo** command just displays a message on the screen.

```
[liste@goblin ~]$ echo Hello world!
Hello world!
```

Examples: whoami, pwd, date, hostname

In case you're lost, you can ask the system **who you are** (!):

```
[liste@goblin ~]$ whoami
liste
```

... where you are with the **pwd** command ...

```
[liste@goblin ~]$ pwd
/home/liste
```

... or get the current **date and time**:

```
[liste@goblin ~]$ date
Fri Sep 12 17:37:32 CEST 2008
```

... or to **which server** you are connected:

```
[liste@goblin ~]$ hostname
goblin.psb.ugent.be
```

The first “useful” command: ls

The **ls** command is used to list files and directories.

- The command can be used to list a specific file

```
[liste@goblin ~]$ ls /bin/sleep
/bin/sleep
[liste@goblin ~]$ ls /bin/slop
ls: /bin/slop: No such file or directory
```

- If a directory is given as the argument, the complete list of files within the directory will be returned.

```
[liste@goblin ~]$ ls /bin/
alsaunmute  csh            ex        ksh       mv             rview      tracepath6
arch        cut            false     link      netstat        sed        traceroute
ash         date           fgrep     ln        nice           setfont    traceroute6
ash.static  dd             gawk      loadkeys  nisdomainname  setserial  true
awk         df             gettext   login     pgawk          sh         umount
basename    dmesg          grep      ls        ping           sleep      uname
bash        dnsdomainname  gtar      mail      ping6          sort       unicode_start
bsh         doexec         gunzip    mailx     ps             stty       unicode_stop
cat         domainname     gzip      mkdir     pwd            su         unlink
chgrp       dumpkeys       hostname  mknod     red            sync       usleep
chmod       echo           igawk     mktemp    rm             tar        vi
chown       ed             ipcalc    more      rmdir          tcsh       view
cp          egrep          kbd_mode  mount     rpm            touch      ypdomainname
cpio        env            kill      mt        rvi            tracepath  zcat
```

More about ls

The **ls** command has many, many options.

- Wildcards characters ***** and **?** can be used to substitute for any other character(s) in a file or directory name.

  ```
  [liste@goblin ~]$ ls /bin/l*
  /bin/link  /bin/ln  /bin/loadkeys  /bin/login  /bin/ls
  [liste@goblin ~]$ ls /bin/?s
  /bin/ls  /bin/ps
  ```

- The -**l** options is giving details about the files (permissions, links, owner, size, date of last change, name).

  ```
  [liste@goblin ~]$ ls -l /bin/sleep
  -rwxr-xr-x  1 root root 22040 May 29 15:09 /bin/sleep
  ```

- The size is expressed in bytes, but it's possible to have it in a more human readable format with the -**h** option (e.g., 1K 234M 2G).

  ```
  [liste@goblin ~]$ ls -lh /bin/sleep
  -rwxr-xr-x  1 root root 22K May 29 15:09 /bin/sleep
  ```

More about ls

- On Unix systems, file names beginning with a "." are hidden, but the -**a** option will show them.

```
[liste@goblin ~]$ ls -a
.              bin.txt      .gconfd          .kde      .Xclients
..             .cache       .gnome           .local    .Xclients-default
.bash_history  .config      .gnome2          .mozilla  .xemacs
.bash_logout   .emacs       .gnome2_private  tmp
.bash_profile  .fullcircle  .gtkrc           toto
.bashrc        .gconf       .ICEauthority    .viminfo
```

- The -**d** option will list only directories, not files.

```
[liste@goblin ~]$ ls -ld /usr/bin/
drwxr-xr-x  2 root root 69632 Sep 15 04:02 /usr/bin/
[liste@goblin ~]$ ls -ld /usr/
drwxr-xr-x  17 root root 4096 Sep  9 09:31 /usr/
```

Command options

Several options can be **combined** with only one - sign (exceptions)

```
ls -l -a -t
ls -lat
```

A wrong option will generate an **error** message.

```
[liste@goblin ~]$ ls -z
ls: invalid option -- z
Try ‘ls --help’ for more information.
```

Redirection

The default output channel for commands is the **screen** (standard output) but this can easily be changed to create a file.

```
ls /usr/bin > bin.txt
```

The content of a text file can be visualized using the cat command.

```
cat bin.txt
```

A text pager program, like **more** or **less**, might be useful if the file is long. Hit **[enter]** to move one line, or **[space]** to move a screen at a time.

```
more bin.txt
```

The less pager fills the biggest lack of the more command: no way to return back. Use the **arrow keys** to move up and down with less.

```
less bin.txt
```

It is also possible to change the default input channel (keyboard) to read from a file, while at the same time redirecting the output to another file.

```
cat < bin.txt > bin2
```

Unix pipes

The **standard output** (default is the screen) of any Unix command can be redirected to the **standard input** (default is the keyboard) of any other command using the pipe | character.

```
ls /usr/bin | less
```

Combining multiple commands to create a **workflow** is a piece of cake.

```
ls /usr/bin | sort | less
```

It is perfectly possible to redirect the **output** of a workflow to a file.

```
ls /usr/bin | sort > bin_sort.txt
```

The **wc -l** command can be used to count the number of lines in a file.

```
[liste@goblin ~]$ wc -l bin_sort.txt
2150 bin_sort.txt
```

The count can also be done **directly** using a pipe.

```
[liste@goblin ~]$ ls /usr/bin | wc -l
2150
```

Help: commands built-in help

Most of the Unix commands have some sort of built-in help. The help is usually displayed by invoking the -**h** or **–help** option.

```
[liste@goblin ~]$ man -h
man, version 1.5o1

usage: man [-adfhktwW] [section] [-M path] [-P pager] [-S list]
        [-m system] [-p string] name ...

  a : find all matching entries
  c : do not use cat file
  d : print gobs of debugging information
  D : as for -d, but also display the pages
  f : same as whatis(1)
  h : print this help message
  k : same as apropos(1)
```

Help: commands built-in help

```
[liste@goblin ~]$ who --help
Usage: who [OPTION]... [ FILE | ARG1 ARG2 ]

  -a, --all         same as -b -d --login -p -r -t -T -u
  -b, --boot        time of last system boot
  -d, --dead        print dead processes
  -H, --heading     print line of column headings
  -i, --idle        add idle time as HOURS:MINUTES, . or old
                    (deprecated, use -u)
  -l, --login       print system login processes
      --lookup      attempt to canonicalize hostnames via DNS
  -m                only hostname and user associated with stdin
```

Help: the man pages

Unix documentation is available through the **man** pages.

```
man ls
```

- Some man pages are split into several pages. You can navigate between the pages using the **arrow** keys; move one screen up with the **f** key (or spacebar, or Page Up) or back with the **b** (or Page Down) key. Exit with the **q** key.
- Search can be done interactively by typing the **frontslash** character, followed by the keyword of interest.
- To search for the man pages corresponding to a **keyword** is done with the -**k** option:

```
[liste@goblin ~]$ man -k gcc
gcc                  (1)  - GNU project C and C++ compiler
gcc [g++]            (1)  - GNU project C and C++ compiler
gccmakedep           (1x)  - create dependencies in makefiles using 'gcc -M'
```

Outline

Introduction to the Unix operating system

Introduction to the command line

Unix file system

Unix file utilities

Unix text files processing

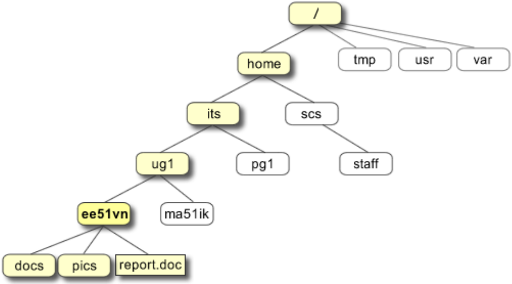
Output streams, Unix jobs & processes control

Unix shell

Optional Exercises

A hierarchical file system

All files in Unix are organized in a hierarchical system than can be viewed as an inverted tree.



Path names

- Path names are **absolute** when the complete path starting from the root is given. Those paths are by definition **unique**.

```
[liste@goblin ~]$ ls /usr/share/man/man1/perlfaq1.1.gz
/usr/share/man/man1/perlfaq1.1.gz
```

- **Relative** paths indicate the location by reference to the **current** directory. They are by definition **not unique**.
- Referring to the **parent** directory with the **double-dot**:

```
[liste@goblin share]$ pwd
/usr/share
[liste@goblin share]$ ls ../
bin  etc    home     kerberos  lib64    local  share  tmp
doc  games  include  lib       libexec  sbin   src    X11R6
[liste@goblin share]$ ls /usr
bin  etc    home     kerberos  lib64    local  share  tmp
doc  games  include  lib       libexec  sbin   src    X11R6
```

Path names

- Referring to paths **under** the current directory with the **simple-dot**.

```
[liste@goblin share]$ pwd
/usr/share
[liste@goblin share]$ ls ./gnome
cursor-fonts     default.wm  help   vfolders  wm-properties
default.session  gkb         panel  vino
[liste@goblin share]$ ls gnome
cursor-fonts     default.wm  help   vfolders  wm-properties
default.session  gkb         panel  vino
[liste@goblin share]$ ls /usr/share/gnome
cursor-fonts     default.wm  help   vfolders  wm-properties
default.session  gkb         panel  vino
```

File names

- Most linux systems will allow file names up until 255 characters (or even more). Some other systems (DOS, old windows versions, ...) have a shorter limit. Keep this in mind when you want to share files.

```
[liste@goblin share]$ stat -f /home/ | grep -i name
[liste@goblin share]$ ID: 0        Namelen: 255     Type: autofs
```

- Almost all characters (both upper- and lower-case) can be used to create file names but some are preferentially avoided:

```
| ; , ! @ # $ ( ) < > / \ " ’ ‘ ˜ { } [ ] = + & ˆ
```

- In terms of what would be good in a unix environment:
  - a-z
  - A-Z
  - 0-9
  - underscore (_)
  - dash (-)
  - period (.)
- **CAUTION**: Spaces can be okay, but make things difficult. Windows users love them, unix/linux don't!!

File names

Some tips for naming files (and directories):

- A filename must be unique inside its directory
- Make use of the long filename possibility to create meaningful names
- Try to reserve the . (dot) for denoting the file extension
- use - or _ to separate logical words (eg. my_first_file.txt)
- Be consistent. Pick a style, and stick with it

Navigating the file system

- We have seen that the **pwd** command is used to get the absolute path to your current location.
- The **cd** (change directory) command is used to move from one place to another:

  ```
  [liste@goblin ~]$ pwd
  /home/student1
  [liste@goblin ~]$ cd /usr/share
  [liste@goblin share]$ pwd
  /usr/share
  ```

- Of course, **relative** path names can also be used with the cd command.

  ```
  [liste@goblin ~]$ pwd
  /home/student1
  [liste@goblin ~]$ cd ..
  [liste@goblin share]$ pwd
  /home
  [liste@goblin share]$ cd student2
  [liste@goblin ~]$ pwd
  /home/student2
  ```

- To switch between two directories (== to go back to the directory you where previously):

  ```
  [liste@goblin share]$ cd -
  ```

Going home

- The **cd** command without any argument will bring you home.

  ```
  [liste@goblin share]$ pwd
  /usr/share
  [liste@goblin share]$ cd
  [liste@goblin ~]$ pwd
  /home/student1
  ```

- Alternatively you can use ”~” . The ~-sign denotes a home directory and it can be used as a shortcut.

  ```
  [liste@goblin share]$ cd ~
  ```

  will take you to your own home dir. To go to someone else's home dir:

  ```
  [liste@goblin share]$ cd ~yvpee
  ```

- You can also use it to construct pathnames.

  ```
  [liste@goblin share]$ cd ~/bin
  [liste@goblin share]$ pwd
  [liste@goblin share]$ /home/liste/bin
  ```

Creating directories

The command **mkdir** (make directory) is used to create one or more new directories.

```
[liste@goblin ~]$ mkdir new
[liste@goblin ~]$ ls
bin_sort.txt  list  new  new.txt  tmp  toto
```

Multiple directories can be created with **one** command:

```
[liste@goblin ~]$ mkdir new1 new2 new3
[liste@goblin ~]$ ls
bin_sort.txt  list  new  new1  new2  new3  new.txt  tmp  toto
```

It is possible to create a directory tree directly using the -**p** option:

```
[liste@goblin ~]$ mkdir -p tree/new
```

This creates both the directory **tree** and its subdirectory **new**

Removing directories

- The **rmdir** command can be used to remove empty directories:

```
  [liste@goblin ~]$ ls
bin_sort.txt  list  new  new1  new2  new3  new.txt  tmp  toto  tree
[liste@goblin ~]$ ls new
[liste@goblin ~]$ rmdir new
[liste@goblin ~]$ ls
bin_sort.txt  list  new1  new2  new3  new.txt  tmp  toto  tree
```

- To remove **non-empty** directories, it is possible to use the command rm (remove files) with the **recursive** option. This command will remove anything under the given path, so it has to be used **cautiously**.

```
  [liste@goblin ~]$ rm tmp
[liste@goblin ~]$ ls
bin_sort.txt  list  new1  new2  new3  new.txt  toto  tree
[liste@goblin ~]$ ls tree
one  two
[liste@goblin ~]$ rm -r tree
[liste@goblin ~]$ ls
bin_sort.txt  list  new1  new2  new3  new.txt  toto
[liste@goblin ~]$
```

Copying files

The **cp** command is used to copy files in various ways. For example it can be used to create a backup copy of a file.

```
  [liste@goblin ~]$ ls
bin_sort.txt  list  new1  new2  new3  new.txt  toto
[liste@goblin ~]$ cp bin_sort.txt bin_sort.txt.bak
[liste@goblin ~]$ ls
bin_sort.txt  bin_sort.txt.bak  list  new1  new2  new3  new.txt  toto
```

The **dot** notation can be used to copy a file to the current directory.

```
[liste@goblin ~]$ cp /bin/ls .
[liste@goblin ~]$ ls
bin_sort.txt  bin_sort.txt.bak  list  ls  new1  new2  new3  new.txt  toto
```

As with any Unix command, **wildcards** character can be used to match several files.

```
[liste@goblin ~]$ mkdir progs
[liste@goblin ~]$ cp /bin/l* progs/.
[liste@goblin ~]$ ls progs/
link  ln  loadkeys  login  ls
```

Copying files

The -**R** option (recursive) can be used to copy entire directory trees:

```
[liste@goblin ~]$ ls -R /etc/joe
/etc/joe:
charmaps  jmacsrc  joerc  jpicorc  jstarrc  rjoerc  syntax

/etc/joe/charmaps:
klingon

/etc/joe/syntax:
asm.jsf   csh.jsf      html.jsf  mail.jsf    perl.jsf    sh.jsf       vhdl.jsf
c.jsf     diff.jsf     java.jsf  mason.jsf   php.jsf     tcl.jsf      xml.jsf
conf.jsf  fortran.jsf  lisp.jsf  pascal.jsf  python.jsf  verilog.jsf
[liste@goblin ~]$ cp -R /etc/joe/ .
[liste@goblin ~]$ ls -R joe/
joe/:
charmaps  jmacsrc  joerc  jpicorc  jstarrc  rjoerc  syntax

joe/charmaps:
klingon
```

Moving and renaming files

The **mv** command renames a file or a directory:

```
  [liste@goblin ~]$ ls
bin_sort.txt  bin_sort.txt.bak  joe  list  new1  new2  new3  newbin  new.txt  p
[liste@goblin ~]$ mv bin_sort.txt.bak bin_sort.old
[liste@goblin ~]$ ls
bin_sort.old  bin_sort.txt  joe  list  new1  new2  new3  newbin  new.txt  progs
```

As with the copy command, **multiple** files can be moved at the same time, but to a **directory** only.

```
  [liste@goblin ~]$ ls
bin_sort.old  bin_sort.txt  list  new1  new2  new3  newbin  new.txt
[liste@goblin ~]$ mv bin_sort.* new1/
[liste@goblin ~]$ ls new1/
bin_sort.old  bin_sort.txt
```

The **mv** command can also be used to rename directories:

```
  [liste@goblin ~]$ ls
list  new1  new2  new3  newbin  new.txt
[liste@goblin ~]$ mv new1 other
[liste@goblin ~]$ ls
list  new2  new3  newbin  new.txt  other
```

How much disk space am I using / free space do I have?

- To check how much disk space is occupied, use the **du** cmd (disk usage)

  ```
  [liste@goblin ~]$ du -h
  [liste@goblin ~]$ du -hS
  ```

- With the **df** cmd, you get an overview of the free disk space (for the whole file system)

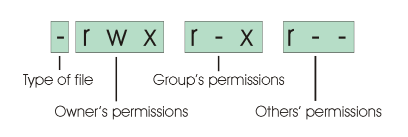
  ```
  [liste@goblin ~]$ df
  ```

File permissions

The Unix system is protecting files using a **three-component** permission system. Files permissions can be seen using the **ls -l** command.

```
  [liste@goblin ~]$ ls -l
total 36
-rw-r--r--  1 student1 class_liste 15923 Sep 16 16:12 list
drwxr-xr-x  2 student1 class_liste  4096 Sep 22 18:13 new2
drwxr-xr-x  2 student1 class_liste  4096 Sep 22 18:13 new3
drwxr-xr-x  3 student1 class_liste  4096 Sep 23 14:10 newbin
-rw-r--r--  1 student1 class_liste     9 Sep 22 18:06 new.txt
drwxr-xr-x  2 student1 class_liste  4096 Sep 23 15:28 other
```

The first block of data is a representation of the permissions.



- r w x r - x r - -

Type of file

Owner's permissions

Group's permissions

Others' permissions

File permissions: examples

Permissions are defined using three slots representing:

- read permission (**r**): specifies if a file can be read
- write permission (**w**): specifies if a file can be modified, overwritten or deleted.
- execute permission (**x**): specifies if a file can be **executed**, meaning it is a script or compiled program.

These three slots are specified for the for:

- **owner**: the owner of the file
- **group**: the group to which the owner belongs
- (the rest of the) **world**: everyone else with access to the system

```
  [liste@goblin ~]$ echo test > file
[liste@goblin ~]$ ls -l file
-rw-r--r--  1 student1 class_liste 5 Sep 23 17:21 file
[liste@goblin ~]$ ls -l /bin/ls
-rwxr-xr-x  1 root root 82688 May 29 15:09 /bin/ls
```

Changing file permissions

The command **chmod** is used to alter file permissions.
The general syntax is:

```
chmod <category u|g|o|a> <+|-> <r(read), w(write), x(execute)> <file>
```

Some examples:

```
  [liste@goblin ~]$ ls -l file
-rw-r--r--  1 student1 class_liste 5 Sep 23 17:21 file
[liste@goblin ~]$ chmod g+w file
[liste@goblin ~]$ ls -l file
-rw-rw-r--  1 student1 class_liste 5 Sep 23 17:21 file
[liste@goblin ~]$ chmod o+w file
-rw-rw-rw-  1 student1 class_liste 5 Sep 23 17:21 file
[liste@goblin ~]$ chmod o-w,g-w file
[liste@goblin ~]$ ls -l file
-rw-r--r--  1 student1 class_liste 5 Sep 23 17:21 file
[liste@goblin ~]$ chmod o+rwx file
[liste@goblin ~]$ ls -l file
-rw-rw-rwx  1 student1 class_liste 5 Sep 23 17:21 file
[liste@goblin ~]$ chmod a+rwx file
[liste@goblin ~]$ ls -l file
-rwxrwxrwx  1 student1 class_liste 5 Sep 25 11:51 file
```

Changing file permissions: absolute assignment

Permissions can be assigned in an absolute manner using a combination of numbers that represent a particular combination of the three basic rights.
The **chmod** command can use a three-digit string to represent this combination.

```
Read:    4 (100)
Write:   2 (010)
Execute: 1 (001)
```

Each category permission can now be represented by the **sum** of the desired permissions.

```
rw-r--r-- <=> (4+2)(4)(4) = 644
```

```
  [liste@goblin ~]$ ls -l ls
-rwxr-xr-x  1 student1 class_liste 82688 Sep 23 17:15 ls
[liste@goblin ~]$ chmod 644 ls
[liste@goblin ~]$ ls -l ls
-rw-r--r--  1 student1 class_liste 82688 Sep 23 17:15 ls
[liste@goblin ~]$ chmod 755 ls
[liste@goblin ~]$ ls -l ls
-rwxr-xr-x  1 student1 class_liste 82688 Sep 23 17:15 ls
```

Directories permissions

Directories permissions **meaning** is slightly different from the files.

- **Read**: enable or restrict the **listing** of the content of the directory.
- **Write**: permission to **create** or **remove** files in that directory.
- **Execute**: executing a directory is of course nonsense. Execution permission in this case is the ability to "pass through" the directory when searching for a subdirectory. For example, the **cd** command won't work if the "execute" permission is off.

The ability to **create** and to **delete** a file depends on the **directory's permissions**, not on the permissions of the file itself. For example, it can be possible to modify a file (write permissions) but not to delete it.

Outline

Introduction to the Unix operating system

Introduction to the command line

Unix file system

Unix file utilities

Unix text files processing

Output streams, Unix jobs & processes control

Unix shell

Optional Exercises

Symbolic links

The **ln -s** command is creating a symbolic link to a file or a directory. You can then use the link to open, modify or use the file. Can be very useful in bioinformatics to create links to **big** data files. It is also used to create access to the same file from different access points.

```
  ln -s <source> <target or linkname>

[liste@goblin ~]$ ln -s /bin/ls myls
[liste@goblin ~]$ ls -l myls
lrwxrwxrwx  1 student1 class_erbon 7 Oct  1 16:23 myls -> /bin/ls
[liste@goblin ~]$ ./myls
find_copy  list  myls  new2  new3  newbin  new.txt  other  progs  test.txt

[liste@goblin ~]$ echo test > file
[liste@goblin ~]$ cd progs/
[liste@goblin progs]$ ln -s ../file .
[liste@goblin progs]$ ls -l file
lrwxrwxrwx  1 student1 class_erbon 7 Oct  1 16:28 file -> ../file
[liste@goblin progs]$ cat file
test
```

When a link is 'broken' eg. when the source file has been deleted or moved, you will see it with a red background and blinking white font.

Compressing files

File compression is very often used to send data or to save some disk space. It is of course very useful in bioinformatics because of the potential large size of data files (genomes, annotations, etc.).
Several programs are available to compress and uncompress files. The most popular ones are **gzip** and **bzip2**.

gzip
Is older and less efficient than bzip2, but faster. It is frequently used to compress files to temporarily save storage space or to ship them around (mail, download, ...)
When using gzip on a file it will rename the file, appending .gz to the input file name and the input file will be removed.

```
 gzip [options] file

-c       output to standard out (as a result will also keep the original file)
-d       decompress
-v       be verbose
-f       force overwrite of output file and compress links
-1 .. -9       compression rate (set block size to 100k .. 900k)
--fast         alias for -1
--best         alias for -9
```

Compressing files

bzip2
Bzip2 compresses better than gzip but at a slower rate. Therefore it is often used to compress files with the purpose of archiving them (there is no need to regularly access them).
The options are very similar to the ones of gzip. To keep the orignal file you can use the option "-k" .

If no filename is provided both tools will read from STDIN and output to STDOUT, making it possible to use them in a workflow using the "pipe"-operator.

Compressing files, examples

```
[liste@goblin ~]$ ls -lh bigfile
 -rwxr-xr-x  1 student1 students 13M Oct 24 13:11 bigfile

[liste@goblin ~]$ gzip bigfile

[liste@goblin ~]$ ls -lh bigfile.gz
 -rwxr-xr-x  1 student1 students 3.8M Oct 24 13:11 bigfile.gz
ezor ~]$ gunzip bigfile.gz

[liste@goblin ~]$ ls -lh
-rwxr-xr-x  1 student1 students  13M Oct 24 13:11 bigfile

[liste@goblin ~]$ gzip -v bigfile
bigfile   58.2% -- replaced with bigfile.gz
[liste@goblin ~]$ gunzip -c bigfile.gz > non_zipped_file

[liste@goblin ~]$ ls -lh
-rwxr-xr-x  1 student1 students 3.8M Oct 24 13:11 bigfile.gz
```

Decompressing files

To decompress a gzipped file, you have 3 options:
gzip -d file (gzipped file will be removed)
gunzip file (gzipped file will be removed)
zcat (will keep the original gzipped file)

Similarly for bzipped files:
bunzip2 -d file
bunzip2 file
bzcat file

ATTENTION: do not confuse gzip (and gunzip) with zip and unzip!!
The latter two are used to compress and decompress zip-archive files, which is the windows equivalent of the tar.gz files.

Decompressing files, examples

```
[liste@goblin ~]$ gunzip -c bigfile.gz | gzip - > bigfile2.gz

[liste@goblin ~]$ ls -lh bigfile*
-rw-r--r--  1 student1 students 3.8M Oct 24 14:06 bigfile2.gz
-rwxr-xr-x  1 student1 students 3.8M Oct 24 13:11 bigfile.gz

[liste@goblin ~]$ gunzip bigfile2.gz

[liste@goblin ~]$ bzip2 bigfile
bzip2: Can’t open input file bigfile: No such file or directory.
[liste@goblin ~]$ bzip2 bigfile2
[liste@goblin ~]$ ls -lh bigfile*
-rw-r--r--  1 student1 students 3.5M Oct 24 14:06 bigfile2.bz2
-rwxr-xr-x  1 student1 students 3.8M Oct 24 13:11 bigfile.gz

[liste@goblin ~]$ bunzip2 bigfile2.bz2
```

Creating archives with tar

The **tar** command is used to create archives of files and directories. Although its primary use was for backup purposes (tape archive), it is frequently used to exchange data and programs in the Unix world.
Entire directory trees can be archived using **tar**. A tar-archive is also often called a ”tarball”.

```
tar [options] <archive name> <files-to-include>

Key options:
-c create an archive
-x extract from an archive
-t display files within an archive, without actually extracting them

Some additional options are frequently used in combination with the key optio
-f specify the name of the tar file
-v display the progress (verbose)
-z tar and compress with gzip
-j tar and compress with bzip2
```

Creating and extracting data from an archive

```
[liste@goblin ~]$ tar cvf test.tar *
file
find_copy/
find_copy/lower3.hpp
find_copy/local_time_adjustor.hpp
...
[liste@goblin ~]$ ls -lh test.tar
-rw-r--r--  1 student1 biocomp 7.3M Oct  1 17:43 test.tar
[liste@goblin ~]$ mkdir test_tar
[liste@goblin ~]$ cd test_tar
[liste@goblin test_tar]$ tar xvf ../test.tar
file
find_copy/
find_copy/lower3.hpp
find_copy/local_time_adjustor.hpp
...
[liste@goblin test_tar]$ ls
file  find_copy  list  new2  new3  newbin  new.txt  other  progs  test.tar test.txt  tmp
```

Content of an archive

```
[liste@goblin test_tar]$ ls
file  find_copy  list  new2  new3  newbin  new.txt  other  progs
test.txt  tmp
[liste@goblin test_tar]$ pwd
/usr/home/student1/test_tar
[liste@goblin test_tar]$ rm -rf *
[liste@goblin test_tar]$ ls
[liste@goblin test_tar]$
[liste@goblin test_tar]$ tar tvf ../test.tar | more
-rw-r--r-- student1/class_erbon 11 2008-10-01 16:43:01 file
drwxr-xr-x student1/class_erbon  0 2008-10-01 15:43:50 find_copy/
-rw-r--r-- student1/class_erbon 4079 2008-10-01 15:51:32 find_copy/lower3.hpp
-rw-r--r-- student1/class_erbon 8153 2008-10-01 15:51:32 find_copy/local_time_adjustor.hpp
-rw-r--r-- student1/class_erbon 6477 2008-10-01 15:51:32 find_copy/lambda_no_ctps.hpp
...
[liste@goblin test_tar]$ ls
[liste@goblin test_tar]$
```

Mixing tar and gzip

tar archives are often compressed to save space.

```
[liste@goblin ~]$ ls -l test.tar
-rw-r--r--  1 student1 students 7618560 Oct  1 17:43 test.tar
[liste@goblin ~]$ gzip test.tar

[liste@goblin ~]$ ls -l test.tar.gz
-rw-r--r--  1 student1 students 3074165 Oct  1 17:43 test.tar.gz


[liste@goblin ~]$ gunzip test.tar.gz
[liste@goblin ~]$ tar xvf test.tar

[liste@goblin ~]$ tar czvf test.tar.gz *

[liste@goblin ~]$ ls -l test.tar.gz
-rw-r--r--  1 student1 students 7037380 Oct 24 14:21 test.tar.gz

[liste@goblin ~]$ file test.tar.gz
test.tar.gz: gzip compressed data, from Unix

[liste@goblin ~]$ mkdir tmp
[liste@goblin ~]$ cd tmp
[liste@goblin tmp]$ tar xzvf ../test.tar.gz
```

Converting Unix text files to DOS and vice-versa

```
[liste@goblin ~]$ od -bc new.txt
0000000 156 145 167 040 146 151 154 145 012
          n   e   w       f   i   l   e  \n

[liste@goblin ~]$ unix2dos new.txt
unix2dos: converting file new.txt to DOS format ...

[liste@goblin ~]$ od -bc new.txt
0000000 156 145 167 040 146 151 154 145 015 012
          n   e   w       f   i   l   e  \r  \n

[liste@goblin ~]$ dos2unix new.txt
dos2unix: converting file new.txt to UNIX format ...

[liste@goblin ~]$ od -bc new.txt
0000000 156 145 167 040 146 151 154 145 012
          n   e   w       f   i   l   e  \n
```

Outline

Introduction to the Unix operating system

Introduction to the command line

Unix file system

Unix file utilities

Unix text files processing

Output streams, Unix jobs & processes control

Unix shell

Optional Exercises

Display the beginning of a file

The **head** utility is used to display the first lines of a file. You can use it to quickly check what exactly is in a file without having to open the complete file.
By default **head** will show the first 10 lines of a file.
If no file is provided it will read standard input.

You can change the number of lines that are printed by using the -**n** option, specifying the number of lines you want to print

- **head** -**n3** print first 3 lines
- **head** -**3** print first 3 lines (specifying -n is not obligatory)
- **head** -**n**-**3** print all except the last 3 lines of the file

Display the end of a file

The **tail** utility is similar to 'head' but displays the last lines of a file. It's often used to for instance check if a file is complete.
By default **tail** will show the last 10 lines of a file.
If no file is provided it will read standard input.

Just like with 'head' you can change the number of lines that are printed by using the **-n** option.

- **tail -n3** print last 3 lines
- **tail -n+3** print all lines starting from the 3th line in the file.

Use the **-f** option of **tail** to monitor lines as they are added to the end of a growing file (eg. a log file).
tail -f log

Sorting data

The **sort** command is a powerful command to sort lines of text files.
There are many useful options:

- -**n** sort according to the numerical value.
- -**r** sort in reverse order.
- -**u** sort and remove duplicated lines.
- -**k** sort according to one or more keys.

Removing duplicate lines: the **uniq** command

The **uniq** command prints the unique lines in a sorted file, retaining only one of a run of matching lines. It is frequently used with **sort** since it compares only consecutive lines.

The most commenly used options:

- **-u** Only print unique lines.
- **-d** Only print duplicate lines.
- **-c** Print the number of times each line occurred along with the line.

Uniq examples

```
[liste@goblin uniq_examples]$ cat sort3
two
two
two
three
one
one
[liste@goblin uniq_examples]$ uniq sort3
two
three
one
[liste@goblin uniq_examples]$ uniq -d sort3
two
one
[liste@goblin uniq_examples]$ uniq -u sort3
three
[liste@goblin uniq_examples]$ uniq -c sort3
3 two
1 three
2 one
```

Select columns: the **cut** command

The **cut** command is used to select columns from a tab-delimited text file.

- -f : specifies the columns you want
- -d: specifies the delineator (if not tab)

```
[liste@goblin sort_examples]$ cut -f 1 sort5
[liste@goblin sort_examples]$ cut -f 1,2 sort5
[liste@goblin sort_examples]$ cut -d ';' -f 1,2 sort5b
```

Joining lines: the **paste** command

The **paste** command is used to join files horizontally (parallel). It will output lines consisting of elements from each of the input files separated by tabs.
Paste will first read all the input files and only then start printing lines.

It only has two often used options:

- **-d** a single or a list of characters to be used in stead of tab.
- **-s** Paste horizontally. Process one file at a time.

Select lines: the **grep** command

The **grep** command is used to print lines matching a pattern.

```
grep [options] PATTERN [FILE...]
```

```
[liste@goblin sort_examples]$ cat sort5
20      sequence1       org2
3       sequence2       org3
5       sequence1       org3
6       sequence0       org3
356     sequence5       org1
356     sequence1       org1
3       sequence7       org2
50      sequence9       org2

[liste@goblin sort_examples]$ grep sequence1 sort5
20      sequence1       org2
5       sequence1       org3
356     sequence1       org1
```

grep useful options

```
-c print the number of lines that match the pattern
-i ignore case
-n print line numbers
-v print all lines except the ones matching the pattern
```

```
[liste@goblin sort_examples]$ grep -n sequence1 sort5
1:20    sequence1       org2
3:5     sequence1       org3
6:356   sequence1       org1

[liste@goblin sort_examples]$ grep -c sequence1 sort5
3

[liste@goblin sort_examples]$ grep -i SEQUENCE1 sort5
20      sequence1       org2
5       sequence1       org3
356     sequence1       org1

[liste@goblin sort_examples]$ grep -v sequence1 sort5
3       sequence2       org3
6       sequence0       org3
356     sequence5       org1
3       sequence7       org2
50      sequence9       org2
```

Basic regular expressions with **grep**

Sophisticated patterns can be build using **regular expressions**
Patterns are build using a set of characters that have a special meaning.

```
^   beginning of the line
$   end of the line
.   any character
*   zero or more occurence of a character
[ ] any group of character that match the chararcters between brackets
 (character class)
```

Basic regular expressions examples

```
[liste@goblin sort_examples]$ cat bio-seq
accaccatgc
gcgatgcttttt
aaattatattgg
tttcagaaacgcggtcgct
tttcagtaacgcggtcgct

[liste@goblin sort_examples]$ grep atgc bio-seq
accaccatgc
gcgatgcttttt

[liste@goblin sort_examples]$ grep -n atgc bio-seq
1:accaccatgc
2:gcgatgcttttt

[liste@goblin sort_examples]$ grep -c atgc bio-seq
2
```

Basic regular expressions examples

```
[liste@goblin sort_examples]$ grep ’ttt...aaa’ bio-seq
tttcagaaacgcggtcgct

[liste@goblin sort_examples]$ grep ’ttt...[at]aa’ bio-seq
tttcagaaacgcggtcgct
tttcagtaacgcggtcgct
```

Note: These examples are simply an illustration of regular expressions. **Never** use grep to find biological motifs in a sequence, as they might be split into two different lines!
Note2: The regular expressions capabilities of grep can be extended with the -P option. This will allow PERL-style reg-ex to be used. eg.

```
grep -P "\t\d+" <file>
```

.

Outline

Introduction to the Unix operating system

Introduction to the command line

Unix file system

Unix file utilities

Unix text files processing

Output streams, Unix jobs & processes control

Unix shell

Optional Exercises

File descriptors

On Unix, there are **3 standard character streams** associated to each command or process:

- **Standard input [0]**: file or stream representing the input data, by default the keyboard.
- **Standard output [1]**: file or stream representing the output data, by default the screen.
- **Standard error [2]**: file or stream representing the error messages, by default the screen.

Any Unix command is reading from and writing to these files.

Character replacing

With the **tr** utility it's possible to replace or delete characters from a stream.

```
Usage: tr [options] 'string1' ['string2']
```

The most commonly used options:

- **-d** delete the listed of character(s).
- **-s** "squeeze repeats": replace multiple occurences of a character with a single one.

Managing processes

Every command or program under Unix is attached to a **process**. There can be thousands of processes running simultaneously on a Unix system.

The command **top** can be used to display the running processes on a given system.

```
top - 11:39:18 up 13 days, 10:35, 22 users,  load average: 0.12, 0.14, 0.33
Tasks: 329 total,   1 running, 323 sleeping,   0 stopped,   5 zombie
Cpu(s):  0.1% us,  0.1% sy,  0.0% ni, 99.7% id,  0.0% wa,  0.0% hi,  0.0% si
Mem:  16384864k total,   608928k used, 15775936k free,    39748k buffers
Swap:  4096564k total,   218892k used,  3877672k free,    64828k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
14963 student1  16   0  8488 1304  832 R  1.0  0.0   0:00.35 top
24639 nanao     15   0  174m  22m 9260 S  0.7  0.1   0:36.96 gnome-terminal
 6188 root      15   0 33668 1804 1312 S  0.3  0.0  97:31.05 X
    1 root      16   0  4772  448  416 S  0.0  0.0   0:01.47 init
    2 root      RT   0     0    0    0 S  0.0  0.0   0:00.45 migration/0
    3 root      34  19     0    0    0 S  0.0  0.0   0:01.23 ksoftirqd/0
```

top is an interactive command, use the ”q” (or ctrl-C) key to exit.

Listing processes

The command **ps** lists processes. Without options, it only lists the processes of the current user.

```
[liste@goblin ~]$ ps
  PID TTY  TIME CMD
31110 pts/22 00:00:00 tcsh
31201 pts/22 00:00:00 ps
```

The **u** option displays the owner (user) of each process

```
[liste@goblin ~]$ ps u
USER     PID %CPU %MEM VSZ  RSS TTY STAT START   TIME COMMAND
liste  31110  0.2  0.0 54844 1644 pts/22 Ss   16:30   0:00 -tcsh
liste  31340  0.0  0.0  7524  724 pts/22 R+   16:33   0:00 ps u
```

The **x** option also lists process not started in a terminal.

```
[liste@goblin ~]$ ps ux
USER     PID %CPU %MEM VSZ  RSS TTY STAT START   TIME COMMAND
liste  31109  0.0  0.0 44876 2012 ?   S    16:30   0:00 sshd: liste@pts/22
liste  31110  0.0  0.0 54848 1652 pts/22 Ss   16:30   0:00 -tcsh
liste  32101  0.0  0.0  7524  724 pts/22 R+   16:43   0:00 ps ux
```

Stopping processes

The command **kill** can be used to terminate a process.

```
[liste@goblin ~]$ ps aux | grep student1
root     16524  0.0  0.0 44864 3556 ?        Ss   12:09   0:00 sshd: student1 [priv]
student1 16544  0.0  0.0 44864 2072 ?        S    12:10   0:00 sshd: liste@pts/9
student1 16545  0.0  0.0 57092 1544 pts/9    Ss   12:10   0:00 -bash
student1 16721  0.0  0.0  7516  744 pts/9    R+   12:13   0:00 ps aux
student1 16722  0.0  0.0 51084  620 pts/9    S+   12:13   0:00 grep student1

[liste@goblin ~]$ kill 16524
-bash: kill: (16524) - Operation not permitted

[liste@goblin ~]$ kill 16544
Connection to razor.fvms.ugent.be closed by remote host.
Connection to razor.fvms.ugent.be closed.
```

Jobs and job control

Jobs can be **moved** between the foreground and the background with specific commands.

```
fg   Brings jobs to the foreground
bg   Moves jobs to the background
[Ctrl-Z]  Suspends the current foreground job
jobs Lists active jobs
kill Kill jobs
```

Those commands are only valid for the **current** shell session.

Outline

- Introduction to the Unix operating system
- Introduction to the command line
- Unix file system
- Unix file utilities
- Unix text files processing
- Output streams, Unix jobs & processes control
- Unix shell
- Optional Exercises

The Unix shell

The Unix shell is both an **interpreter** and a **scripting** language.

The shell behavior is determined by a set of **environment** variables, for example the **SHELL** or the **PATH** variables. Applications often obtain information about the process environment from those variables.

```
[liste@goblin ~]$ echo $SHELL
/bin/bash

[liste@goblin ~]$ echo $PATH
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/usr/home/student
[liste@goblin ~]$
```

By convention, environment variables are in **uppercase**.

Shell history

The **history** command is listing previous commands, even those used in previous sessions.

```
[liste@goblin ~]$ history | tail
  997  cat .bashrc
  998  vi .bashrc
  999  exit
 1000  which blastall
 1001  cat .bashrc
 1002  ls
 1003  more new.txt
 1004  ls -lh bigfile.gz
 1005  history
 1006  history | tail

[liste@goblin ~]$ history | grep blastall
  832  nohup /usr/local/blast/bin/blastall -p blastp -i prot1.fa
  -d /blastdb/shared/prot > res &
```

Alternatively, one can browse through previous used commands by using the ”up” and ”down” arrow keys.

Aliases

An **alias** is a shortcut to a command or to a set of commands.

```
[liste@goblin ~]$ alias
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-t

[liste@goblin ~]$ alias b=/usr/local/blast/bin/blastall

[liste@goblin ~]$ b

blastall 2.2.18   arguments:

  -p  Program Name [String]
  -d  Database [String]
    default = nr
  -i  Query File [File In]
    default = stdin
  -e  Expectation value (E) [Real]
...
```

Aliases can be made **permanent** using the **.cshrc** file.

Edit files

On a Unix system there are several ways to edit files. Some are easy to use other are much harder.

- hard ways
  - **vi** / vim (terminal based file editor)
  - **nano** , easier than vi (also terminal based)
- easy **nedit** : notepad like file editor

Shell programming

Shell **scripts** can be handy to automate routine tasks

The shell is a **complete** programming language, with iterations, tests, numerical operations and so on.

However, it is **not a compiled** language (a shell script is never translated into machine code) and therefore it is not recommended for **computer intensive** tasks.

Command line arguments

Arguments are assigned to **special variables** (positional parameters)
The first argument is **$1**, the next **$2**, etc.

```
[liste@goblin ~]$ cat shell1.sh
echo $1
echo $2
echo $3

[liste@goblin ~]$ bash shell1.sh one two three
one
two
three
```

Self executable script

It is possible to make a script **self-executable**.
No need then to **prefix** your shell script to execute it.

```
[liste@goblin ~]$ cat shell1.sh
#!/bin/bash
echo $1
echo $2
echo $3

[liste@goblin ~]$ chmod +x shell1.sh

[liste@goblin ~]$ ./shell1.sh een twee drie
een
twee
drie
```

That’s all

Thanks for your attention.

Outline

Introduction to the Unix operating system

Introduction to the command line

Unix file system

Unix file utilities

Unix text files processing

Output streams, Unix jobs & processes control

Unix shell

Optional Exercises

Exercises

1. Check the load on the cluster.
2. Create a shells script. Put the followings commands in it: "date", "hostname", "sleep 30" and submit it to the cluster. Check if it is running. When it has finished check the output files.

Exercises

Exercise: Create a connection to Goblin/Vampire using your preferred approach.
Exercise: Type in an echo command. Practice command line **utilities**: tab completion, move to the end of the line, move to the beginning of the line, erase the line.

Command line intro: exercises

Mention with each answer which command line you used.

1. How many **.h** files in the /usr/include directory begin with the letters **ma**?
2. How many **.h** files are there in the **/usr/include** directory (tip: combine ls and wc -l)
3. What is the size of the file named **unistd.h**, located in the **/usr/include** directory?
4. Let's consider the command **tar -x -v -f toto.tar**, is there a way to make it shorter?
5. The command **wc -l** can be used to count the number of lines in a file. Use this command to count the number of files in the **/usr/bin** and **/sbin** directories (you might have to combine it with other commands), and check that if you combine the list of files of the two directories in the same output file the sum is correct.

Getting help: exercises

1. What is the -**B** option of the **ls** command doing? You should be able to find the answer in at least **two** different ways.
2. How many man pages contain the word **concatenate** (you might have to use the command **wc -l**)? How many of them are in the section 1?

Files and directories: exercises

1. What is the command **ls ./ls** doing? Can you propose an alternative syntax?
2. What command would you use to remove everything under **/bin** without removing **/bin** itself? Will it work?
3. What is the difference between the commands **cp toto.txt toto2.txt** and **mv toto.txt toto2.txt**?
4. Create a directory called "UnixCourse" in your home folder.
5. Navigate to that folder and copy the file /scratch/tmp/PSB_unix_intro.txt to it.
6. Read the content of the file.

Files and directories permissions: exercises

1. What command would you use to make a file that has the permissions -**rw-r–r–** executable by anyone?
2. What are the security consequences of creating a file with permissions **777**?
3. The command cd bar failed, assuming that bar is a directory, how can that happen?
4. Assuming that a file's current permissions are rw-r-xr–, specify the chmod expression required to change to (i) **rwxrwxrwx**, (ii) **r–r**—–, (iii) —**r–r–**, using both relative and absolute methods of assigning permissions.
5. If a file has permissions **000** can you still remove it? Explain why you may or may not be able to remove the file.

Exercises

1. Create a symbolic link in your home directory to the file /home/liste/UNIXsandbox/cds.cre.tfa.gz and name it "chlamyCDS".
2. Use the link to count the number of lines in this file, without actually decompressing the file physically.
3. Create a **tar** file named bin.tar with the content of the /bin directory.
4. Compress the tar-file you just created (with gzip and bzip2).
5. Create a text file with the **echo** command and convert it to the DOS format.

Exercises

1. Print the 96th line of the file /home/liste/UNIXsandbox/IDlist.txt. Give 2 possibilities to do this.
2. In the previous file, find out: how many unique lines there are, which one(s) are the most occurring and how many non-redundant ones
3. Consider the file /home/liste/UNIXsandbox/geca_genes_exons.csv . Create a table listing the number of genes that have 1 exon, 2 exons, ...
4. How many **proteins** are there in the fasta file /home/liste/UNIXsandbox/TAIR10_pep_20101214.fasta.gz (Arabidopsis sequences - hint: use gunzip & grep)?
5. The fasta header line for this file contains several fields, separated by the pipe character. Use the cut command to select the only the **symbol** field.
6. Arabidopsis proteinIDs are following the syntax **AT(chr number)G(gene number)**. How many proteins are encoded on chromosome 1?
7. The last field of the header line contains information about where on the chromosome each encoding gene is located in the form of chr1:18870555-18872570. Write a command to extract only the start coordinates from the header lines without the chromosome number. **Hint**: you can chain multiple cut commands together with each other using a different delimiter.

Exercises

1. Add the path /home/<your name>/bin to your PATH variable and make that change permanent.
2. How many commands do you have in your history file? How many blastall commands?
3. Create an alias named goblin for the command 'ssh liste@goblin.fvms.ugent.be' (replace liste with your own login name). Try it.
4. Write a shell script that prints the message "The current directory is ", followed by the current directory. Make this script self-executable.