Introduction to Linux

and to the Wexac cluster

Ester Feldmesser

Thanks for slides and ideas

- Dr. Shifra Ben-Dor
- Dr. Noa Wigoda
- The IT unit at WIS

Home › High Performance Computing - WEXAC

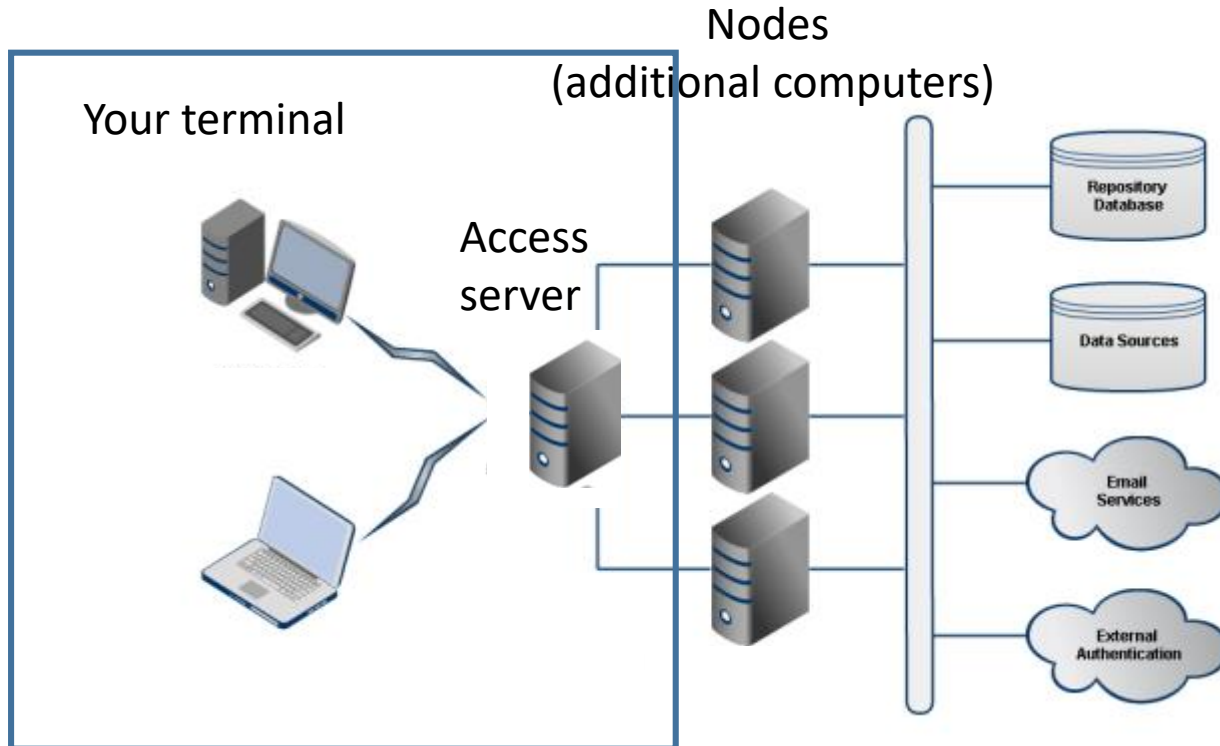## High Performance Computing - WEXAC

Additional links

- http://www.weizmann.ac.il/DIS/sites/DIS/files/uploads/it/files/wexac_introduction.pdf
- http://www.weizmann.ac.il/DIS/sites/DIS/files/uploads/it/wexac_training_session.pdf

LINUX

The operating system of all the BIOINFORMATICS computers in WIS.

The cluster of computers that is used for NGS analysis, WEXAC,

has a linux operating system.

Wexac architecture

Nodes
(additional computers)

Your terminal

Access
server

Repository Database

Data Sources

Email Services

External Authentication

- A terminal is a device used to communicate with other computers (entering data into, displaying and sending commands)
- A software is needed to open a terminal and connect to the access server
- From the access server you can send jobs to the nodes
- In modern computers many processes run at once

To open a terminal on a PC:
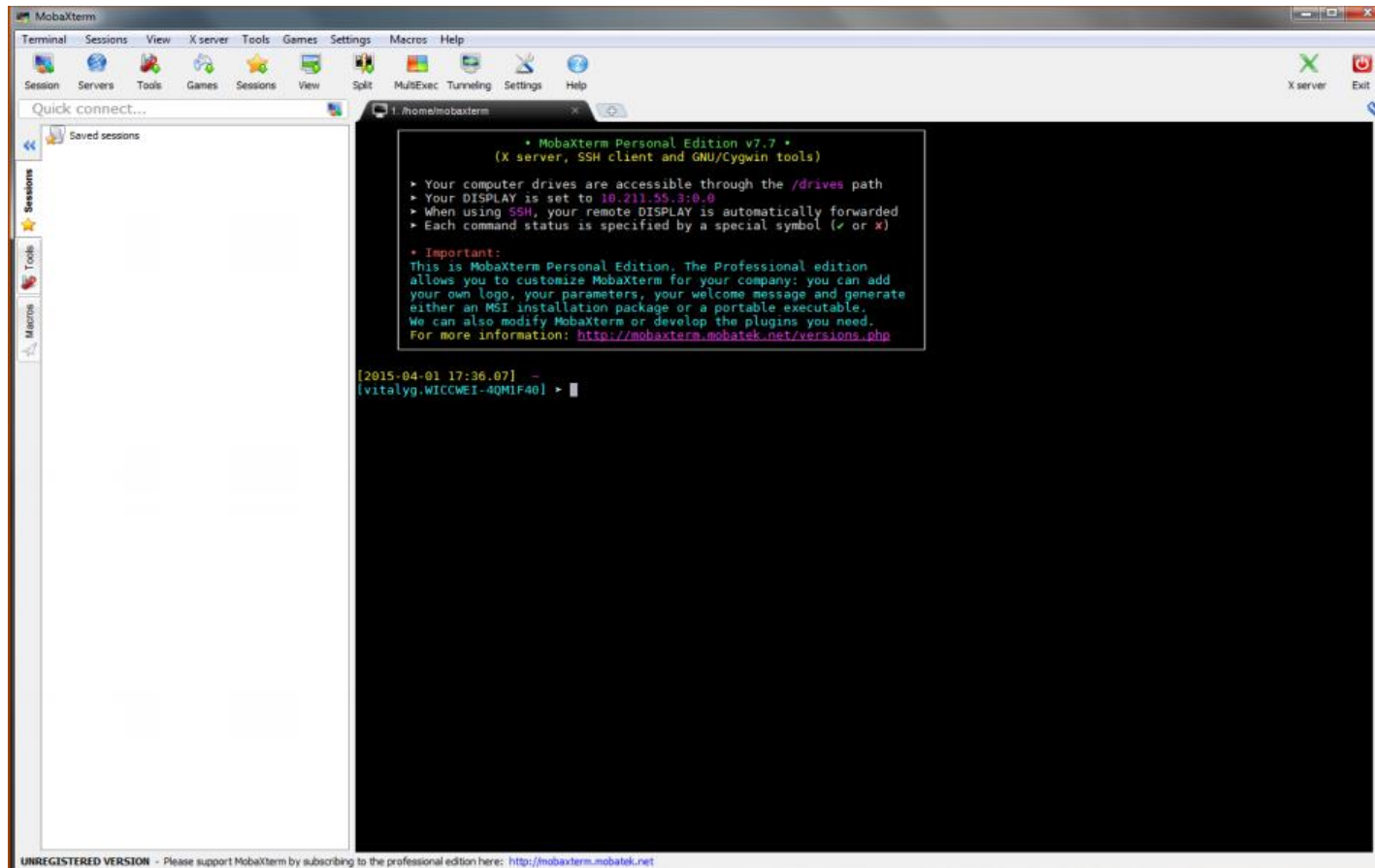
1. Write *ssh userid@access2.wexac.weizmann.ac.il* **(commands will be in Italic)** , you will be requested a password
2. Don't forget to click enter in the keyboard at the end of each command

OR

1. Go to Session in the upper menu, choose New session, in the new window choose ssh and then fill the required fields.

ssh is a communication protocol

In order to use the system you must have:

- Userid - provided by system administrator.

- Password - provided by system administrator.

- Passwords - the typed text is not displayed. Click Enter after typing.

- The userid and password on a specific computer allow the user to access his "home directory" on this computer. From there he may run programs, create files and save them.

- A user's home directory is protected and other users can not read/write there.

"Golden Rules" for working with Linux

- linux is a **case sensitive** operating system. Most commands are in lower case.

- **Never** use a space as part of a file or directory name.

- **Never** start a file or directory name with the " - " character.

- **Never** use these special characters:

  ! @ # $ $ / & * in a file or directory name.

- Think twice before you delete or rename a file or directory. Linux saves no previous versions. Files and directories can be overwritten.
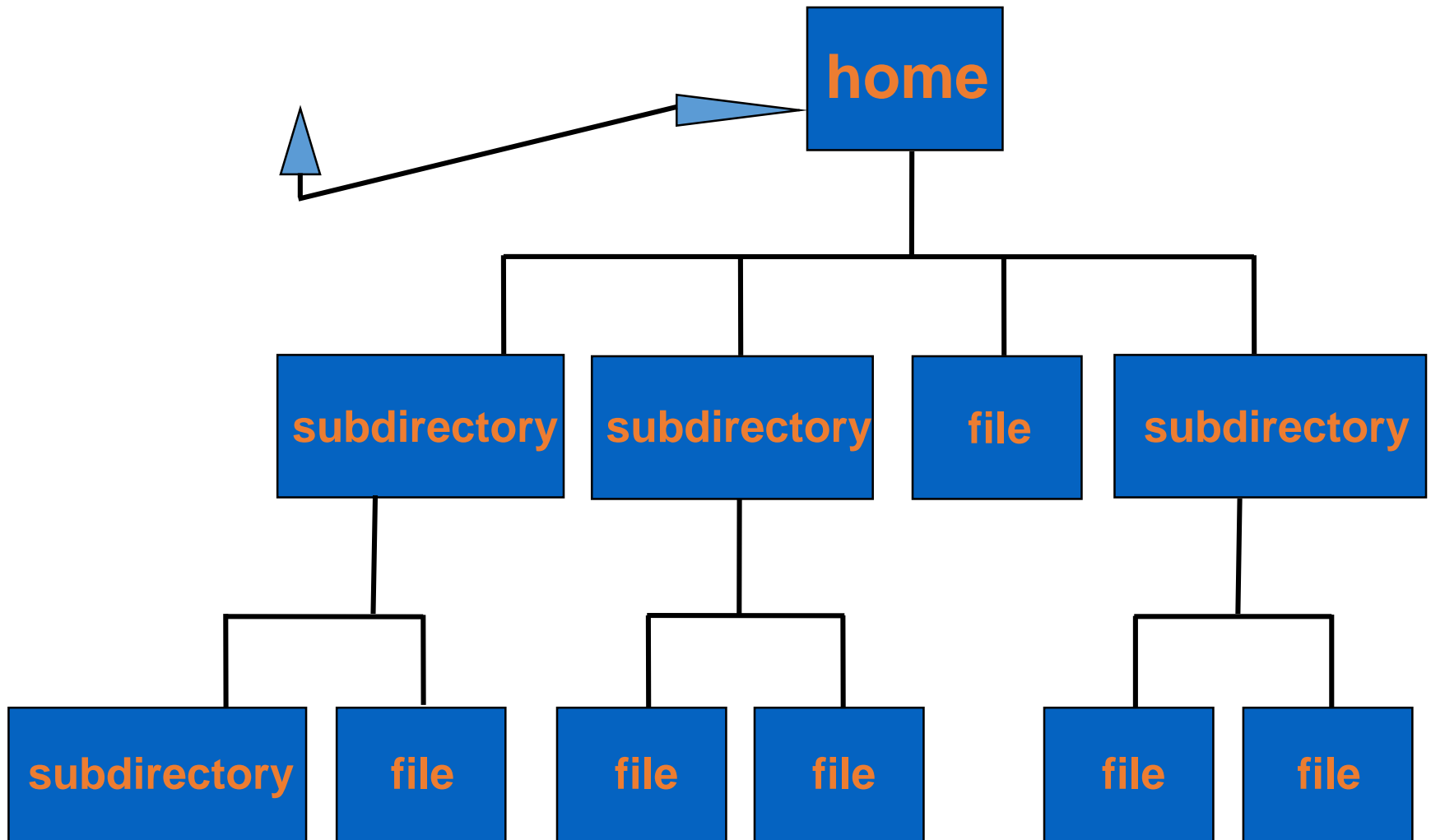
Navigating within the linux file system

The linux file system is built like a tree
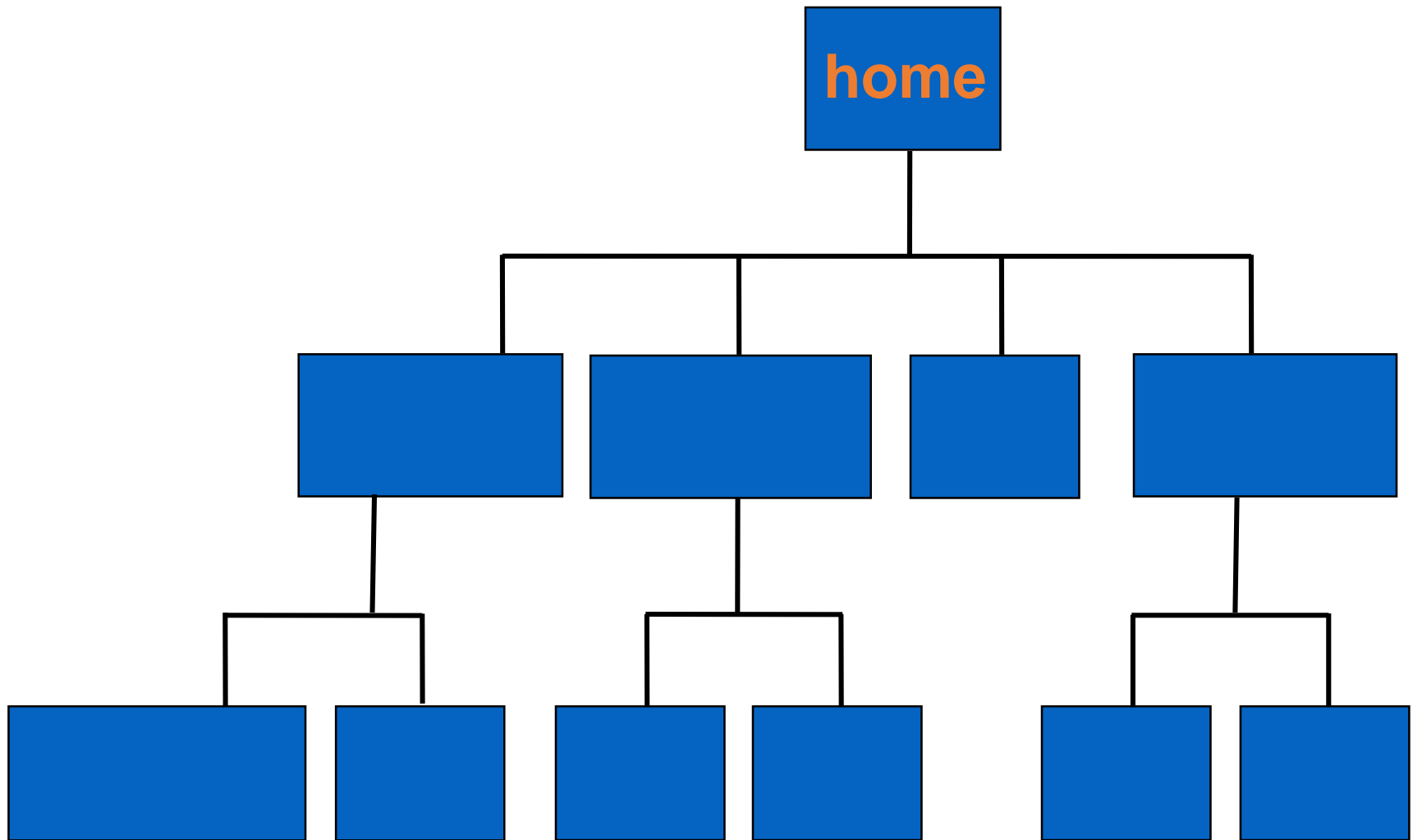
Linux command line format

- When you log on to a linux computer like dapsas, you'll receive the following prompt:

[esterf@bio ~]$

(~ represents your home directory)

- You can enter a command after the "$"
- Example:

  To list contents of a directory:

[esterf@bio ~]$ *ls*

1025.seq	hsil2rbc.frg
upstream.fasta	hum-gen/
1430.seq	human.pep

Linux Command Options:
different preferences for a command

- Unix commands and most software can be run with parameters or options.

- Options allow the user to execute a command with different preferences, in order to get results closer to what you want

- Syntax for using options:

    $ *command -option*

Example of the various options of the "*ls*" command

• To list all files:

$ *ls -a*

.AppleDouble/	gidi/	il2ra.seq
.addressbook	gnrh.blast	il2rb
.history	gpcr.pair	il2rb.con
.login	gpcr.rna	il2rb.fil

• To list detailed file information:

$ *ls -l*

```
-rw-rw-r--   1 lishifra bioserv     2760 Dec  9  1992 0214.pdoc
-rw-rw-r--   1 lishifra bioserv    32508 Aug  7  2000 1025.map
-rw-rw-r--   1 lishifra bioserv      954 Jul 23  2000 1025.mapsort
```

More than one option can be used with one command

- Example:

  The "*ls*" command used with the "*-lt*" options. These options will give a detailed list of the files in the directory by order of their creation time (newest first in the list):

$ *ls -lt*

drwxr-xr-x	2 lishifra bioserv	8192	Feb 24 12:06 rutyw/
-rw-rw-r--	1 lishifra bioserv	5564	Feb 17 12:37 p130b.aln
-rw-rw-r--	1 lishifra bioserv	54	Feb 17 12:37 p130b.dnd
-rw-rw-r--	1 lishifra bioserv	4926	Feb 17 12:37 p130b.msf
-rw-rw-r--	1 lishifra bioserv	1403	Feb 17 12:36 p130b.tfa
drwxrwxr-x	2 lishifra bioserv	8192	Feb 16 01:43 multiple/

Wildcards

- Sometimes it is necessary to specify a group of files with one or two symbols.

- This can be done by using special symbols that serve as "place holders."

- A wildcard serves as an ambiguous replacement for one or more characters, and means "anything or nothing".

Using Wildcards

Wildcards can be used to select a few files if their names have a common character(s)

Examples:
Select all the files that end with "pep"
$ *ls *pep*
mouse.pep   u76bws.pep

Select all the files that have "gb" as part of their name
$ *ls *gb**
humil2rbc.gb_pr l05404.gb

Creating and manipulating directories

- When logging on to a linux computer, the user accesses his home directory.   In his home directory he may create other subdirectories. A subdirectory is a directory that is inside another directory.

- (Remember the Tree structure of linux file system)

- Directories and subdirectories are also called folders
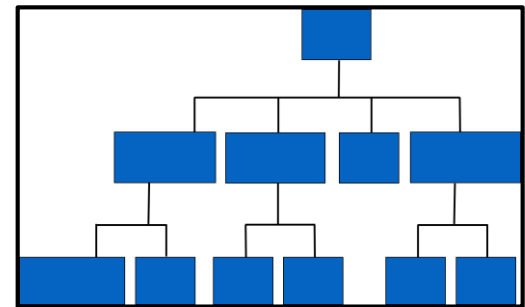
Creating and deleting directories

New directories are created by using the command:
  $ *mkdir directory_name*

Empty directories can be deleted by using the command:
  $ *rmdir directory_name*

Directories which are not empty can be deleted by using the command:

  $ *rm -r directory_name*
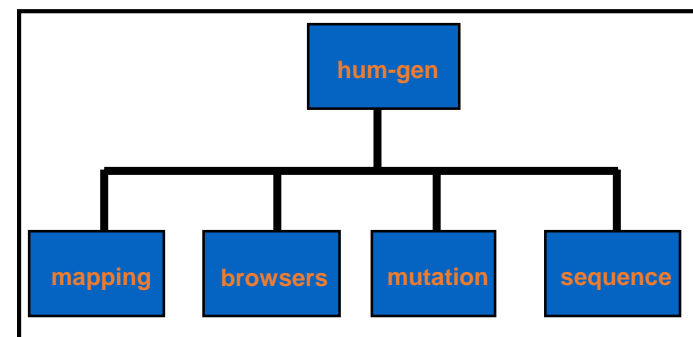
Navigating between subdirectories

To move from a directory to a subdirectory use the command "cd" (change directory)

Example 1:

lishifra42 [~]$ *cd hum-gen*

lishifra43 [~/hum-gen]$

Example 2:

lishifra44[~]$ *cd hum-gen/mapping*

lishifra45[~/hum-gen/mapping]$

The place of the square brackets and the length of the path can change

To move up from a subdirectory to a subdirectory above: $ cd ..

Example1:

[lishifra mapping]$ cd ..
lishifra46hum-gen/]$
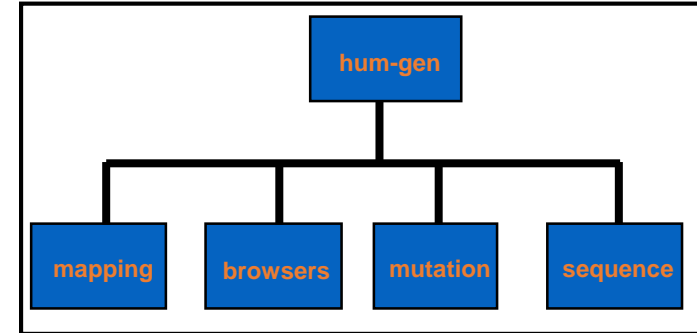
Example2:

lishifra45[~/hum-gen/mapping]$ cd ../..
lishifra46[~]$

To return directly to your home directory:

lishifra45[~/hum-gen/mapping]$ cd ~

To move between subdirectories on the same level:
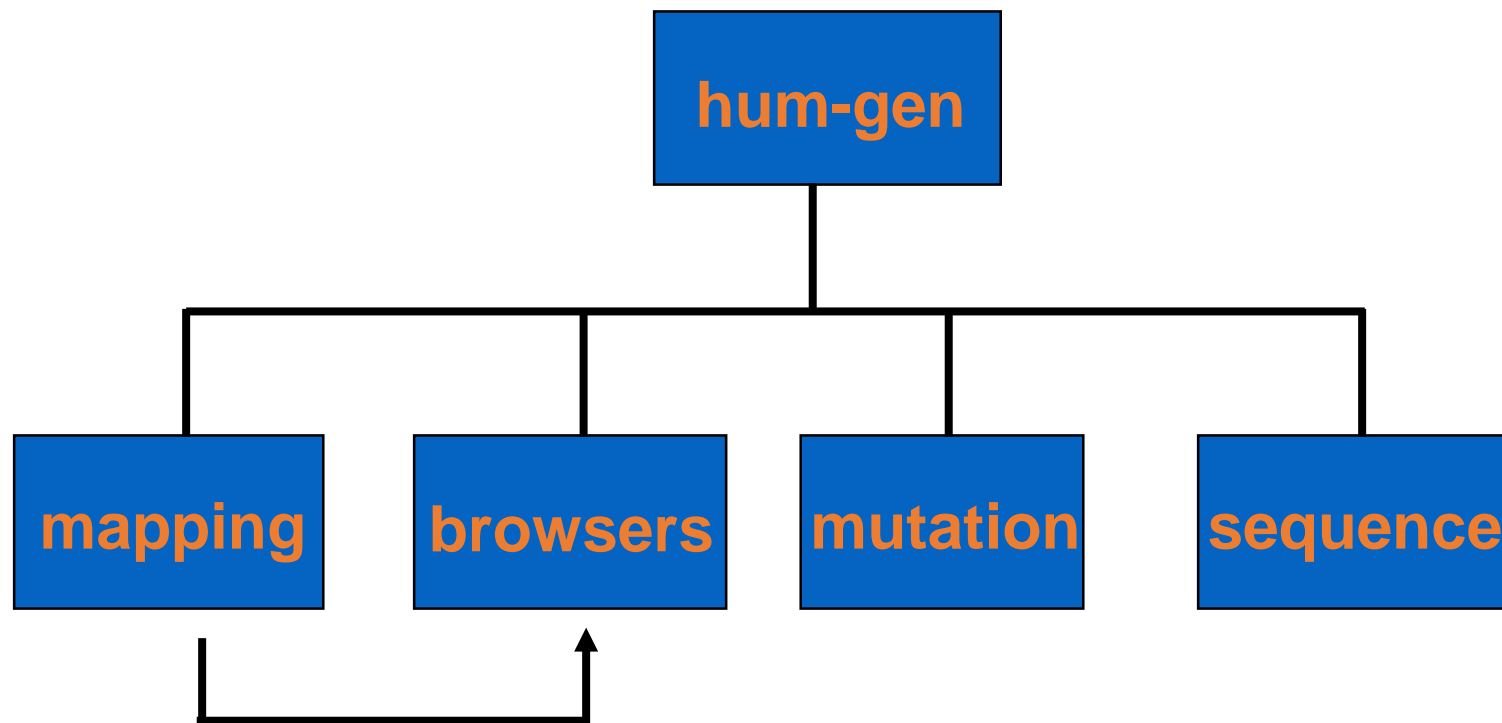
To get from here (mapping) to there (browsers)
[bfbecker mapping]$ *cd ../browsers*
[bfbecker browsers]$

Creating Files

- Files are created as output from programs that have been executed by the user.

- Files can also be created manually by using a text editor, such as emacs or vi (we will not use them today).

- A new empty file can be created by using the command:
$ *touch* filename

Copying files

- A file can be copied by using the command *cp*

    $ *cp filename newfilename*

    (You now have 2 identical files with different names in the same directory)

- Specify a path to copy a file to a new location (another directory)

    $ *cp file.1 outputs/*

    (File will be copied to the subdirectory "outputs")

- Copy a file to a new location and name it differently

    $ *cp file.1 outputs/file-copy*

- Copy a file from a different location to your working directory ("**.**" means your current location)

    $ *cp outputs/file-copy* **.**

Copying and writing commands

- Be careful when copying commands, quotes and dashes are sometimes not correctly copied and you need to delete them and rewrite.

- To copy from the terminal window, select what you want to copy. Ctrl + C in the terminal stops a running process. It does not copy.

- To paste press the wheel button of the mouse or right click the mouse and press paste.

- When writing a command, if you want to move in the line use the left and right arrows to move left and right. The mouse does not work.
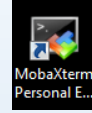
- Double click on the Mobaxterm icon.

- Connect to the Wexac computer by writing a command or by defining a new session.

- List (*ls)* the files and folders in your home directory.

- Create a new directory inside your home directory and then create an additional one inside the new directory (name these directories by yourself).

- Create one more directory named "test" at the same level as the last one and then remove it.

- Go inside the first directory that you created.

- Create an empty file with the following command:
  touch Hello (Hello is the name of the file)

- List the files to be sure that the file was created.

- Return to your home directory.
- Now you will copy a folder that I prepared to your home directory (<u>note</u>: the dot at the end of the command is part of it and it means your current directory):

  cp -R /home/labs/testing/Collaboration/linux_exercise .

  -R is for recursive; copies the folder and all its content.
- Go inside the directory linux_exercise/my_data.
- Write the command: *ls pep**
  - What did you get?
- Write the command: *ls *fa*
- Now try: *ls *fa**
- Why do you see a difference in the output?

Sftp window

- MobaXterm includes an Sftp window, where you can also see the lists of files and directories

- The Sftp window can be used to upload and download files (secure file transfer protocol)
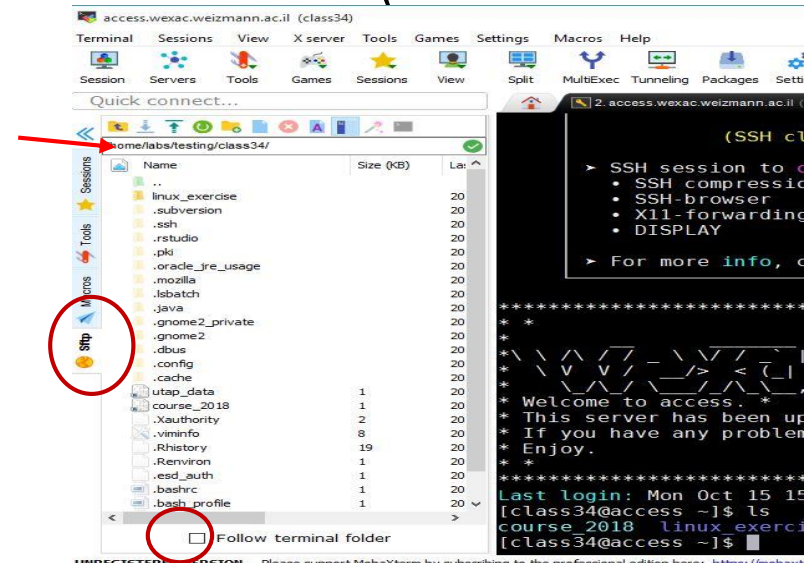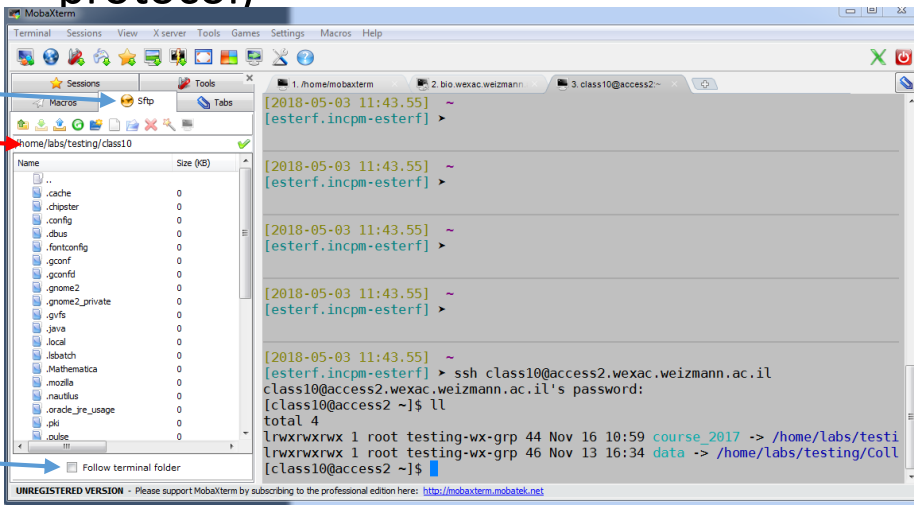
- By default you see the list in your home directory

- If you click on "Follow terminal folder", the window should show the list of the current folder.

- Sometimes it does not work, then you need to write in the window (red arrow) the full path of the folder you want to see.

- To find the full path of the current folder, use the command: *pwd* (path to working directory)

Editing a file

- MobaXterm has a build-in editor

- From the Sftp window, you can right click a text or an empty file name and open it in a new window with the default text editor (figure at the right).

- Now you can then edit it and save it back in the same place (directory).

- The editor window will work as a regular PC window
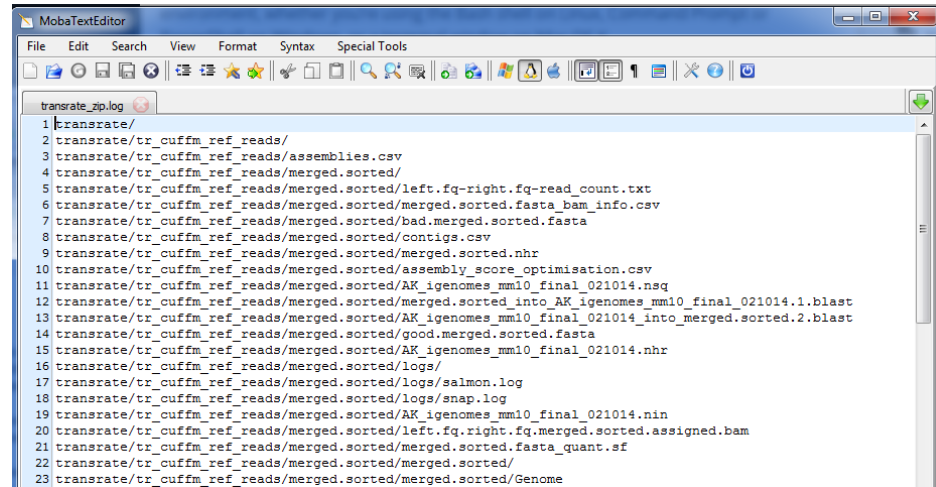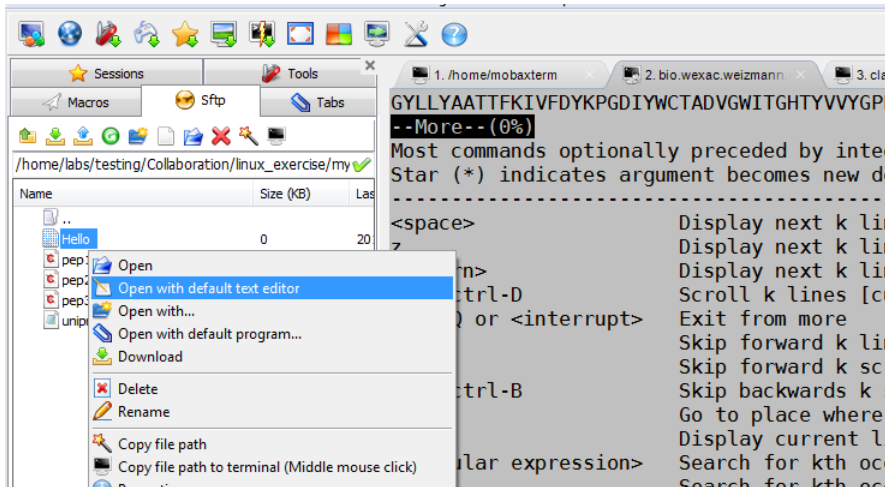
Displaying files on your screen

To display the contents of a file on your screen,

you can use 2 commands.

- The "*more*" command to display it slowly stopping at every page.
- The "*cat*" command to view a file rapidly.

The command format is:

$ *more file_name*   or $ *cat file_name*

Files from your current directory or from another directory can be displayed, by using the exact path of the file location.

Example:

$ *more ../mouse.pep*

Using the more command

- When viewing a file with "more" you may want to use the command options.

- When you are in the display: to search for a string press "/"
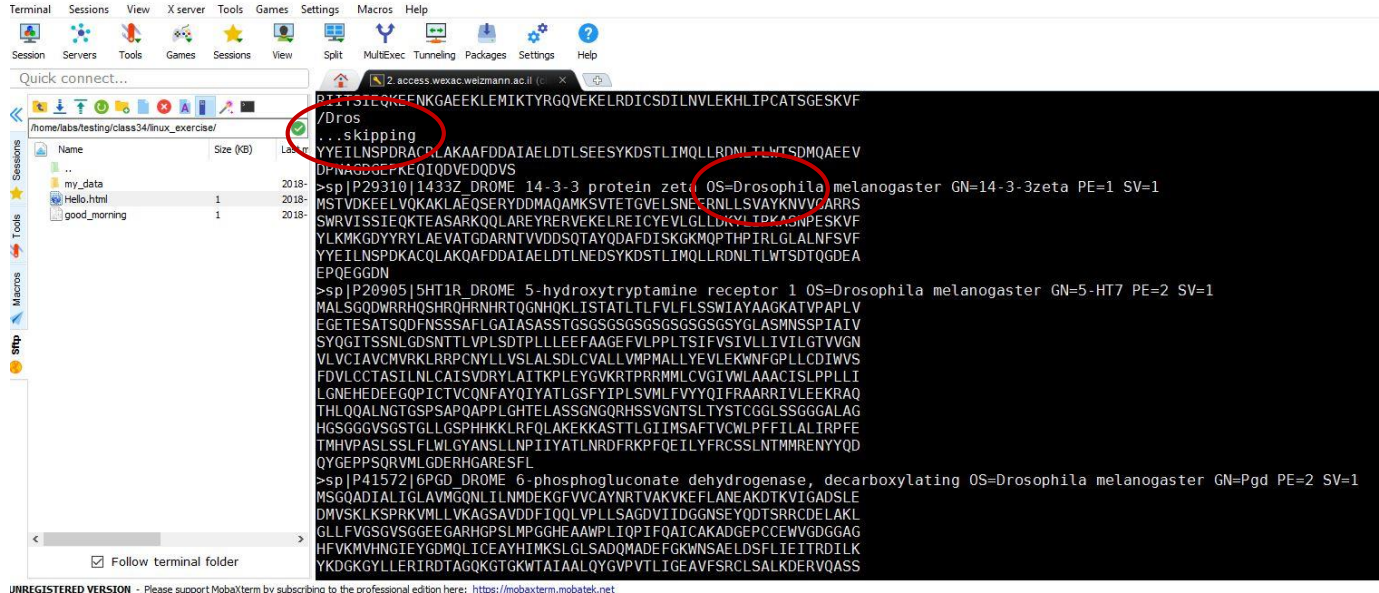
- To exit the file display press "*q*"

Renaming and Moving files

- Files can be moved from one location (directory) to another by using the command mv
  $ *mv file_name dir_name*

Example:
  $ *mv sss.seq outputs/*
(the file sss.seq has been moved from it's location to the

  subdirectory "outputs")

- If you specify a non-existing directory, the file will be renamed to the "directory_name" instead of moving there.
  Example:
  $ *mv file.1 output*
(The name of file "file.1" will be changed to "output")

Renaming and Moving files

- To rename a file we use the same command "mv" . The command format is:

  $ *mv file_name newfile_name*

  Example: [~]$ *mv sss.seq s1.seq*

- You can move a file to a new location and also rename it with the same command:

  $ *mv file_name dir_name/newfile_name*

  Example: [~]$ *mv sss.seq tmp/s1.seq*

Deleting Files

- Files can be removed using the command:

  $ *rm file_name*

  Example:

  lishifra50[~]$ *rm il15.seq*

- To remove a group of files that have some common characters in their names, use the command:

  $ *rm *.seq*

  Example:

  lishifra51[~]$ *rm *seq*

Deleting Files

- Please take notice that linux does not tolerate spelling mistakes. If you make a mistake you may accidentally erase a file or a directory, so you have to be careful when deleting or renaming files and directories.

- On the Wexac, linux will prompt you to confirm overwriting or removing files. In order to actually execute the command you will have to enter: y (yes) when prompted.

Redirecting output

- By default, there are commands (programs) that redirect their output to the screen (terminal)

- Instead of displaying output on the screen, you can also redirect the standard output to a file

- To redirect the standard output from a command (program) to a file, use the > (greater than) symbol followed by the name of the output file.

  $ align t.seq f.seq > out

  If the file that you redirected the output to does not already exist it will be created. If it exists it will be overwritten.

Linking commands - Piping

- Piping is another way of redirecting commands input and output. linux allows more than one command to be linked together by a pipe "|".

- The output of the first command becomes the input for the second command, without creating an intermediate file.

- By creating a pipeline between a number of simple commands, the user can perform a complex function.

- Example: $ *ls -lt | head -1*

Tip: completing words automatically

- To complete words, you can use the Tab key

  For example, let's say you want to run the **firefox** command. You can just type **fir** or **fire** into the terminal and press Tab — if your system doesn't have any other commands that begin with those letters, it will automatically fill in **firefox** and you can press Enter to run the command.

- If there is more than one command that starts with **fire** and you print and it and press Tab, you will get a list of all the commands that start with fire.

- Go into the directory where you created the file "Hello".
- Run the command "*pwd*" to see the full path of the directory you are in.
- Click on "Follow terminal folder", it should show the list of the folder you are currently in
- If it does not work, then you need to copy the full path of the folder you want to see as explained in the lecture.
- Open the Hello file from the Sftp window and write in the file:

  Good morning world!
  What a beautiful day!
- Save and close the file.
- Use the command *more* to view what you have written in the file.
- Now we want to view the text of the uniprot-all.fasta file that is located inside linux_exercise/my_data. Use both commands: *more* and *cat* to view the file content.
- Press q to exit the more command.

- Run again the *more* command and this time look in the file for the string "Acetylcholinesterase" by typing:

  /Acetylcholinesterase

  Do not forget to press enter.

- Press q to exit the more command.

- Rename the Hello file to Good_morning

- Run the command $ *who > who.file*

  "who" displays a list of all the users who are logged on at that moment.

- Write *cat who.file*

- How many users are looged in now?

- Write *wc -l<who.file* or *wc -l who.file*

  - In this case, the two commands give the same result

  - *wc -l* counts the number of lanes in the file

- Copy the file uniprot-all.fasta to a file called "uniprot-all.new.fasta" and then remove it. Try to use the tab button to complete names.

Exercise 1 part 2 - slide 3 of 3

- In your home directory, write: *ls*, then: *ls | head -3,*

  then: *ls | head -3 | tail -1*

  and last: *ls | head -3 | tail -1 > myoutput*

- See what was written into myoutput. Be sure that you understand the incremental processes of the commands.

- Go to the directory linux_exercise.

- Print: *firef*, press the tab key to complete firefox, press the space key, print: *Hello* and tab to complete and then enter.

- Wait for the window to open. That's the way to open and see html files in linux.

- Close the web browser.

Wexac architecture
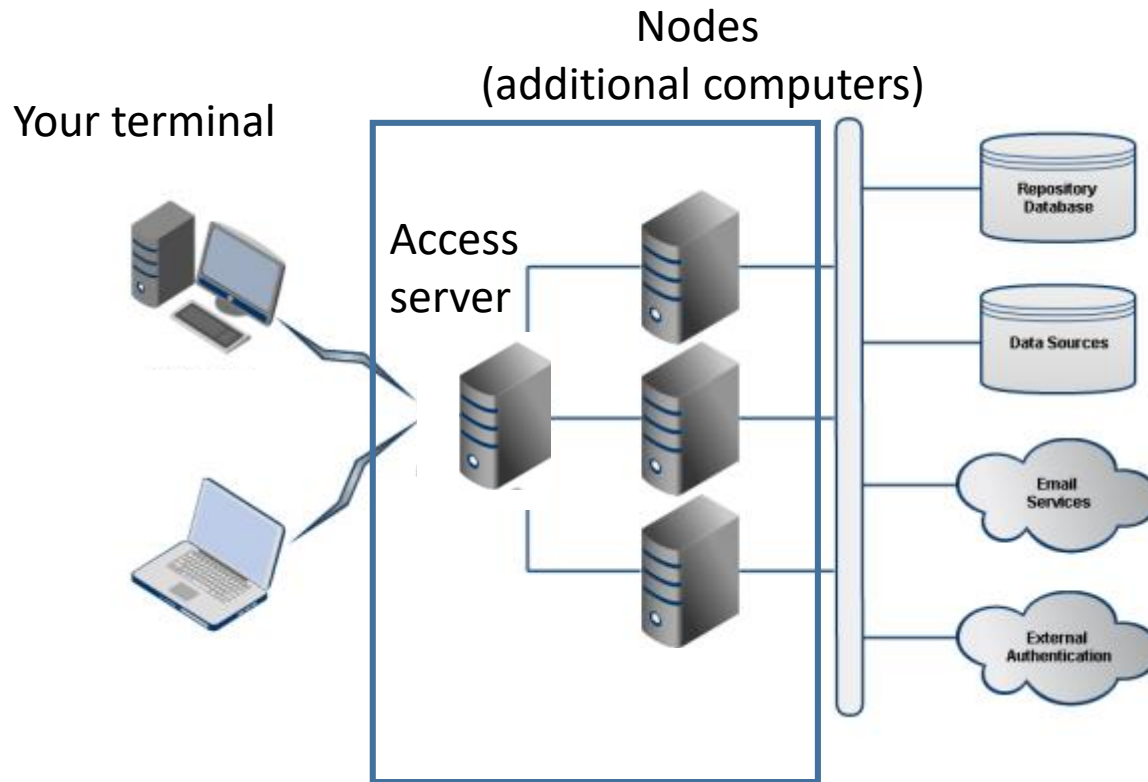
- A terminal is a device used to communicate with other computers (entering data into, displaying and sending commands)
- A software is needed to open a terminal and connect to the access server
- From the access server you can send jobs to the nodes

Modules

- The Cluster hosts a large and extensive set of software
- Using the [Environment Modules Package](#) or "modules" for short to keep the software organized.

```
[esterf@bio ~]$ module --help

 Modules Release 3.2.10 2012-12-21 (Copyright GNU GPL v2 1991):

 Usage: module [ switches ] [ subcommand ] [subcommand-args ]


Switches:
    -H|--help          this usage info
    -V|--version        modules version & configuration options
    ...
 Available SubCommands and Args:
    + add|load         modulefile [modulefile ...]
    + rm|unload         modulefile [modulefile ...]
    + switch|swap        [modulefile1] modulefile2
    + display|show       modulefile [modulefile ...]
    + avail           [modulefile [modulefile ...]]

        ...
```

Examples of Modules

❖ To work with a software that is installed as a module, you need to load it
❖ Also to work in the accession server or to send a job to nodes in the cluster

- *module avail*
- *module load blast/2.2.26*
- *module list*

  Currently Loaded Module files:

  1) blast/2.2.26

- *module unload blast/2.2.26*
- *module list*

  No Module files Currently Loaded.

Job Submission & Control

`bsub`   - submits a batch job to LSF

`bjobs`   - displays information about LSF jobs

`bhist`   - displays historical information about jobs

`bkill`   - sends signals to kill, suspend, or resume
     unfinished jobs

`bmod`   - modifies job submission options of a job

`bpeek`   - displays the stdout and stderr output of an
        unfinished job

`bstop`   - suspends unfinished jobs

`bresume` - resumes unfinished jobs

`bswitch` - switches jobs from one queue to another

bsub - Methods for Submitting Jobs

- By script or command
```
$ bsub -q      new-all.q -J example -o
example-%J.o -e example-%J.e date
```

- By job spooling
```
$ bsub < job_file
```

```
job_file example:
#BSUB -q new-all.q
#BSUB -J example
#BSUB -o example-%J.o
#BSUB -e example-%J.e
date
```

`bsub -q new-all.q -J example -o example-%J.o -e example-%J.e date`

%J is for job number

date is the command that I want to use.

Usually the command I want to send to the cluster is much more complex and longer.

In this case I can write the command into a file. Let's say the file with the command is called command.txt. Then I can sent the command to the cluster:

`bsub -q new-all.q -J example -o example-%J.o -e example-%J.e <command.txt`

Redirecting Standard In (stdin) <

The same basic redirect can also be done in the reverse direction in that an interactive program that requires input from a user can be automated.

bsub -q new-all.q -J example -o example-%J.o -e example-%J.e date

The command I am interested in

Telling the cluster how to perform the command

%J is for job number

-q qname	submits the job to the specified queue
-o file	redirect stdout, stderr and resource usage information of the job to the specified output file
-e file	redirect stderr to the specified error file
-J jobname	assigns the specified name to the job
-R res_req	runs job on a host that meets the specified resource requirements

- In the cluster, jobs or processes are placed in an array called a **run queue**.
- The queue manages and gives priority values to each process.
- There are several queues in the Wexac, some of them are public (everyone has the same priority on them) and others give priority to certain groups in the nodes they bought.

Resource Requirements Examples

Example 4. Candidate hosts should have min 500MB free RAM, job will reserve 400MB RAM.

```
$ bsub -n 4 -R "select[mem>500] rusage[mem=400]" myJob
```

Example 5. All slots required for a parallel job should reside on the same host

```
$ bsub -n 4 -R "span[hosts=1]" parallelJob
```

-n number specify number of job slots

Job Submission & Control

`bsub`   - submits a batch job to LSF

`bjobs`   - displays information about LSF jobs

`bhist`   - displays historical information about jobs

`bkill`   - sends signals to kill, suspend, or resume unfinished jobs

`bmod`   - modifies job submission options of a job

`bpeek`   - displays the stdout and stderr output of an unfinished job

`bstop`   - suspends unfinished jobs

`bresume` - resumes unfinished jobs

`bswitch` - switches jobs from one queue to another

Exercise 1 part 3 -slide 1 of 3

- Send the following job to the cluster:

bsub -q new-all.q -J example -o example-%J.o -e example-%J.e date

- Run *bjobs* command again and again to see if your job is pending (PEND) or running (RUN) until you get "No unfinished job found".

If it takes too long to finish you can kill the job and use the queue: short

- Look for the file with the suffix .o and see if the date appears at the beginning of the file.

- Send the command in the two additional ways we saw changing the name of the output. Pay attention to letter upper or lower case.

- Write the commands you used to a new file called my_linux_exercise. Write in this file the answers to the questions below.

- Question 1: Compare all the outputs.

- Question 2: What is the full path to your home directory?

- Go back to your home directory if you are not there. Send the following job to the cluster. Note that we are sending two commands separated by ";":

bsub -q new-all.q -J example -o example-%J.o -e example-%J.e sleep 30;wc -l linux_exercise/my_data/uniprot-all.fasta>count

Sleep is just wait, the number after is seconds to wait.

- Check with *bjobs* until the job is finished. Check the output in the file "count".

- Question 3: Explain when will you choose to use the command "mv" or "cp"?

- Question 4: Suggest how can you count the number of files/directories (only the first level) in the folder course_2018?

Exercise 1 part 3 -slide 3 of 3

- I wanted to get a list of files sorted by date in my current directory, I typed: *ls -lT* and I got the following message:

  ls: option requires an argument -- 'T'

  Try 'ls --help' for more information.

  Question 5: Can you correct my command?

- Send to the cluster a command asking to save to a file called list.txt a list of the files that are in the directory: linux_exercise/my_data. Write the command in the file my_linux_exercise.

- Create a new folder called linux_all in your home directory and move (*mv*) to it all the directories and files created during the exercise.

Basic Linux commands:

```
ls ................... show directory, in alphabetical order
logout ............. logs off system
mkdir .............. make a directory
rmdir .............. remove directory (rm -r to delete folders with files)
rm .................. remove files
cd .................. change current directory
more  ............. views a file, pausing every screen
grep ............... search for a string in a file
head .............. show the first few lines of a file
tail ................. show the last few lines of a file
cat ................. print the content of a file
cut  ................ print selected parts of lines
cp   ................ copy file
mv   ............... move file
wc -l .............. print the number of lines
sort   ............. sort lines of text files
```