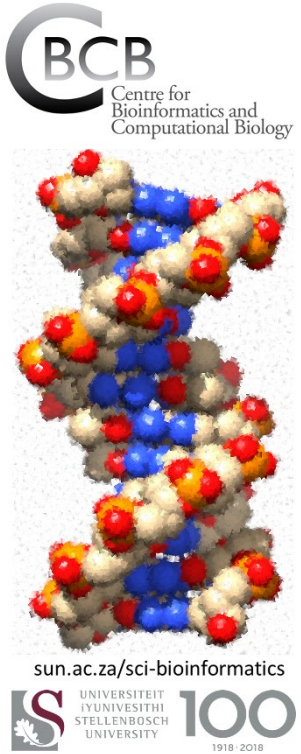
Introduction to Linux and the HPC



by Hugh Patterton
(hpatterton@sun.ac.za)

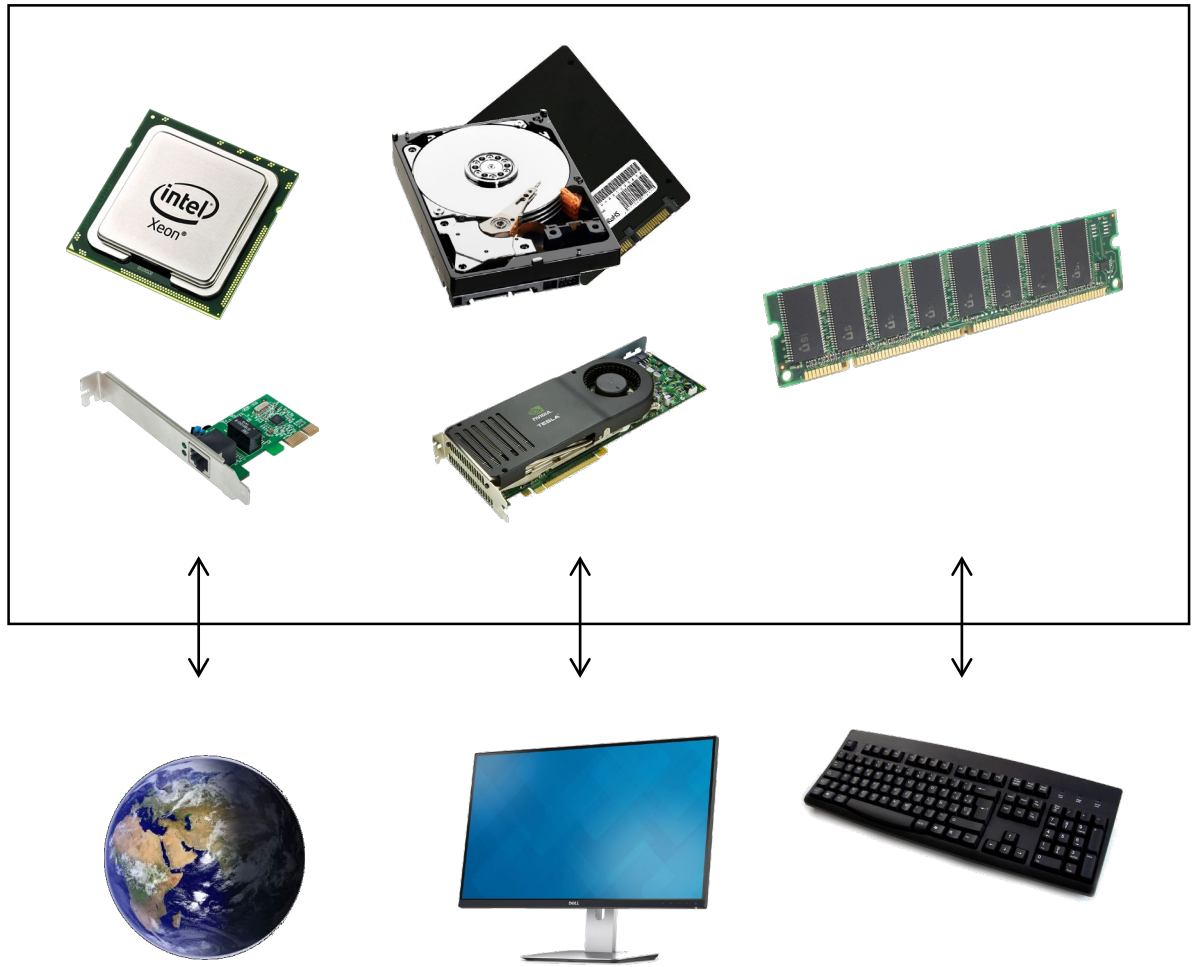
Center for Bioinformatics and Computational Biology
Stellenbosch University
South Africa

sun.ac.za/sci-bioinformatics

Notebooks & desktops

Typical standard PC architecture

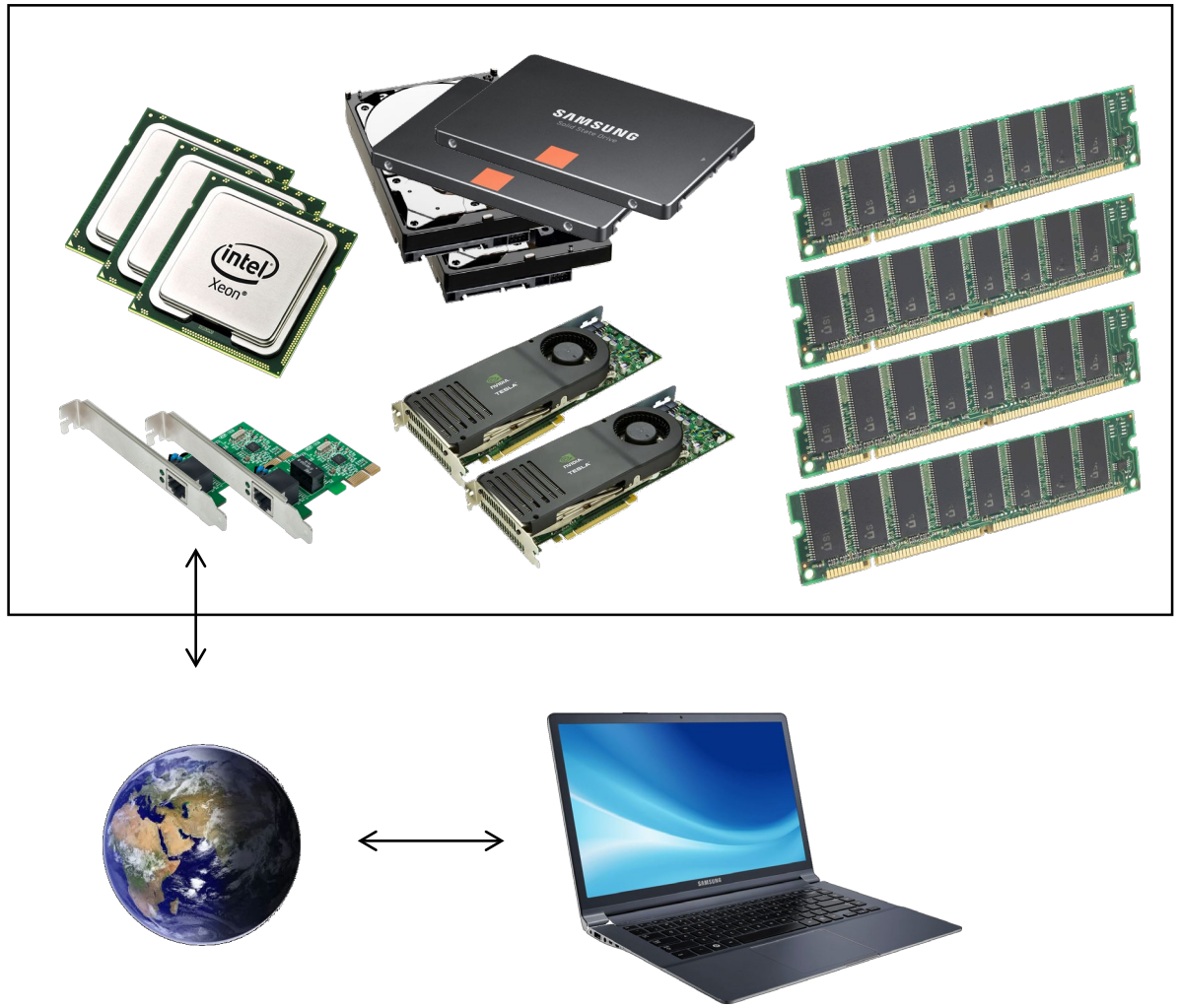
- One processor (CPU with multiple cores)
- Storage: hard drive(s), SSD(s)
- Network: Ethernet card
- Graphics processor (GPU)
- DRAM memory
- Keyboard
- LCD monitor



Servers

Typical server architecture

- One to four processors (CPU)
- DRAM memory (with ECC)
- One or more GPUs (optional)
- Storage: hard drives, SSDs
- Ethernet
- Access is almost always via network ports using TCP/IP

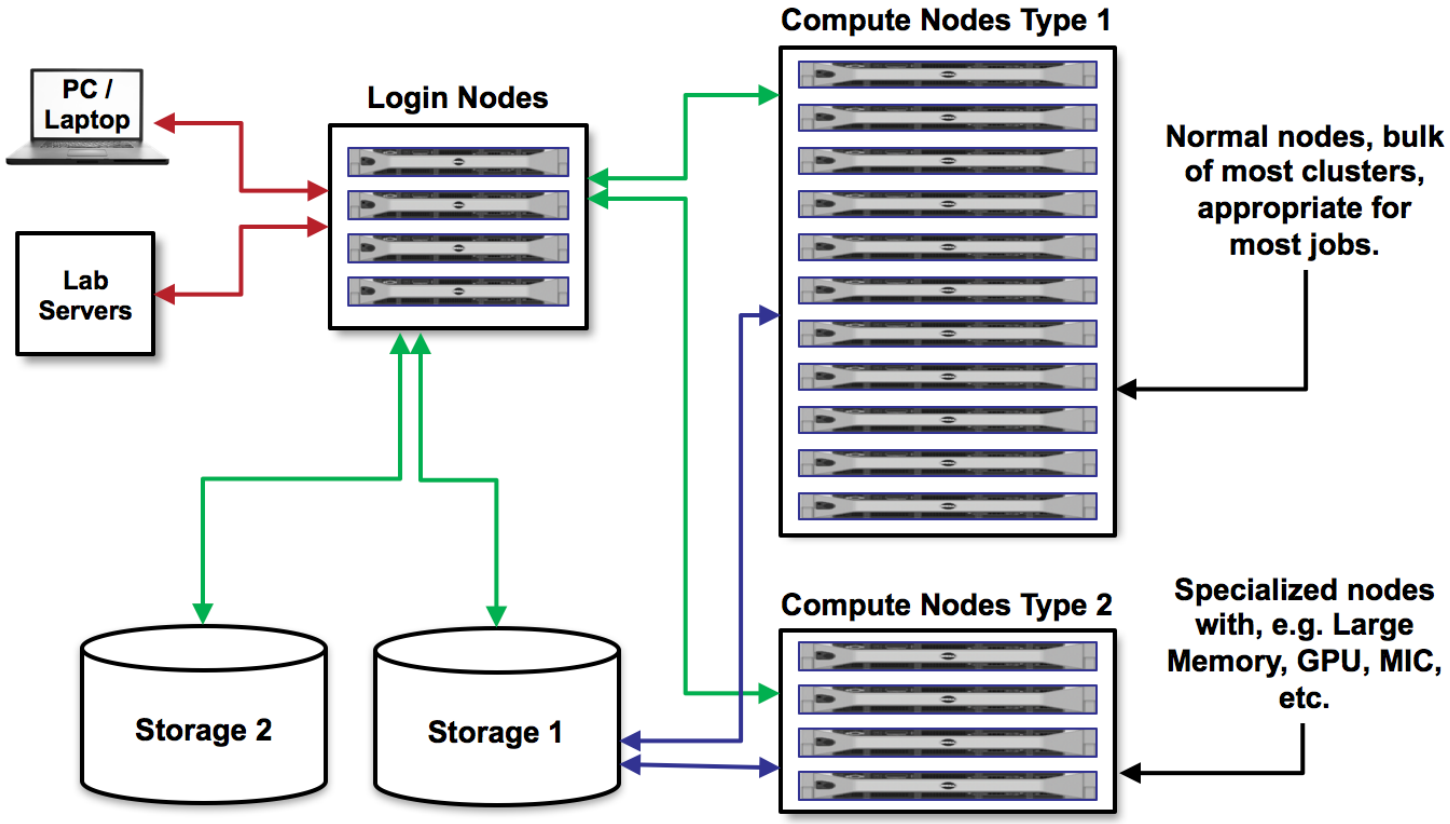


High Performance Computing (HPC)

Computational Clusters

- Many (10s-1000s) servers (“nodes”)
- Many CPUs per node with 8-80 cores
- 100Gb+ network
- Fast InfiniBand (100Gb) interconnects between nodes
- Linux

Typical HPC architecture



Why Linux?

- To use High Performance Computing, you must know how to use Linux
- Linux is robust, scalable to inter-connected server setting and is free
- Works well in multi-user environment
- Full source code is available
- Based on the principles of Unix: in use since 1969, encouraging minimalist, modular, extensible software development
- Linux have evolved from command line only to GUIs on notebooks (KDE, Gnome)
- HPC systems use the Linux command line

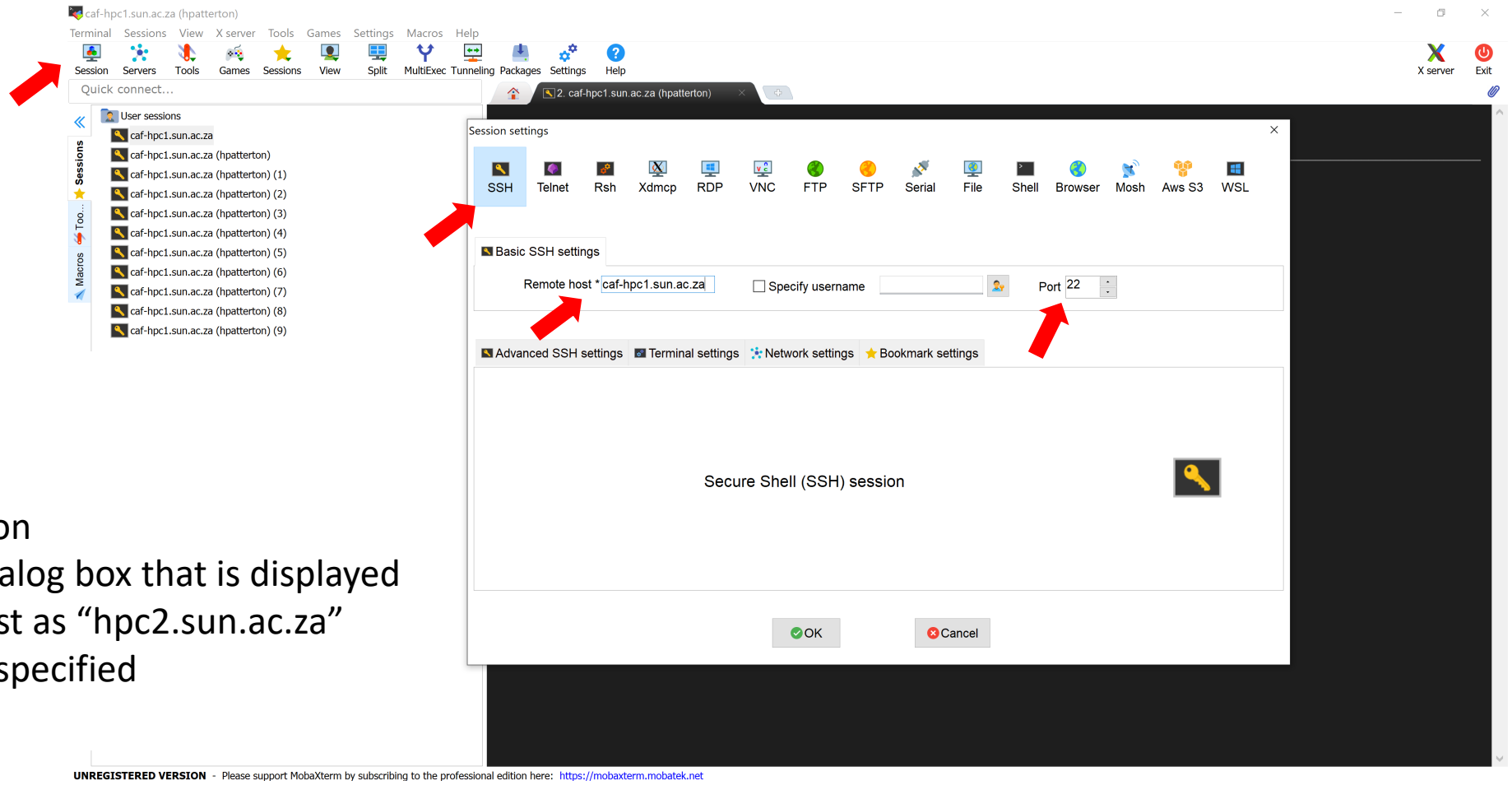
Connecting to an HPC system

- At Stellenbosch University the HPC is at hpc2.sun.ac.za
- Stellenbosch students and staff can register for an account at sun.ac.za/hpc

Use the Secure Shell protocol (SSH)

- Under Linux or Mac OS X
    - Open a terminal and type: `ssh username@hostname`
    - (for example, `ssh johnsmith@hpc2.sun.ac.za`)
- Under Windows
    - Download and use MobaXterm (https://mobaxterm.mobatek.net/)

Connecting to the HPC with MobaXterm

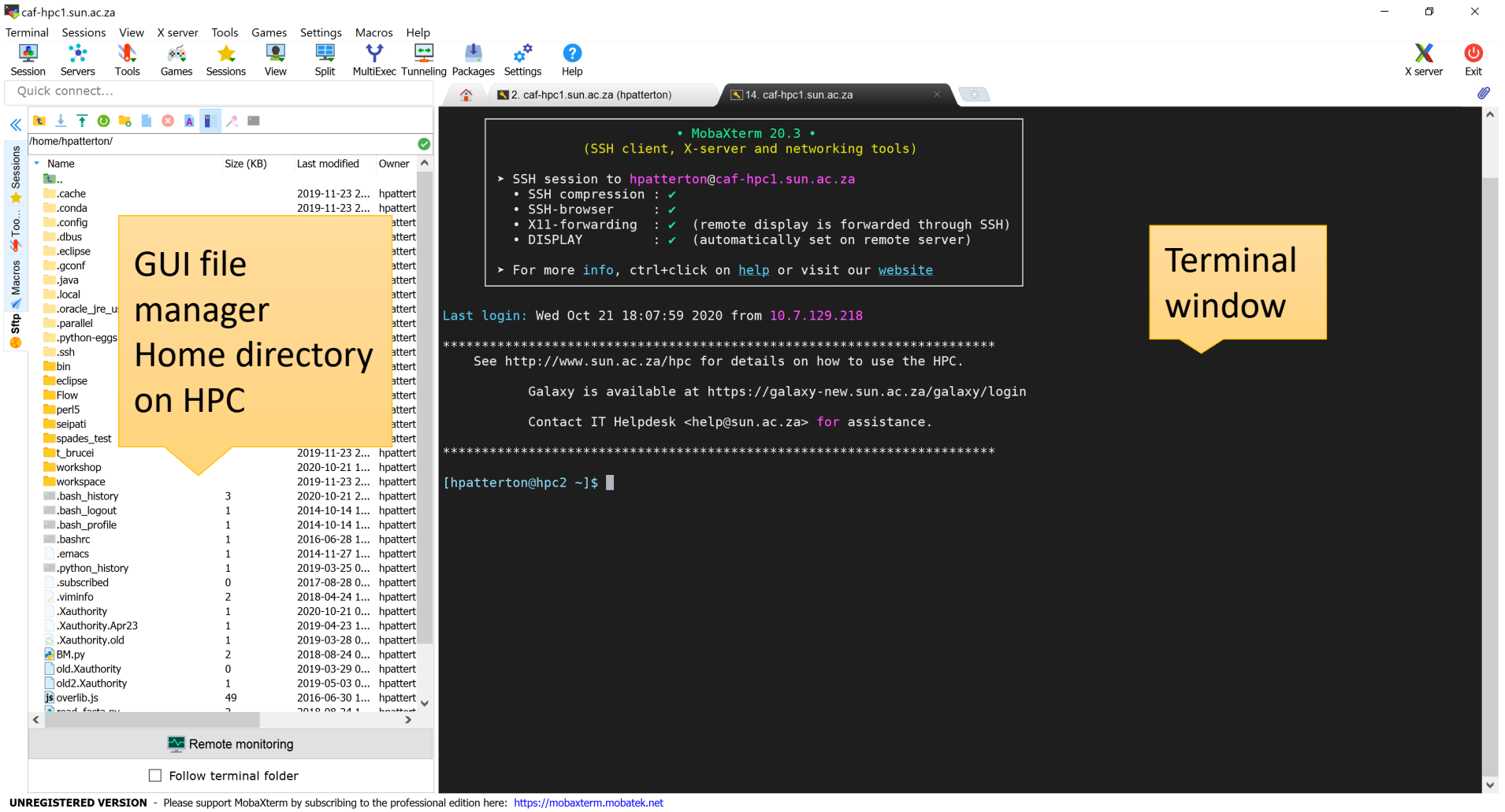


Using MobaXterm

- Launch MobaXterm
- Click the “Session” icon
- Select “SSH” in the dialog box that is displayed
- Enter the Remote Host as “hpc2.sun.ac.za”
- Make sure port 22 is specified
- Click OK

The MobaXterm interface

- Login with your username and your password, as prompted



Linux commands – getting help

Getting help

- For a brief summary on how to use a command, and the options associated with the command line options, type `command --help`.
- For example `cp --help`

```
[hpatterton@hpc2 ~]$ cp --help
Usage: cp [OPTION]... [-T] SOURCE DEST
  or:  cp [OPTION]... SOURCE... DIRECTORY
  or:  cp [OPTION]... -t DIRECTORY SOURCE...
Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

Mandatory arguments to long options are mandatory for short options too.
  -a, --archive                same as -dR --preserve=all
      --attributes-only        don't copy the file data, just the attributes
```

- For some Bash shell commands (see later), try `help command`
- To view the built-in manual page, try `man command`
- Conventions: `[ ]` indicate optional arguments, italics indicate replaceable parameters
- If you have forgotten or do not know, Google

Linux command options

Command line options

- Many commands (programs) have optional command line settings or options
- By convention, command line options appear as the first argument(s)
- Two forms of options exist, long-form and short-form options
- Long options start with two hyphens, “--”, followed by a word
- Short-form options start with one hyphen, “-”, followed by one letter or digit
- By convention, short-form options can be combined, usually in any order:
  options in `ls -a -l -F` can be combined as `ls -alF` or `ls -laF` etc
- Most short-form options have a corresponding long option: `ls -a` is the
- same as `ls --all`, but `ls -l` is `ls --format=long`
- Some options have arguments which may be optional: `tail -n 20 myfile`
  or `tail --lines=20 myfile`

Simple commands

- `cd ~` # change directory to my home directory (/Home/johnsmith, but system dependent). The tilde is the shorthand for the logged-in users home directory
- `cd ~/myscripts/src/dna` # Change directory to ~/myscripts/src/dna
- `ls` # List the contents of the directory that I am currently “in”
- `cd src; ls` # Multiple commands can go on one line, separated by “;”
- `pwd` # show the current working directory (directory that I am in)
- `ls --help` # Over five pages of help on the “ls” command
- `man ls` # SPACE or PAGEDOWN for the next page, “q” to quit
- `ls -a -l` # “-a”: also list files starting with “.”; “-l”: more detailed format
- `ls -al` # Combining command line options
- `ls --all -l` # Mixing long and short-form options

HPC2 at Stellenbosch

- https://sun.ac.za/hpc

The HPC currently has the following compute specifications

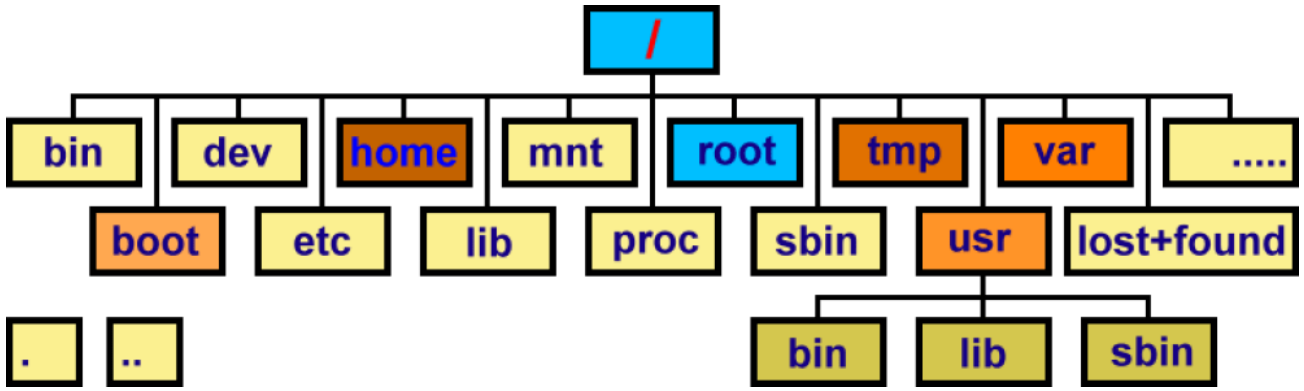
- 1x 80-core Intel Xeon E7-4850 @ 2.00GHz with 1024GB RAM, Infiniband interconnect
- 3x 48-core Intel Xeon E5-2650 v4 @ 2.20GHz with 512GB RAM, Infiniband interconnect
- 2x 64-core AMD Opteron 6274 @ 2.20GHz with 128GB RAM, Infiniband interconnect
- 2x 24-core Intel Xeon X5650 @ 2.67GHz with 48GB RAM, Infiniband interconnect
- 3x 16-core Intel Xeon E5530 @ 2.40GHz with 24GB RAM

The total is 448 available cores

Pathnames

Absolute pathnames

- Any file or directory can be uniquely represented by an absolute path
- gives the full name of the file or directory
- starts with the root “/” and lists each directory along the way
- has a “/” to separate each directory in the path
- Example /home/johnsmith/workshop/spades_data



Pathnames

Relative pathnames

When a program (command) is running, it is called a process

- Every process has a current working directory (the directory I am currently “in”)
- When you log in, the system sets your current working directory to your home directory, something like /home/johnsmith (highly system dependent)
- Any process can change its current working directory (“cd directory”) at any time
- A relative pathname points to a path that starts at the *current directory*
    - does not start with “/”
    - path components are still separated with slashes “/”
- Current directory is denoted by “.” (dot)
- The directory above the current one (parent directory) is denoted by “..” (dot-dot)

Examples of relative paths

Assume current directory is /home/hpatterton

Relative path	Absolute path
test1.py	/home/johnsmith/test1.py
t_brucei/kinetoplast/chromosome.ini	/home/johnsmith/t_brucei/kinetoplast/chromosome.ini
spades_test/contigs.fasta	/home/johnsmith/spades_test/contigs.fasta
./spades_test/contigs.fasta	/home/johnsmith/spades_test/contigs.fasta
../file.txt	/home/file.txt

Bourne again shell (bash)

Official manual page entry (“man bash”)

- Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell
- Bash is an SH-compatible command language interpreter that executes commands read from the standard input (typically, the keyboard) or from a file
- Bash is a command processor that runs in a text window where the user types commands
- Bash can also read and execute commands from a file, called a shell script
- Like all Unix shells, it supports filename globbing (wildcard matching), piping, command substitution, variables, and control structures for condition-testing and iteration.
- Interprets your typed commands and executes them
- Just another Linux program, started by the system when you log in

Bourne again shell (bash)

Some features of Bash

- Powerful command line with shortcuts to make things easier
- Tab completion (press the TAB key to complete commands and pathnames, TAB TAB
- to list all possibilities)
- Command line editing:
  - ↑ (Up-Arrow) to recall previous commands
  - CTRL-R (C-R or^R) to search for previous commands
  - ← and → to move along current command line
- Bash is a full programming and scripting language:
  - Variables and arrays
  - Loops (for; while; until), control statements (if … then … else; case)
  - Functions and co-processes
  - Text processing (“expansion” and “parameter substitution”)
  - Simple arithmetic calculations
  - Input/output redirection (e.g., redirect output to files)

File and directory patterns

- The Bash shell interprets certain characters in the command line by replacing them with matching pathnames
- Called pathname expansion, pattern matching, wildcards or “globbing”
- At the start of a filename: “~” is replaced with your home directory, “~user” is replaced with the home directory of user *user*
- For existing pathnames: “*” matches any string, “?” matches any single character, “[abc]” matches any one of the enclosed characters (in this case, “a”, “b” or “c”)
- Glob patterns “*”, “?” and “[…]” only match existing pathnames
- Even for pathnames that do not exist: “{alt1,alt2,…}” lists alternatives, “{n..m}”
- lists all numbers between n and m, “{n..m..s}” from n to m in steps of s (brace expansion)

File and directory names

- Linux allows any characters in a filename except “/” and NUL
- You may create filenames with unusual characters in them:
  - spaces and tabs
  - starting with “-”: conflicts with command line options
  - question marks “?”, asterisks “*”, brackets and braces
  - other characters with special meanings: “!”, “$”, “&”, “#”, “'”, etc.
- To match such files: use the glob characters “*” and “?”
- Linux file systems are case-sensitive: README.TXT is different from readme.txt, which is
- different from Readme.txt and ReadMe.txt!
- File type suffixes (e.g., “.txt”) are optional but recommended. Use the suffix to remind yourself what the format of a file is (fasta, fastq, bam, vcf, etc.)
- Filenames starting with “.” are usually hidden from globs and ls output

- Use “a”-“z”, “A”-“Z”, “0”-“9”, “-”, “_” and “.” only

Managing directories

- To create a directory: “mkdir dir”
- To create intermediate directories as well: “mkdir -p dir”
- To remove an *empty* directory: “rmdir dir”

Exercise

```
cd; ls                  # Change to your home directory and list its contents (should be empty)
mkdir test1             # Create the directory test1
cd test1                # … and change to it
mkdir sub{1,2,3}        # What does this do?
mkdir ../test2          # Where is the directory test2 created?
cd ../test2             # Change to it
mkdir sub{04..10}       # How to make lots of subdirectories in one go!
cd ~                    # Go back to the home directory
```

Managing files

Make a new, “empty file” in the working directory: `touch filename`
To output one or more file’s contents: `cat filename …`

- To view one or more files page by page: `less filename …`
- To copy one file: “`cp source destination`”
- To copy one or more files to a directory: “`cp filename dir`”
- To preserve the “last modified” time-stamp: “`cp -p`”
- To copy recursively: “`cp -pr source destination`”
- To move one or more files to a different directory: “`mv filename dir`”
- To rename a file or directory: “`mv oldname newname`”
- To remove files: “`rm filename`”

Recommendation: use “`ls filename …`” before `rm` or `mv`: what happens if you accidentally type “`rm *`”?

Managing files and directories

- To copy whole directory trees: `cp -pr filename destination`
- To copy to and from another Linux or Mac OS X system, use secure copy: `scp [-p -r] source destination`
- Either source or destination (but not both) can contain a remote system identifier followed by a colon: `[user@]hostname:`
- Can use rsync: `rsync -vax [--delete] [--dry-run] srcdir/ destdir/`
- Powerful command but tricky! Note the trailing “/” on the directory arguments

Permissions

- Each file can be made to be readable, writeable or executable
- Permissions on who can do what to files and in directories are set with the *chmod* command
- Look at the permissions returned by the ls -l command:
  - `-rw-r--r--`
- Note that for directories the permission indicate that it is a directory with the set permissions
  - `drw-r--r--`
- For a file, the permissions pattern can be divided into the directory letter and 3 groups of 3 letters each
  - `-` `rw-` `r--` `r--`
- The first group of 3 letters refer to the user (u), the 2nd group to the group (g), and the 3rd group to other (o)
- Within each group of 3 letters, each letter can be either on or off
- Imagine the value of $r=2^2=4$, $w=2^1=2$ and $x=2^0=1$ (see the pattern?; value 0-7, octal numbers)
- Thus, if you want to set the user permissions to rwx, the value is 4+2+1=7. If you wanted to set the group of 3 letters to rw-, the value is 6
- So if you wanted to set u to rwx, g to rw- and o to r--, it is represented by the numbers 7, 6 and 4
- Use the command `chmod 764 filename`
- Generally files are set to 644 (rw-r--r--)
- You can right-click on a file in the MobaXterm GUI to set its permission

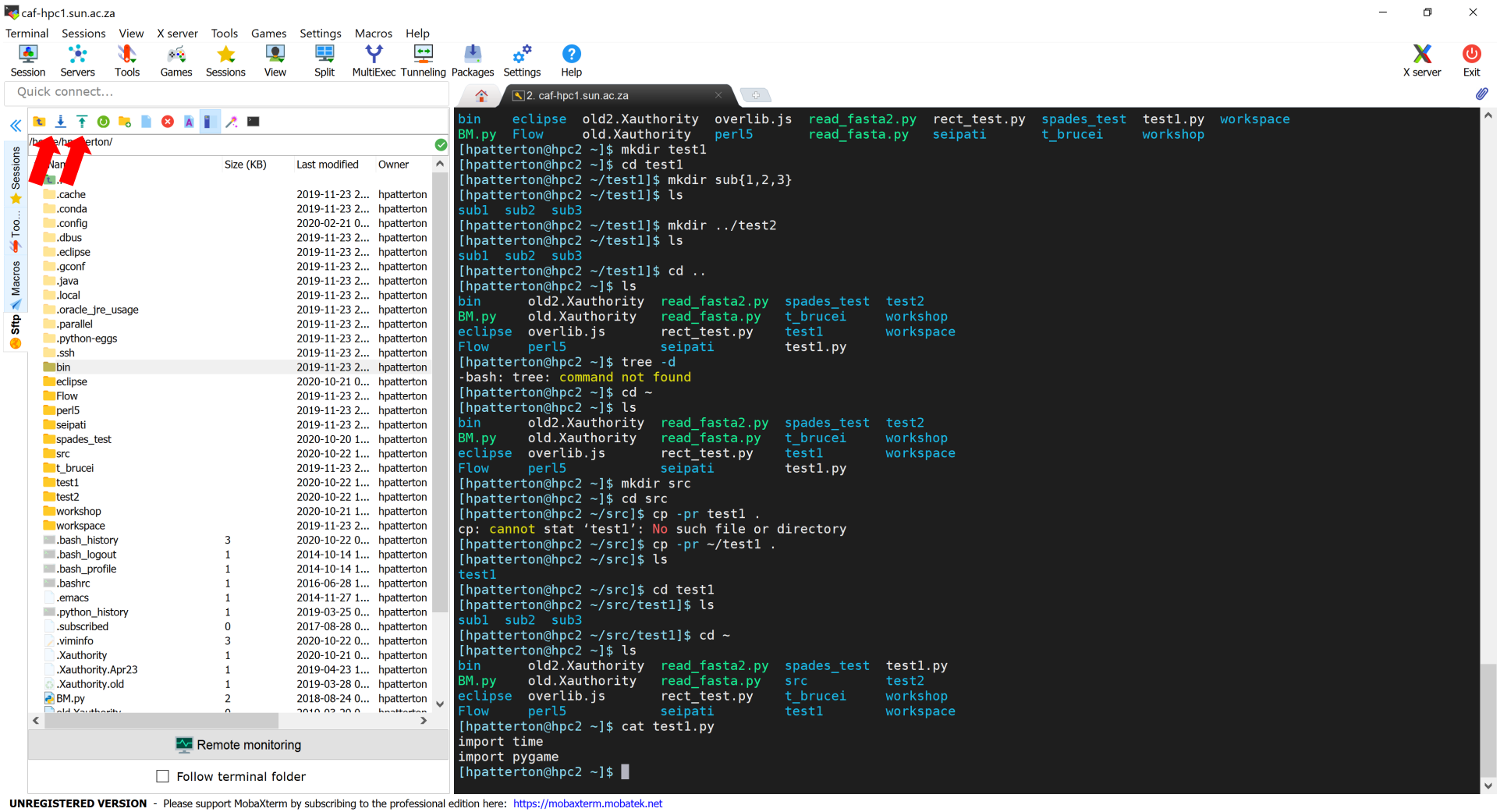
Playing with pathname expansion

Exercize

```
cd ~; mkdir src; cd src
cp -pr ~/test_directory .        # Copy directories recursively to "." (current directory)
cd test_directory                # Change to the newly copied directory
cat ~/myprogram1.py              # Display the contents of this file
ls */*.c                         # List all files matching "*/*.c"
rm */*.c                         # … and then remove them!
ls */*.c                         # What happens now?
mv README my-new-filename        # Rename the README file
cp INSTALL new                   # Make a copy of INSTALL and call it "new"
ls -l INSTALL new                # What is the difference between the listings?
cp -p INSTALL same               # Copy INSTALL, preserving time-stamps
ls -l INSTALL same               # Verify the two files have the same date and time
```

Transferring files to and from the HPC

- MobaXterm has easy GUI



Redirecting input and output

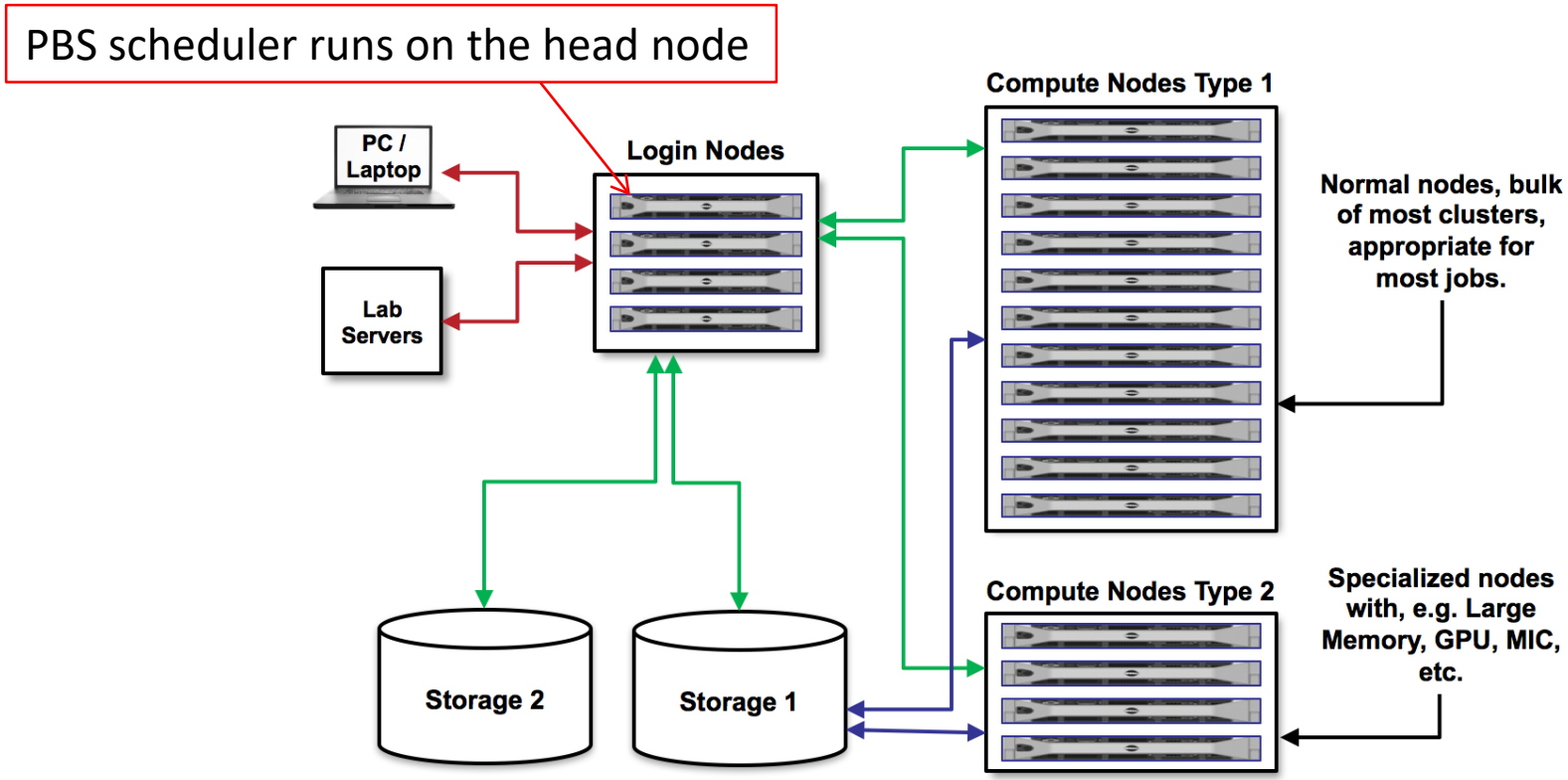
- Standard input, standard output and standard error can be redirected to/from a file or even “piped” to another program
- To redirect output to file, use “`>filename`”
- To append output to file, use “`>>filename`”
- To redirect input from filen, use “`<filename`”
- To connect the output from one program to the input of another (a pipe), use “`program1 | program2`”
- To redirect output to both a file and the screen, use “`| tee filename`”
- Multiple pipes are allowed: “`program1 | program2 | … | programN`”

Playing with file redirection

Exercise

```
cd ~
ls > ~/dir-list1.txt                    # Redirect the output of ls to ~/dir-list1.txt
cat ~/dir-list1.txt                     # Show what is in that file
ls spades_test >> ~/dir-list1.txt       # Append the output of “ls src” to ~/dir-list1
cat ~/dir-list1.txt                     # What does the file contain now?
wc -l < ~/dir-list1                     # Run “wc -l” (count lines in a file), but use ~/dir-list1.txt
cat ~/dir-list1 | wc -l                 # Use a pipe from cat to wc (output of cat becomes input of wc)
```

Portable Batch System (PB)



- A program is run on the HPC by submitting a command to the “scheduler”
- This command is part of a list of instructions (a script), requesting resources and setting other options
- The PBS scheduler places the instructions (“job”) in a queue depending on requested resources
- When the resources are available, the job is initiated (program is run) on the assigned node with the requested number of CPUs and memory

Simple scripting

- Shell scripts are just files containing a list of commands to be executed
- First line (“magic identifier”) must be `#!/bin/bash`
- Comments are introduced with `#`

Variables

- To set a variable, use `varname=value` (no spaces)
- To use the contents of a variable, use `$varname` or `${varname}`
- Variable names start with a letter, may contain letters, numbers and _
- Variable names are case-sensitive

Simple scripting, continued

For loops

```
for variable in list …; do
 process using ${variable}
Done
```

Conditional statements (multiple “elif” allowed; “elif” and “else” clauses are optional)

```
if [ comparison ]; then                    # Use literal “[” and “]” characters
        if-true statements
elif [ second-comparison ]; then
        if-second-true statements
else
        if-all-false statements
fi
```

Simple scripting, continued

While loops

```
while [ comparison ]; do
        while-true statements
Done
```

Until loops

```
until [ comparison ]; do
        while-false statements
done
```

Examples of comparisons

- `string1 = string2` # strings string1 and string2 are equal
- `number1 -lt number2` # number1 is less than number2
- `file1 -nt file2` # file1 (e.g., a data file) is newer than file2 (e.g., output file)
- See the manual page for test ("`man test`") for more information

Creating your first script

- Launch the MobaXterm MobaTextEditor utility
- Enter the following text

```
#!/bin/bash
#PBS -l walltime=00:05:00
echo "I am user $(whoami), running on $(hostname)"
```

- Save the text file as *myscript1.pbs* (or another name) on your computer
- Use “pbs” (or another meaningful term) as the filename extension for all your script files, so you know what they are when you look at a directory listing
- Click on the “Upload” icon in MobaXterm
- Select the file that you have just saved, and click “OK”
- The file is uploaded to your current working directory on the HPC

- Run the script by typing `qsub myscript1.pbs` in the terminal window
- Note the assigned job number (for example, 76957)
- Refresh the MobaXterm file manager window, and find `myscript1.pbs.o76957`
- It contains the text “I am user johnsmith, running on n05.hpc”
- Viola! Your first script!

Where did my output go?

PBS automatically redirect standard input, standard output and standard error

- standard input from /dev/null
- standard output to script_filename.ojob_number in the current working directory
- standard error to script_filename.ejob_number in the current working directory

Running programs on the HPC

- There are usually many versions of an app on the HPC
- How do you choose which one to use?
- Applications are managed using the **module system**
- On the HPC applications are stored in the /apps directory
- Module files are stored in the /app/Modules/modulefiles directory
- Module files set shell environment variables such as `PATH`
- `PATH` controls where applications are searched for (the search path)
- To see available applications enter `module avail`
- To see currently loaded applications enter `module list`
- To load an application enter `module load app/application[/version]`
- To unload an application enter `module unload application`

Finding the programs

Exercise

```
module avail                          # What applications are available?
module list                           # What applications are currently loaded?
echo $PATH                            # See the current value of the PATH variable
module load app/SPAdes/3.14.0         # Set the PATH to include SPAdes
echo $PATH                            # What does PATH look like now?
module unload app/SPAdes              # We don't want to use spades any more…
echo $PATH                            # PATH no longer contains the SPAdes directory
```

PBS commands

- PBS is designed to manage the distribution of batch jobs and interactive sessions across the available nodes in the cluster

Job Control

- `qsub`: Submit a job
- `qdel`: Delete a batch job
- `qsig`: Send a signal to batch job
- `qhold`: Hold a batch job
- `qrls`: Release held jobs
- `qrerun`: Rerun a batch job
- `qmove`: Move a batch job to another queue

Job Monitoring

- `qstat`: Show status of batch jobs
- `qselect`: Select a specific subset of jobs

Node Status

- `pbsnodes`: List the status and attributes of all nodes in the cluster

Manual: https://albertsk.files.wordpress.com/2011/12/pbs.pdf

Creating a script

To submit a job to the cluster

- Create a shell script file
- Add `#PBS` directives as required directly after `#!/bin/bash`
- Add `cd $PBS_O_WORKDIR` after the `#PBS` directives
- Submit the script to the PBS scheduler with `qsub script_filename.pbs`
- Wait for the job to run, checking its status with `qstat`
- If you have not submitted a job using `qsub`, you are almost certainly running your job on the resource-scarce head node!
- Running jobs on the head node does not use the processing power of the HPC

Common PBS directives

Some common #PBS directives

• `#PBS -N scriptname`	# Set a name for the script
• `#PBS -P project`	# Charge resources from this project
• `#PBS -q queuename`	# Which queue to submit to
• `#PBS -l ncpus=n`	# Request n processor cores in total
• `#PBS -l ngpus=n`	# Request n GPUs
• `#PBS -l walltime=hh:mm:ss`	# How much time is required for running the job
• `#PBS -l mem=sizeMB`	# How much memory is required (in MB)
• `#PBS -l software=licname`	# Use software licence licname
• `#PBS -M email`	# Send notifications to the email address
• `#PBS -m abe`	# What notifications to send by email
• `#PBS -l wd`	# Run from the same directory as submission

An example script

Here is an example of a script that requests 1 hour of execution time, renames the job to 'My-Program', and sends email when the job begins, ends, or aborts:

```
#!/bin/bash
#PBS -N My-Program                  # Name of my job
#PBS -l walltime=1:00:00            # Run for 1 hour
#PBS -e myprog.err                  # Where to write stderr
#PBS -o myprog.out                  # Where to write stdout
#PBD –M johnsmith@company.com       #Where to send e-mails
#PBS -m abe                         # Send email when my job aborts, begins, or ends
cd $PBS_O_WORKDIR                   # This command switched to the directory from which the
                                    qsub command was run
Module load /app/myprog/1.0.0       # load  myprog version 1.0.0
myprog –options argument1           # execute my program
```

Getting e-mails when your job aborts, starts and ends

Include the following in your script

#PBS –M e-mail_address
#PBS –m abe #specify when to send e-mail: abort (a), begin (b) or end (e)
E-mail when job finishes:

```
PBS Job Id: 76814.hpc2.hpc
Job Name:   small_job
Execution terminated
Exit_status=0
resources_used.cpupercent=101
resources_used.cput=00:00:09
resources_used.mem=0kb
resources_used.ncpus=1
resources_used.vmem=0kb
resources_used.walltime=00:00:05
```

0 means no errors were generated

Using disk space on the compute node

- If your job makes frequent writes to disk, it may improve performance to use scratch space on the compute node instead of continuously move data between your home directory and the compute node

```
#!/bin/bash
#PBS -l nodes=1:ncpus=1
#PBS -l walltime=5:00
#PBS -N small_job
TMP=/scratch/${PBS_JOBID}                    # use /scratch for compute node and /scratch2 for storage node
mkdir -p ${TMP}                              # make the new directory
/usr/bin/rsync -vax "${PBS_O_WORKDIR}/" ${TMP}/      # copy everything from your working directory to
                                                     the new directory on /scratch on the compute node
cd ${TMP}                                    # change directory to new directory
module load app/fastqc                       # load the default fastqc application environment
fastqc ${TMP}/R1.fastq.gz ${TMP}/R2.fastq.gz         # execure the application, reading the input files from the
                                                     new directory on /scratch
/usr/bin/rsync -vax ${TMP}/ "${PBS_O_WORKDIR}/"      # copy everything back from the directory on
                                                     /scratch to your working directory
[ $? -eq 0 ] && /bin/rm -rf ${TMP}           # if the copy succeeded, delete temporary files
```

`$?` is the end status of the last process to execute, and is 0 if the command was successful (without any errors)
Thus, if `rsync` finished successfully, [$? -eq 0] evaluates to true, the command after the && executes, i.e., everything in directory TMP is removed

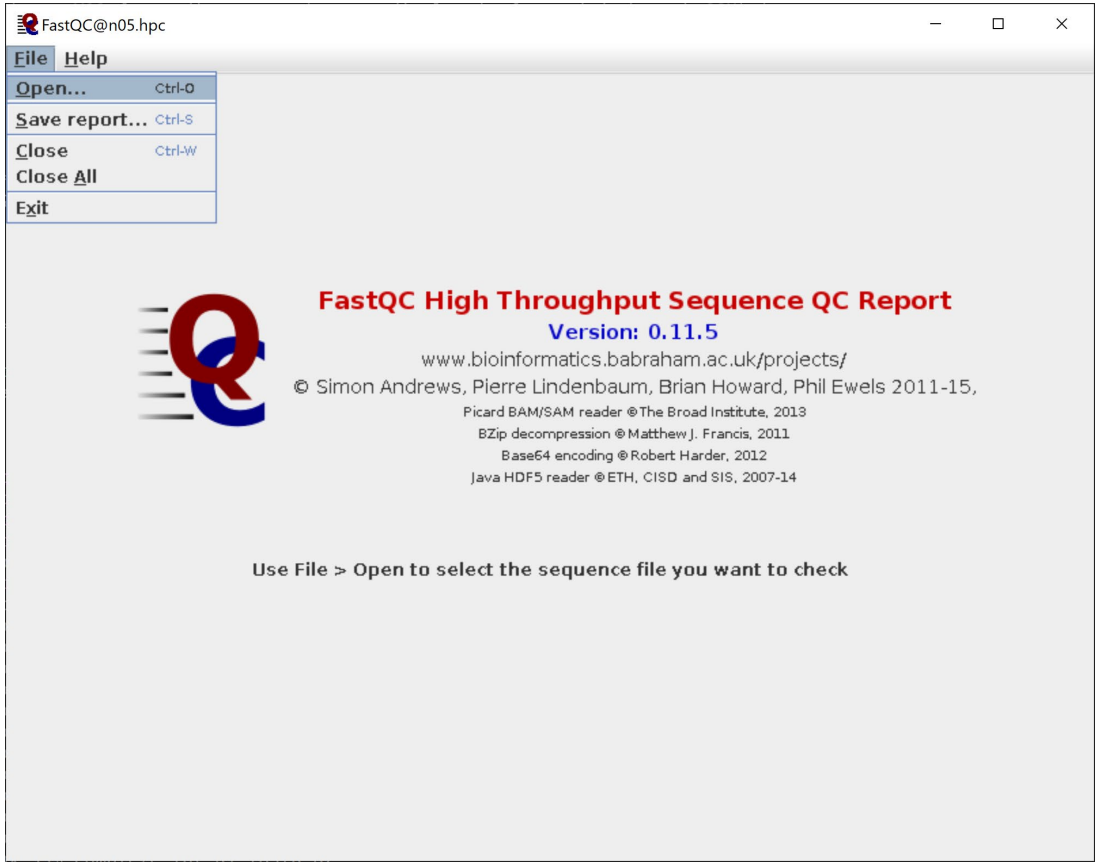
Interactive jobs

- You can perform jobs interactively on the HPC
- Use the commend **qsubi** or **qsubix**
- The qsubix passes additional information to use an X11 window for output where this is supported
- **Note qsubi uses the head node which is resource scarce**

```
[hpatterton@hpc2 ~]$ qsubi
Requesting interactive session with 4 cores, 8GB RAM, for 24 hours
qsub: waiting for job 76811.hpc2.hpc to start
qsub: job 76811.hpc2.hpc ready
[hpatterton@n05 ~]$ module load app/fastqc
[hpatterton@n05 ~]$ fastqc ~/workshop/spades_data/R1.fastq.gz
Started analysis of R1.fastq.gz
Analysis complete for R1.fastq.gz
[hpatterton@n05 ~]$
```

Interactive jobs

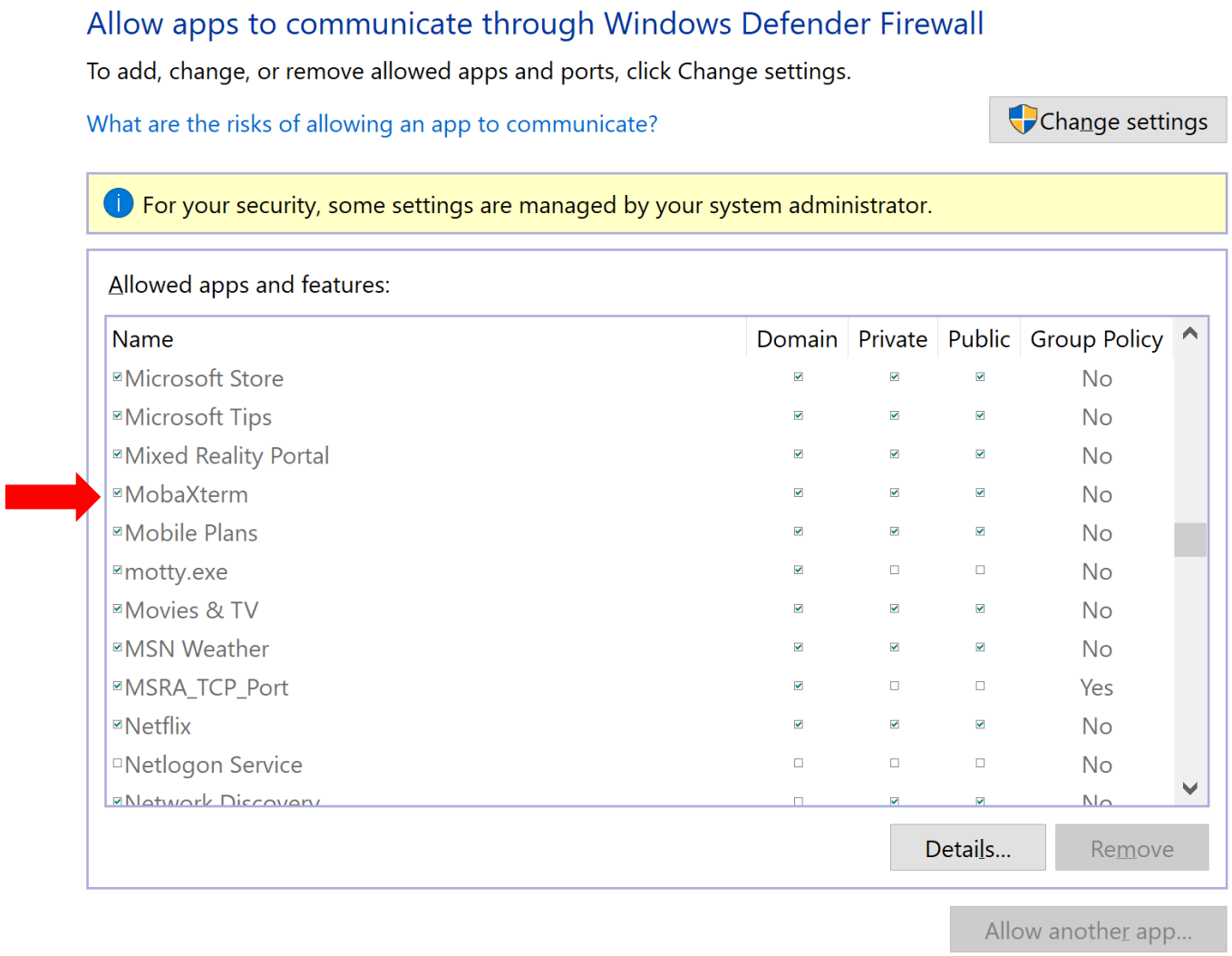
```
[hpatterton@hpc2 ~]$ qsubix
Requesting interactive session with 4
cores, 8GB RAM, for 24 hours
qsub: waiting for job 76812.hpc2.hpc
to start
qsub: job 76812.hpc2.hpc ready

[hpatterton@n05 ~]$ module load
app/fastqc
[hpatterton@n05 ~]$ fastqc
```



Note: when using `qsubix`, make sure of your local firewall setting allows communication (next slide)

Firewall on Windows10

- Make sure MobaXterm is in the list of applications that may communicate through the firewall



Examining the queue

You can look at the queue by using the `qstat` command.
qstat will display the queue ordered by JobID.

```
[username@hpc1 ~]$ qstat
Job id              Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
32.hpc1           JobName          username          351:04:3 R long
33.hpc1           JobName          username          351:06:1 R day
34.hpc1           JobName          username          390:30:2 R week
40.hpc1           JobName          username          496:38:2 R month
46.hpc1           JobName          username          506:13:5 R long
```

Look at node loads

You can look at the load on each node using the `pestat` command.

```
[hpatterton@hpc2 ~]$ pestat
Queues:  short day week month long
Node             state    cpu        memory   jobids/users
----                    tot used    tot used
n01.hpc          free    80    0  1022G   0%
n02.hpc          free    48    0   254G   0%
n04.hpc          free    48    0   510G   0%
n05.hpc          free    48    0   510G   0%
n11.hpc          free    96    0   503G   0%
n12.hpc          free    80    0  3022G   0%
```

Look at node loads, graphically

You can get a “graphic” presentation of node loads and running processes with the `htop` command