



Introduction to HPC

June 2025

Basic course:

Introduction to Linux for HPC

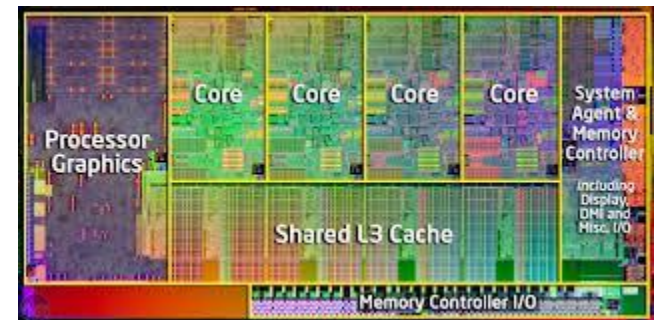
- 1) Overview of HPC: (30 min)
- 2) Linux (60 min)
- break (30 min)
- 3) Transferring files (20 min)
- 4) Submitting jobs (40 min)

Module 1: What is HPC?

30 minutes

What is HPC?

- HPC is the aggregation of computing resources.
 - Cores (cpus / chips / sockets)
 - RAM
 - Disk
 - Interconnect



What HPC isn't

- HPC is not supercomputing.
- Linux, not Windows.
 - Cost, flexibility, scalability, performance and support
 - No Windows HPC clusters in the Top500
- No GUI, point and click interface.
 - Generalized command line approach, not dedicated to a package.
 - However some apps can export X-display
 - However(2) we now have OnDemand – a GUI frontend

Architecture

- Operating system: Rocky Linux
- X86_64
- HPC server: hex.uct.ac.za (the head node)
- Scheduler: SLURM
 - Simple Linux Utility for Resource Management
 - Majority of the world's top 500 supercomputing facilities
- Worker nodes:
 - Dell C6620 48 core nodes
 - Dell C6420 40 and 44 core nodes
 - Dell C6145 64 core nodes
 - Various Dell GPU nodes

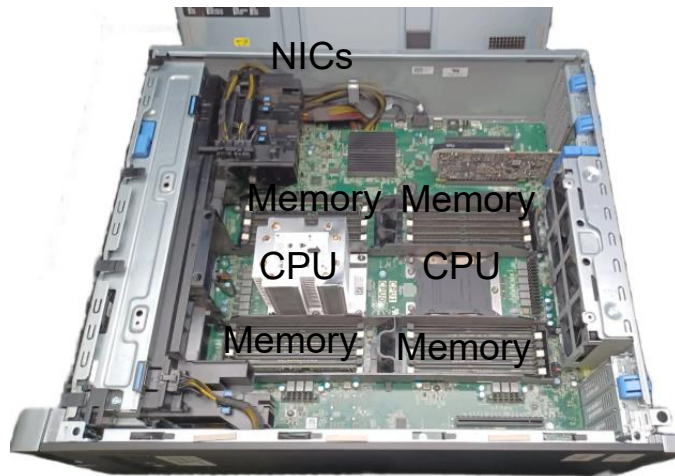
Architecture – what is a node?



A “node” is just a computer

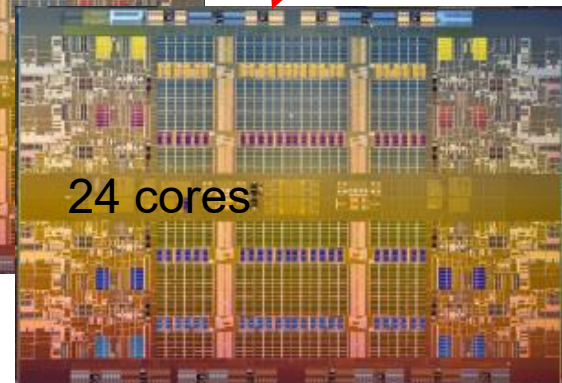
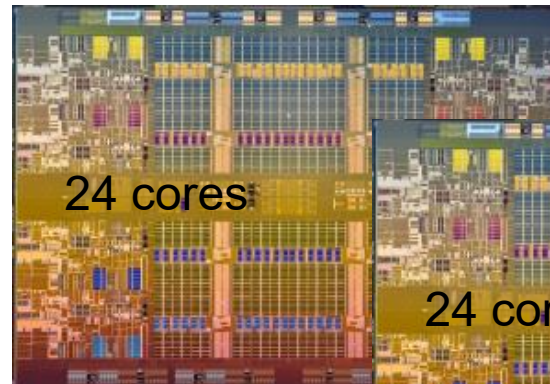
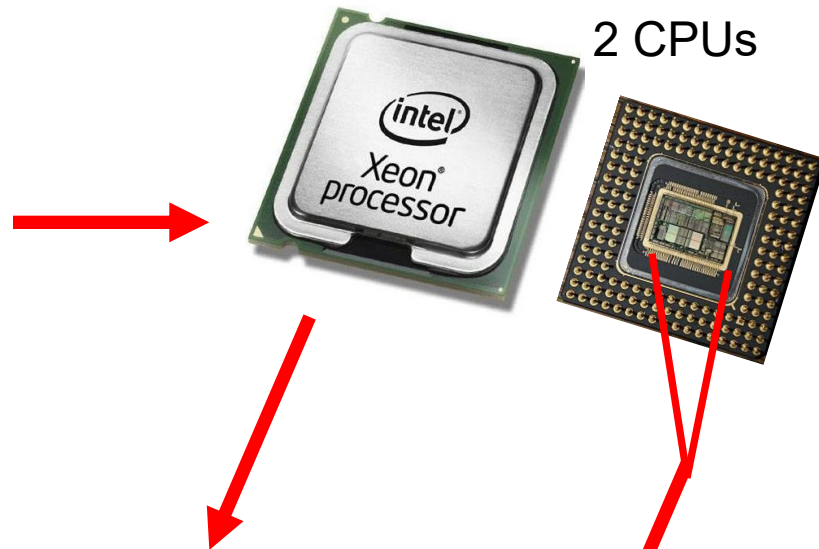


Architecture – what is a CPU\Processor\Core?

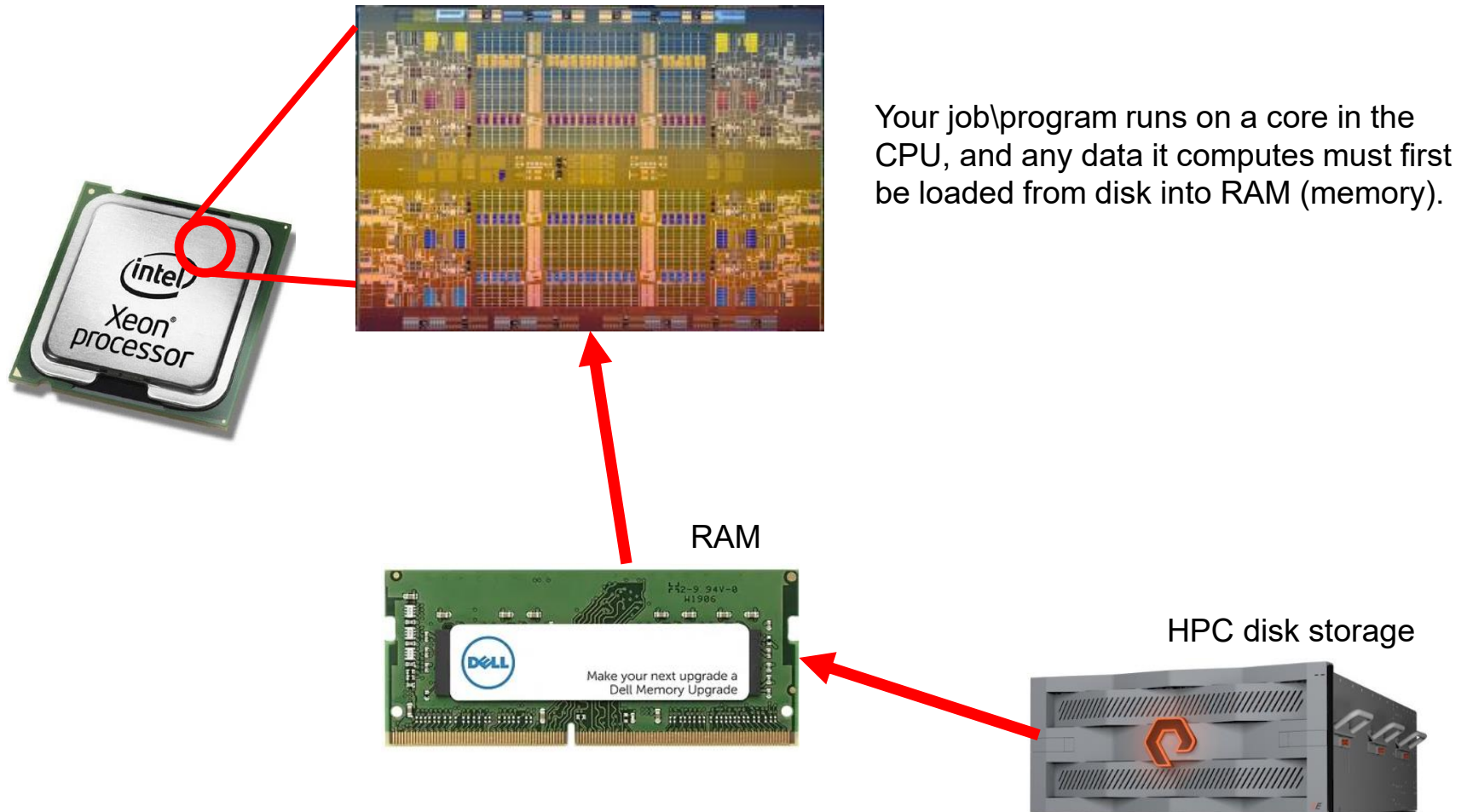


Worker node
with 48 cores

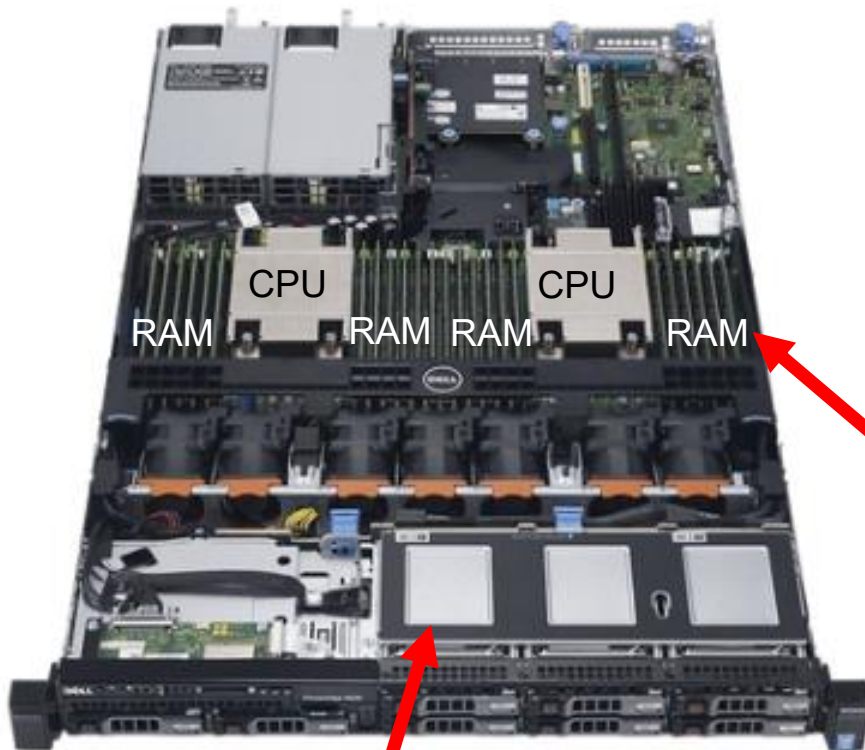
A “core” is where
your program
“thread” runs.



Architecture – what is RAM/memory/disk?



Architecture – what is RAM/memory/disk?



CPU CPU
RAM RAM RAM RAM

The CPU cannot compute on data that is on disk, only data that is in RAM.

The data must be loaded into RAM (memory).

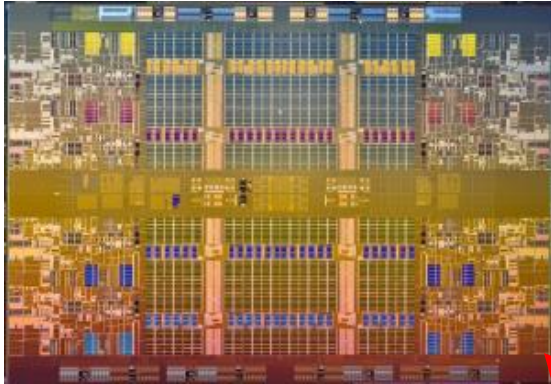
Local worker node disk storage
for operating system and /tmp

HPC disk storage
/home
/scratch



Architecture – what is RAM/memory/disk?

CPU and cores

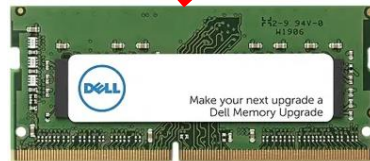


Once your data is in RAM it can be computed on.

If your job is going to do lots of reading and writing to disk it's a good idea to first copy your data to /tmp on the worker node.

Very fast

RAM (memory)



fast



Worker node's local RAID 1 SSD
hard drives for OS and /tmp

Network and
switches

slow

HPC disk storage

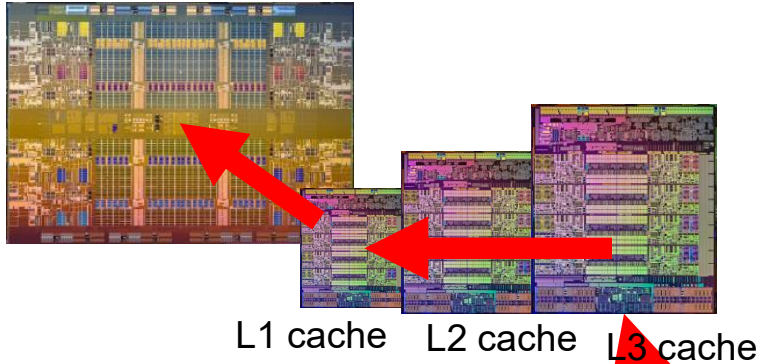
/home
/scratch



Architecture – what's faster?

CPU and cores

The CPU has its own “built in” RAM called a cache. It's used for much faster access to data the CPU determines to be more frequently called. You have little to no control over this.



RAM (memory)



fast



Worker node's local RAID 1 SSD
hard drives for OS and /tmp

HPC disk storage

/home
/scratch



These numbers are only examples!

Architecture – what's faster?

You need to worry about this side of the page.

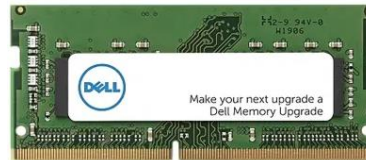
The HPC admins need to worry about this side of the page

25 μ S access time
1TB
Fairly cheap

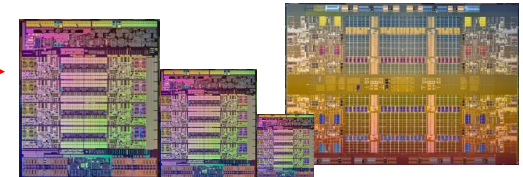


SSD local drives

50nS access time
32GB, 8 of these per CPU
Expensive



RAM (memory)



L1	L2	L3	Registers
10nS	5nS	1nS	300pS
60MB	2MB	80KB	1KB
Per CPU	Per core	Per core	Per core

NVME network drives

Access attenuated by network



10 μ S access time
Many 1TB chips
Not cheap

HPC disk storage

/home
/scratch

10mS access time
5TB
Cheap

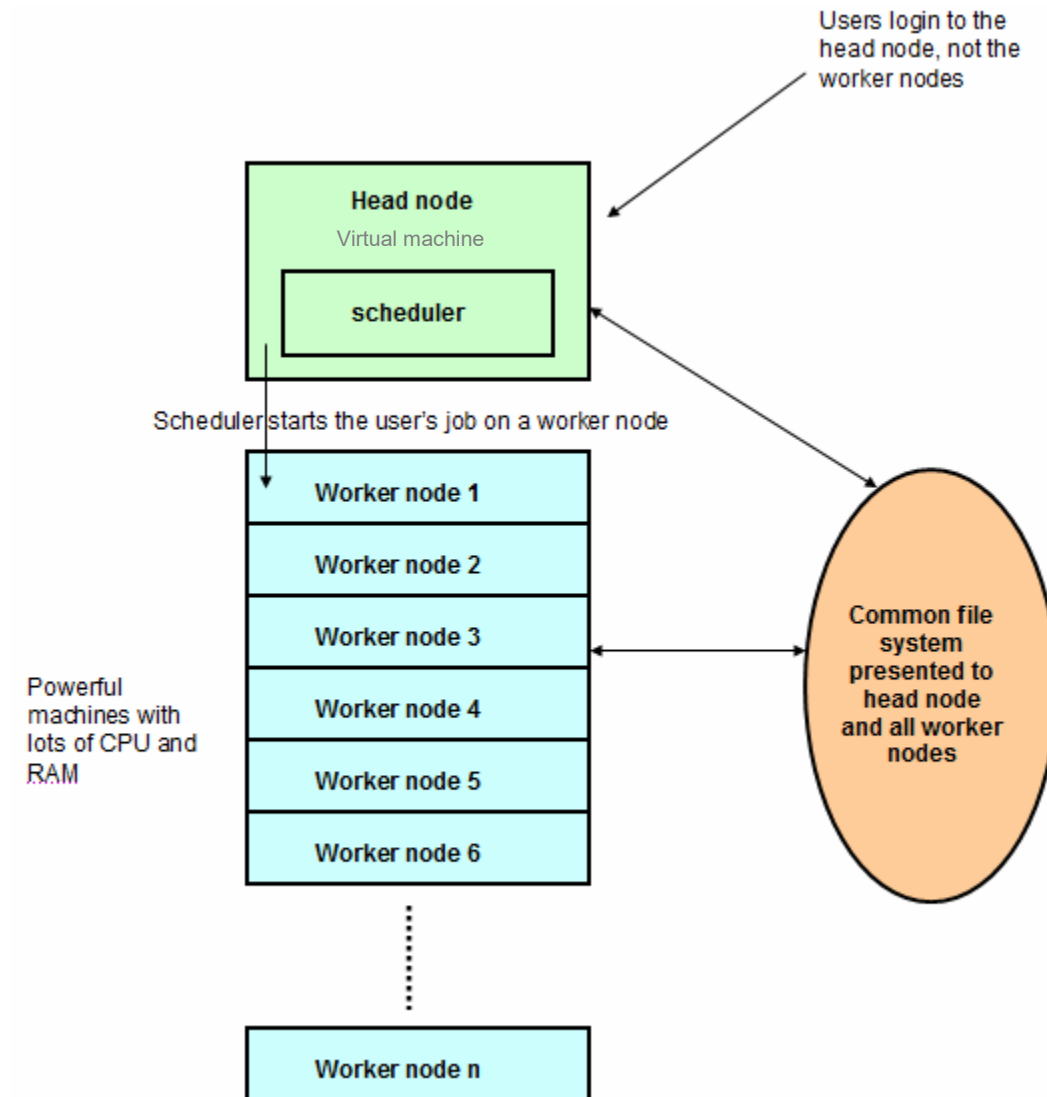


Older "spinning" disks

Architecture – RAM and cores

- RAM (memory), just like cores, is a consumable resource.
- RAM and cores don't like to be shared, it slows things down, hence we prohibit this.
- Therefore, there exists a relationship or ratio between cores and RAM.
- On the ada series it's about 8GB of RAM per core.
- If you request 10 cores you get 80GB of RAM.
- If your job exceeds this your job will be immediately cancelled by the scheduler.

Architecture - recap

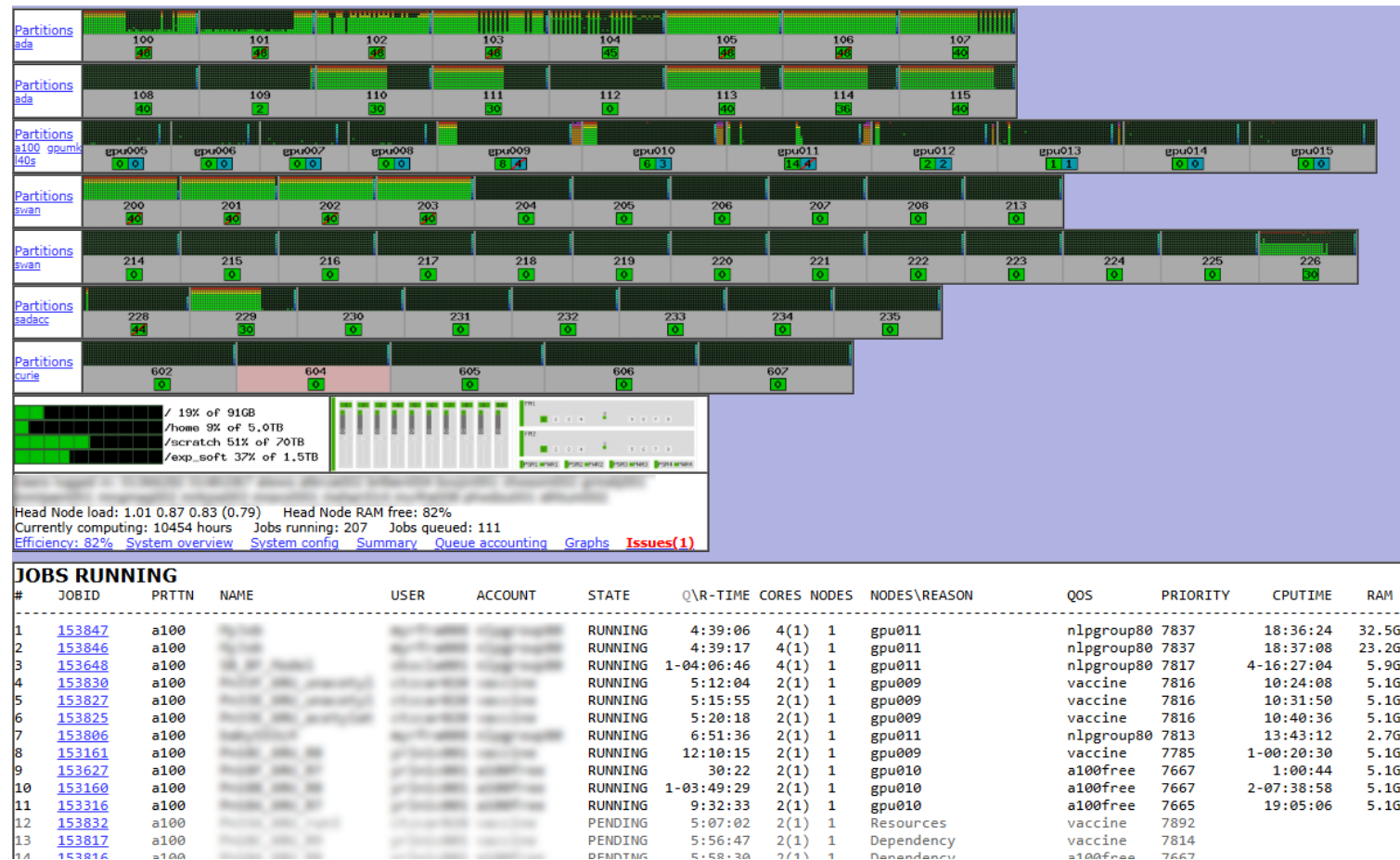


Shared file system

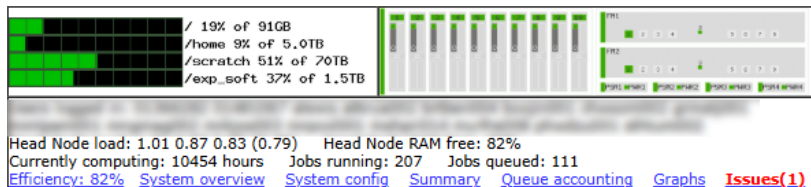
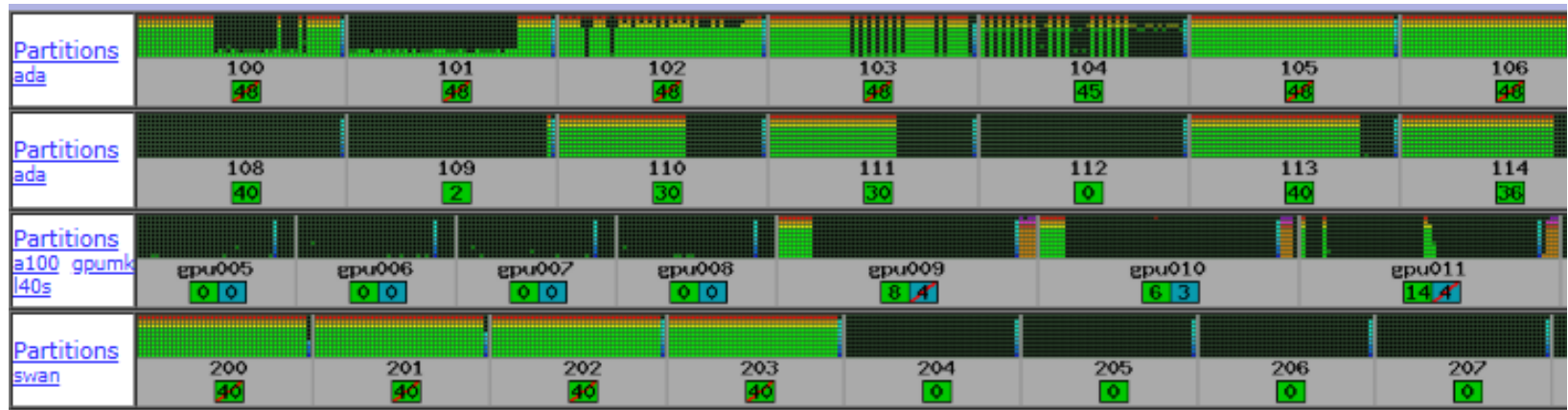
- The head node and worker nodes all have hard drives, however...
- Each server is connected to several shared disks:
 - /home
 - /opt/exp_soft
 - /scratch
- If you delete /home/\$user/mydata on a worker node it will be deleted on the head node!

The dashboard

- To keep track of the workload and the jobs that are running on the cluster go to: <https://hex.uct.ac.za/db>

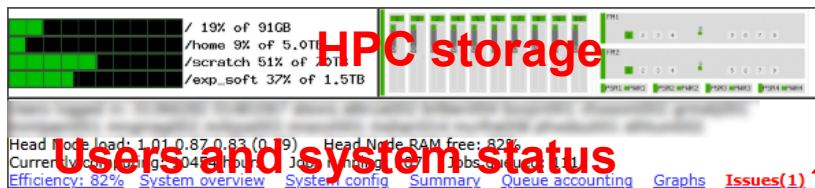
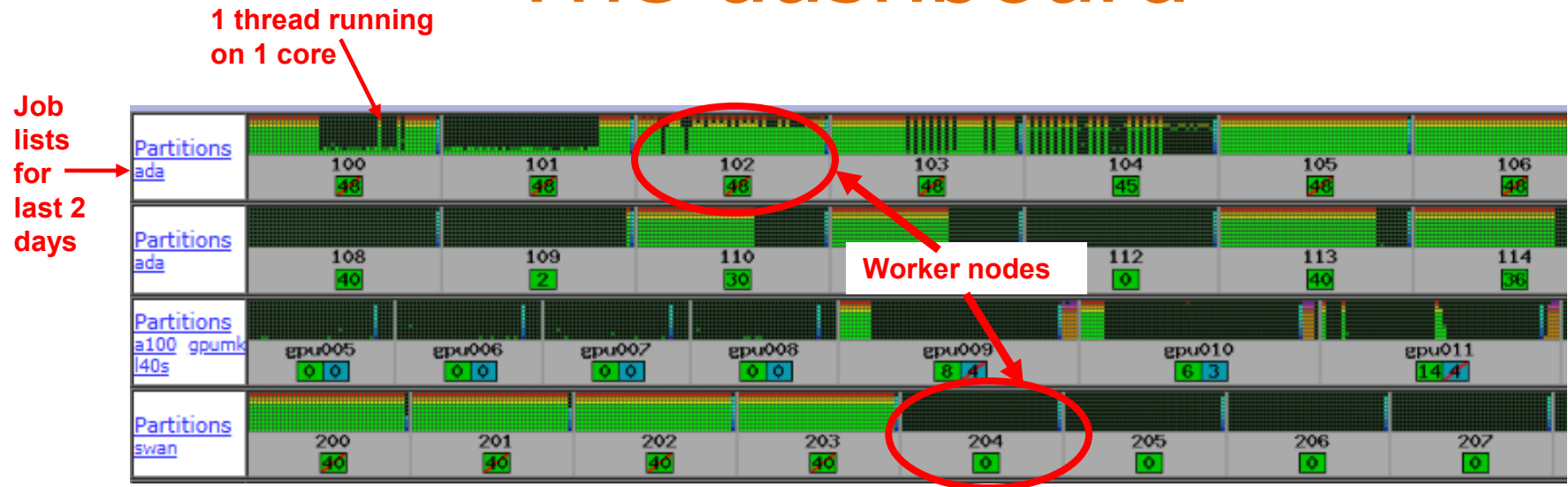


The dashboard



JOBS RUNNING														
#	JOBID	PRTTN	NAME	USER	ACCOUNT	STATE	Q\R-TIME	CORES	NODES	NODES\REASON	QOS	PRIORITY	CPUTIME	RAM
1	153847	a100				RUNNING	4:39:06	4(1)	1	gpu011	nlpgroup80	7837	18:36:24	32.5G
2	153846	a100				RUNNING	4:39:17	4(1)	1	gpu011	nlpgroup80	7837	18:37:08	23.2G
3	153648	a100				RUNNING	1-04:06:46	4(1)	1	gpu011	nlpgroup80	7817	4-16:27:04	5.9G
4	153830	a100				RUNNING	5:12:04	2(1)	1	gpu009	vaccine	7816	10:24:08	5.1G
5	153827	a100				RUNNING	5:15:55	2(1)	1	gpu009	vaccine	7816	10:31:50	5.1G
6	153825	a100				RUNNING	5:20:18	2(1)	1	gpu009	vaccine	7816	10:40:36	5.1G
7	153806	a100				RUNNING	6:51:36	2(1)	1	gpu011	nlpgroup80	7813	13:43:12	2.7G
8	153161	a100				RUNNING	12:10:15	2(1)	1	gpu009	vaccine	7785	1-00:20:30	5.1G
9	153627	a100				RUNNING	30:22	2(1)	1	gpu010	a100free	7667	1:00:44	5.1G
10	153160	a100				RUNNING	1-03:49:29	2(1)	1	gpu010	a100free	7667	2-07:38:58	5.1G
11	153316	a100				RUNNING	9:32:33	2(1)	1	gpu010	a100free	7665	19:05:06	5.1G
12	153832	a100				PENDING	5:07:02	2(1)	1	Resources	vaccine	7892		
13	153817	a100				PENDING	5:56:47	2(1)	1	Dependency	vaccine	7814		
14	153816	a100				PENDING	5:58:30	2(1)	1	Dependency	a100free	7667		

The dashboard

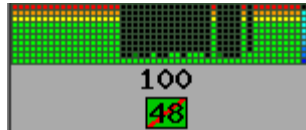


There is currently a problem on the cluster

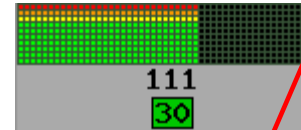
JOBS RUNNING														
#	JOBID	PRTTN	NAME	USER	ACCOUNT	STATE	Q\R-TIME	CORES	NODES	NODES\REASON	QOS	PRIORITY	CPUTIME	RAM
1	153847	a100				RUNNING	4:39:06	4(1)	1	gpu011	nlpgroup80	7837	18:36:24	32.5G
2	153846	a100				RUNNING	4:39:17	4(1)	1	gpu011	nlpgroup80	7837	18:37:08	23.2G
3	153648	a100				RUNNING	1-04:06:46	4(1)	1	gpu011	nlpgroup80	7817	4-16:27:04	5.9G
4	153830	a100				RUNNING	5:12:04	2(1)	1	gpu009	vaccine	7816	10:24:08	5.1G
5	153827	a100				RUNNING	5:15:55	2(1)	1	gpu009	vaccine	7816	10:31:50	5.1G
6	153825	a100				RUNNING	5:20:18	2(1)	1	gpu009	vaccine	7816	10:40:36	5.1G
7	153806	a100				RUNNING	6:51:36	2(1)	1	gpu011	nlpgroup80	7813	13:43:12	2.7G
8	153161	a100				RUNNING	12:10:15	2(1)	1	gpu009	vaccine	7785	1-00:20:30	5.1G
9	153627	a100				RUNNING	30:22	2(1)	1	gpu010	a100free	7667	1:00:44	5.1G
10	153160	a100				RUNNING	1-03:49:29	2(1)	1	gpu010	a100free	7667	2-07:38:58	5.1G
11	153316	a100				RUNNING	9:32:33	2(1)	1	gpu010	a100free	7665	19:05:06	5.1G
12	153832	a100				PENDING	5:07:02	2(1)	1	Resources	vaccine	7892		
13	153817	a100				PENDING	5:56:47	2(1)	1	Dependency	vaccine	7814		
14	153816	a100				PENDING	5:58:30	2(1)	1	Dependency	a100free	7667		

Currently running jobs

The dashboard



All 48 cores reserved, not fully used though



30 cores fully utilized, 18 cores still free

Blue bar is available RAM. This bar drops as RAM is used.



Node offline for maintenance

GPU server, only 3 cards in use



GPU server, all 4 cards in use

Module 2: Linux

60 minutes

What is a Shell?

- The shell is a command line interpreter or shell that provides an interface to the Linux operating system.
- A shell has a single purpose – to allow users to enter commands to execute, or create scripts containing commands, to direct the operation of the computer.
- Various types of shells available for your preference, examples “ csh, ksh, bash, zsh, pdsh, tcsh the list goes on “

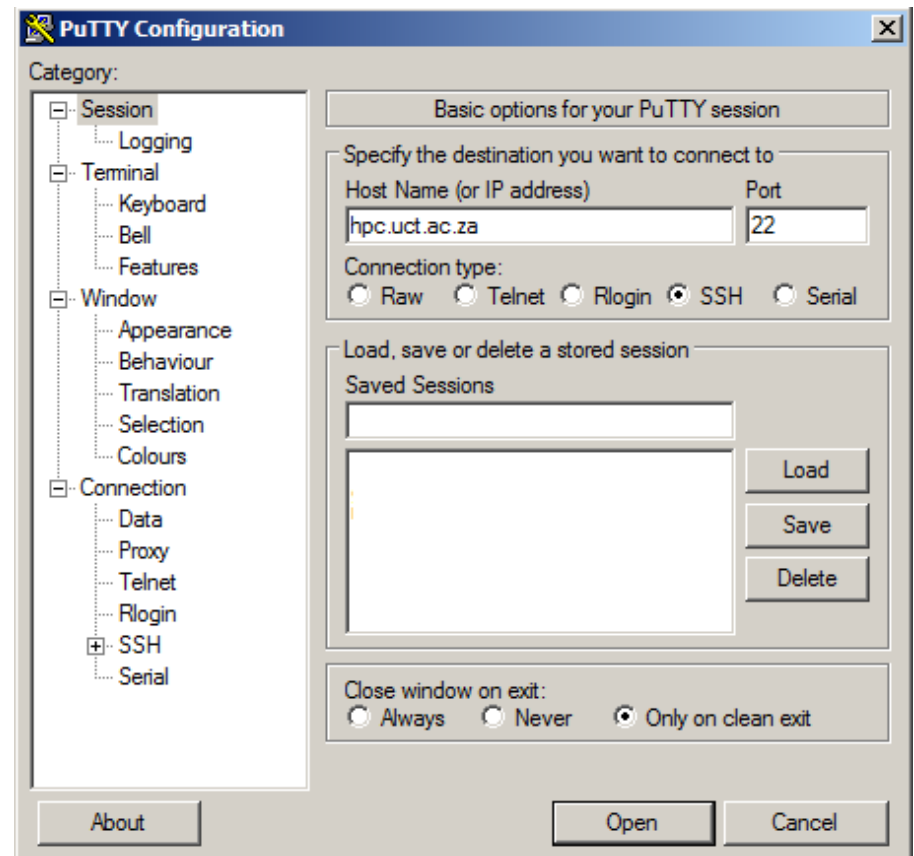
Install software

Use your web browser to download Putty and PuttySCP from:
<http://www.putty.org>

- Click on the “Download Putty” link and download:
 - putty.exe (a Telnet and SSH client)
 - pscp.exe (an SCP client, i.e. command-line secure file copy)
 - WinSCP (GUI-BASED SCP)
- Double click to install on your PC.

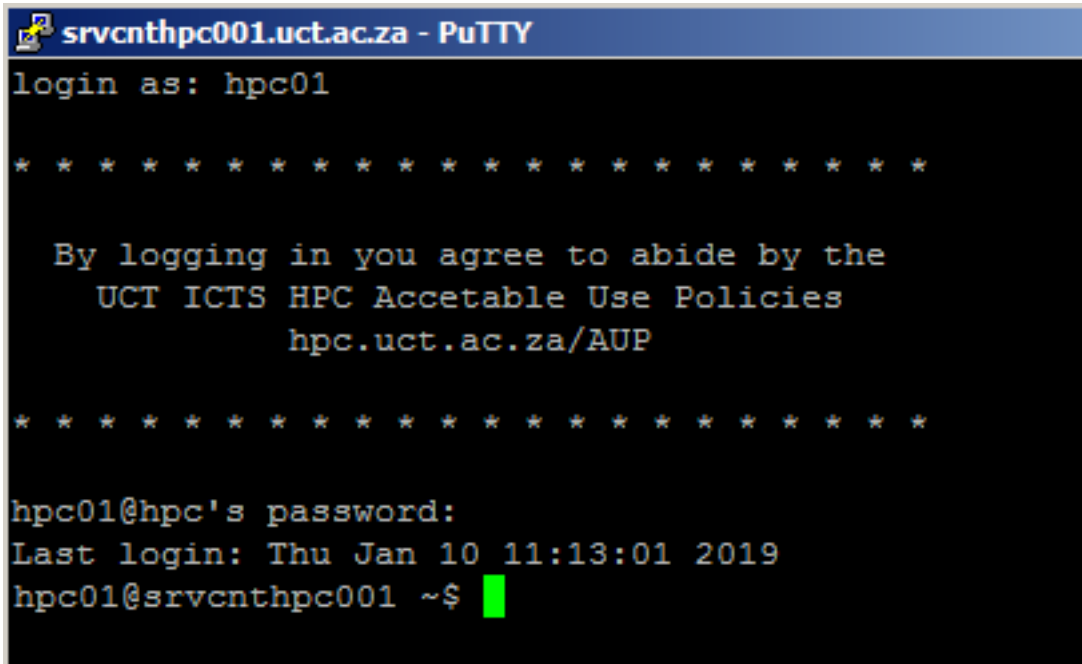
Logging On

- Start the putty telnet/ssh client by double clicking on putty.exe and connect to the HPC Machine
 - Host: **hpc.uct.ac.za**
 - Connection Type: **ssh**
 - Port: **22**



Log On to the HPC Machine

- Log into the training HPC system using the Test Account allocated to you, e.g.
 - **Account Name:** hpc01
 - **Password:** [no characters will appear as you type]



```
srvcnthpc001.uct.ac.za - PuTTY
login as: hpc01

* * * * *

By logging in you agree to abide by the
UCT ICTS HPC Accetable Use Policies
hpc.uct.ac.za/AUP

* * * * *

hpc01@hpc's password:
Last login: Thu Jan 10 11:13:01 2019
hpc01@srvcnthpc001 ~$
```

What does BASH do?

- The shell (command line interpreter) interprets the commands entered by the user and passes those to the Linux operating system.
- When you enter a command and hit return
 - BASH parses the command line into tokens
 - 1st token is interpreted as the **command**
 - Remaining tokens are interpreted as **arguments**

Anatomy of a command

- Commands comprise of:
 - a **command** that invokes a program
 - **arguments** to that program

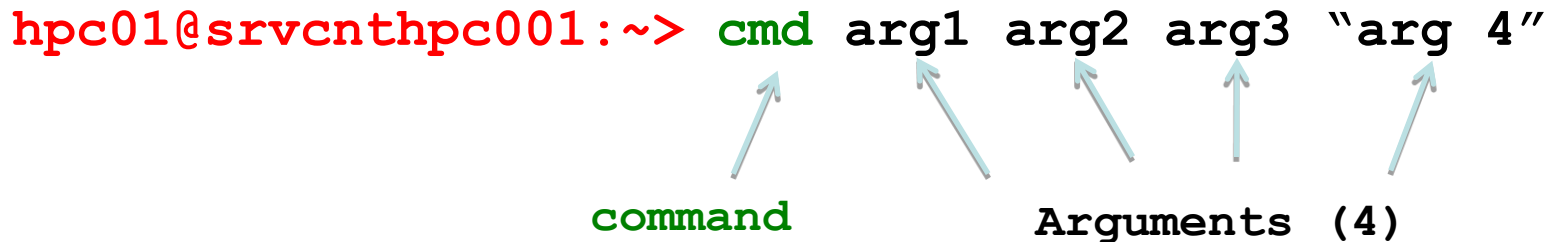
hpc01@srvcnthpc001:~> cmd arg1 arg2 arg3



command prompt command Arguments (3)

The diagram illustrates the components of the command line 'hpc01@srvcnthpc001:~> cmd arg1 arg2 arg3'. Arrows point from the labels below to the corresponding parts of the command: 'command prompt' points to the red text 'hpc01@srvcnthpc001:~>', 'command' points to the green text 'cmd', and 'Arguments (3)' points to the three arguments 'arg1', 'arg2', and 'arg3'.

hpc01@srvcnthpc001:~> cmd arg1 arg2 arg3 "arg 4"



command Arguments (4)

The diagram illustrates the components of the command line 'hpc01@srvcnthpc001:~> cmd arg1 arg2 arg3 "arg 4"'. Arrows point from the labels below to the corresponding parts of the command: 'command' points to the green text 'cmd', and 'Arguments (4)' points to the four arguments 'arg1', 'arg2', 'arg3', and '"arg 4"'.

Commands and Shells

- Commands allow users to interact with the operating system via the *shell*
- Two-way communication is possible between the *shell* and commands
 - e.g. the `ls` command will return a list of files in a directory

Exercise 1(a)

Running Commands

Command	Description
ls	<i>Shows a directory listing</i>
w	Shows who is logged on to the system and what they are doing
w hpc01	Shows login, idle time and what user hpc01 is doing
date	Prints the system time and date
uptime	Tells you how long the system has been running

Command Line Options (Flags)

- **Command Line Options / Flags** modify the operation of the command.
 - There are two forms of flags – **Short Form & Long Form**.
 - Flags are case sensitive!
- **Short form options** start with a single hyphen “-”
 - `rm -f <filename>` → force removal of file
- **Long form Options** start with a double hyphen “--”
 - `rm --force <filename>` → force removal of file

Both commands do the exact same thing!

More Commands & Flags

- `ls -l`
 - short form flag `"-l"`
 - Shows the long format listing for all files in the directory

Command Line Arguments

- **Command Line Parameters** are arguments sent to the program being called.
 - There are two forms of parameters – **Short Form & Long Form.**
 - Parameters are case sensitive!
- **Long form Options** start with a double hyphen "--"
`./configure --prefix=path1/dir_install` →

Default argument would be passed if nothing was specified
"/usr/local"

Exercise 1(b)

Flags and Parameters

Command	Description
<code>ls</code>	Shows a directory listing
<code>ls -l</code>	(long) Lists directory contents of current directory using long listing format
<code>ls -a</code> <code>ls --all</code>	(all) Lists directory contents of current directory and does not ignore entries starting with .
<code>ls -la</code> <code>ls -all -l</code>	(all+long) Lists directory contents of current directory using long listing format and does not ignore entries starting with .
<code>ls --format=horizontal</code>	Lists directory contents of current directory horizontally

The online manual

- **man** is a command that takes as its argument the name of another command.

NAME

find - search for files in a directory hierarchy

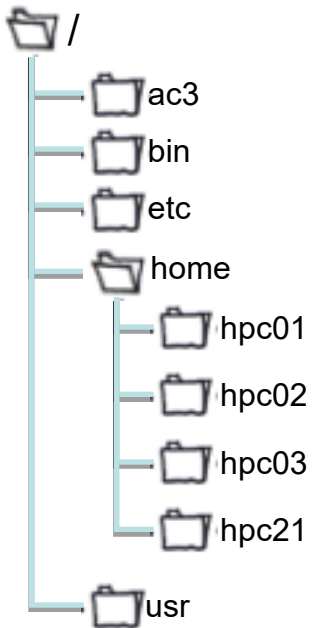
SYNOPSIS

find [-H] [-L] [-P] [path...] [expression]

- The [] indicate an optional argument (you don't type these)

Directories

- Called 'folders' under windows
 - Exactly the same concept in Linux
- You run into them a LOT more under Linux than you do under windows
- A filesystem directory is a container
 - It can contain files and other directories



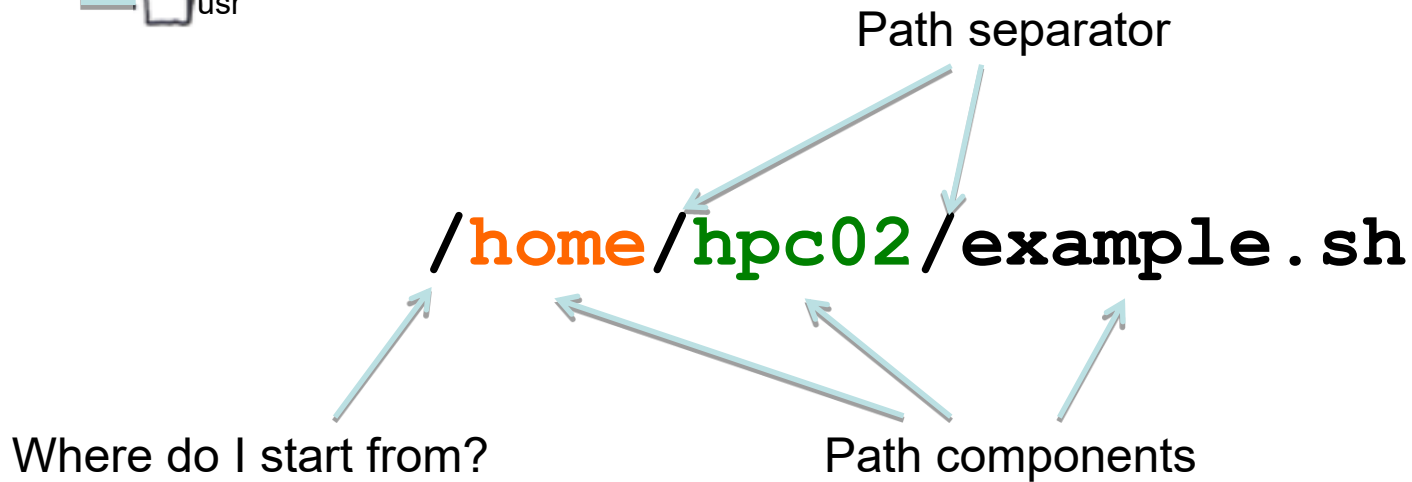
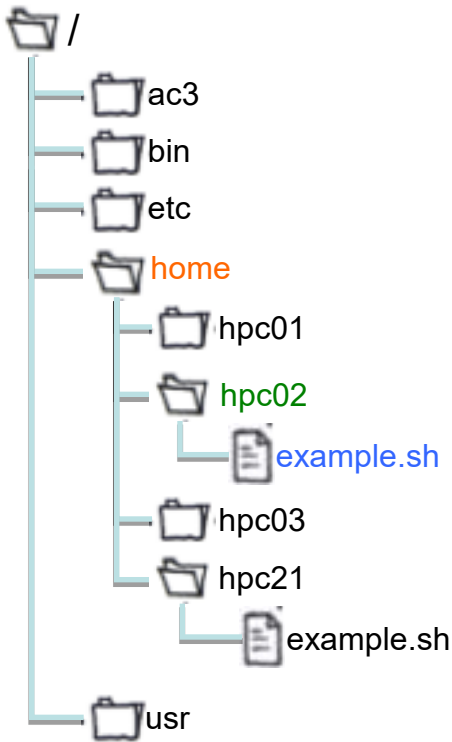
What are the names of the contents of the directory called **/home**?

/home/hpc01

/home/hpc02

/home/hpc03

/home/hpc21

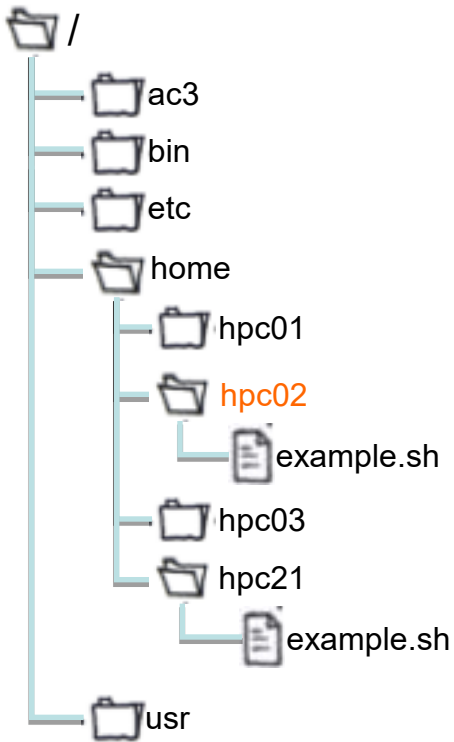


Rules So Far

- There is one root, called “/”
 - This is different from windows, where there is one root for each disk drive C:, D:, etc
- A path from the root designates exactly one file: such a path is called an “absolute path”

The Home Directory

- Each account (user) has a special home directory
 - That's where you 'get put' when you log in. (More on this later)
- BASH uses the "~" character to indicate "home directory"
- Two forms
 - ~ "my home directory"
 - ~hpc01 "hpc01's home directory"



Path separator

~hpc02/example.sh

Where do I start?

Path components

Rules So Far

- There is one root, called “/”
- A path starting with “/” means “from the root”
- A path starting with “~” means “from the home”

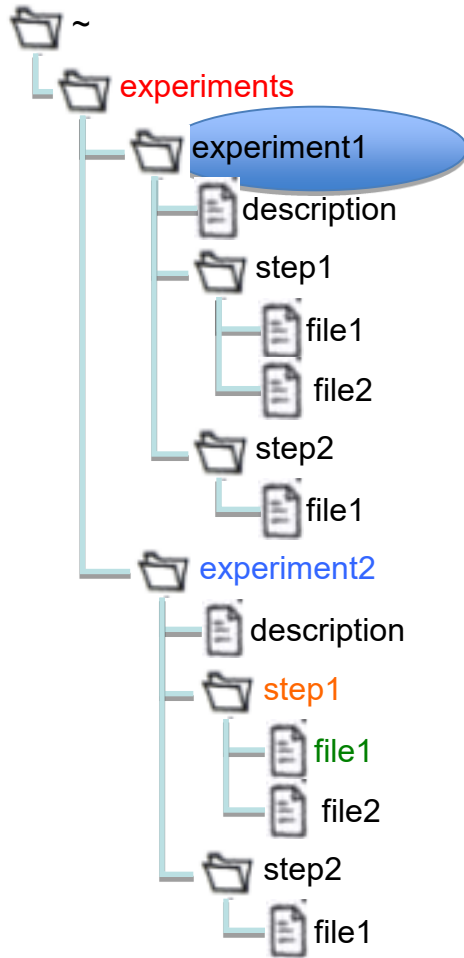
Exercise 2(a)

Finding your way around

Command	Description
cd	Change the working directory to your home directory
cd <i><path></i>	Change directory to <i><path></i>
ls	List the contents of the working directory
ls <i><path></i>	List the contents of <i><path></i>

Looking upwards

- UNIX uses “..” to denote the parent of a directory. You can use this when running commands, e.g.
 - `cd ..` → change up 1 directory
 - `cd ../hpc01` → change up 1 directory, then down 1 directory to hpc01
- UNIX understands “..” as a “normal directory” and it can appear in a path



Path separator

../experiment2/step1/file1

Where do I start?

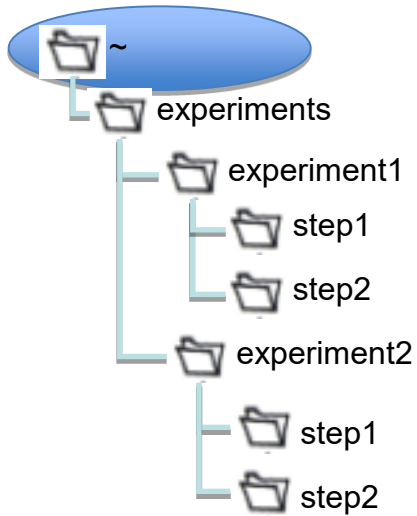
Path components

Making New Directories

- By default you can create only one directory at a time:

```
mkdir </path/new_directory>
```

```
mkdir </path/to/new/directory>
```



What happens when you execute the commands?

```
mkdir ~/experiments
```

```
cd ~/experiments
```

```
mkdir experiment1
```

```
mkdir experiment1/step1
```

```
mkdir experiment1/step2
```

```
mkdir experiment2/step1
```

- Did the above command work?

```
mkdir -p experiment2/step1
```

```
cd experiment2
```

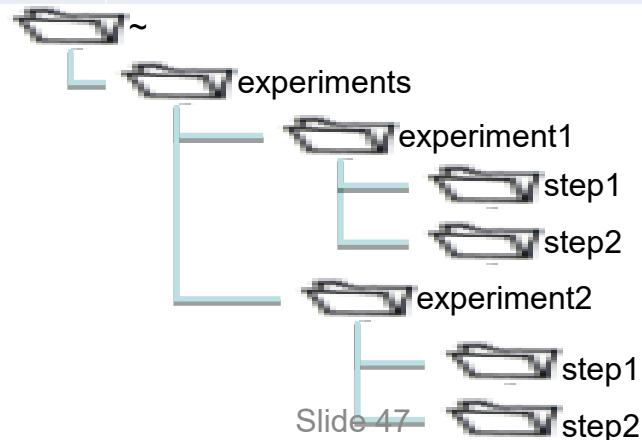
```
mkdir step2
```

```
tree ~/experiments
```

Exercise 2(c)

Making Directories

Command	Description
mkdir <i><path></i>	Make a directory at <i><path></i>
pwd	Print the name of the current working directory
cd <i><path></i>	Change directory to <i><path></i>
ls <i><path></i>	List the contents of <i><path></i>



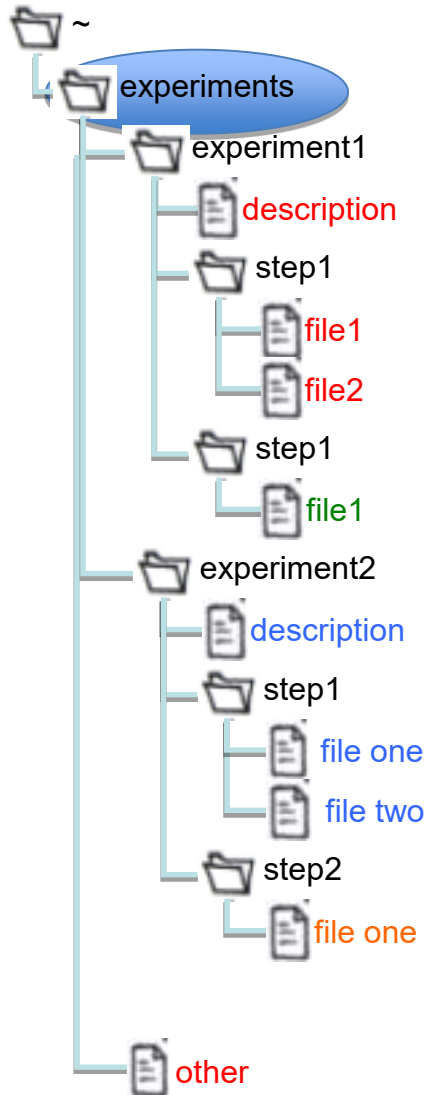
Creating/Moving/Copying Files Around

- Files live in exactly one location
- Files can be copied and moved between directories
- Files can be created by using the " touch " command

cp *<from path/file> <to path/file>*

mv *<from path/file> <to path/file>*

touch *<path/file>*



What happens when you execute the commands?

```
cd ~/experiments
touch experiment1/description
cp experiment1/description experiment2
cd experiment1

touch step1/file1 step1/file2
cp step1/file1 ../experiment2/step1
cp ../experiment2/description ../.

cd ../experiment2/step1
mv file1 ../step2

cd ~/experiments

tree experiment1 experiment2
```

Exercise 2(d)

Moving and Copying Files

Command	Description
cp <i><from></i> <i><to></i>	Copy a file from <i><from></i> to <i><to></i>
mv <i><from></i> <i><to></i>	Copy a file from <i><from></i> to <i><to></i> then remove <i><from></i>
man cp man mv	Print the online manual entry for <i><command name></i> . What happens when you provide more than two arguments to cp and mv ?

Editing files

- Invoke with:

```
nano <file_name.txt>
```

- Use the arrow keys to navigate your document
- Update the text by typing, backspace, etc.
- Use **Control**+**O** to save the file (^O).
- Use **Control**+**X** to quit (^X).

Exercise 4(a)

Editing a file in-situ

Command	Description
nano <i><file></i>	Will open file <i><file></i> in a text file editor
CTRL+O	Using CTRL+O within the nano editor will cause any changes made to the file while editing to be saved to the file
CTRL+X	Using CTRL+X within the nano editor will cause the editor to close. If you have not already saved your changes you will be asked if you wish to save those changes by answering Y or N

Break : 30 min

Module 3 : Transferring files

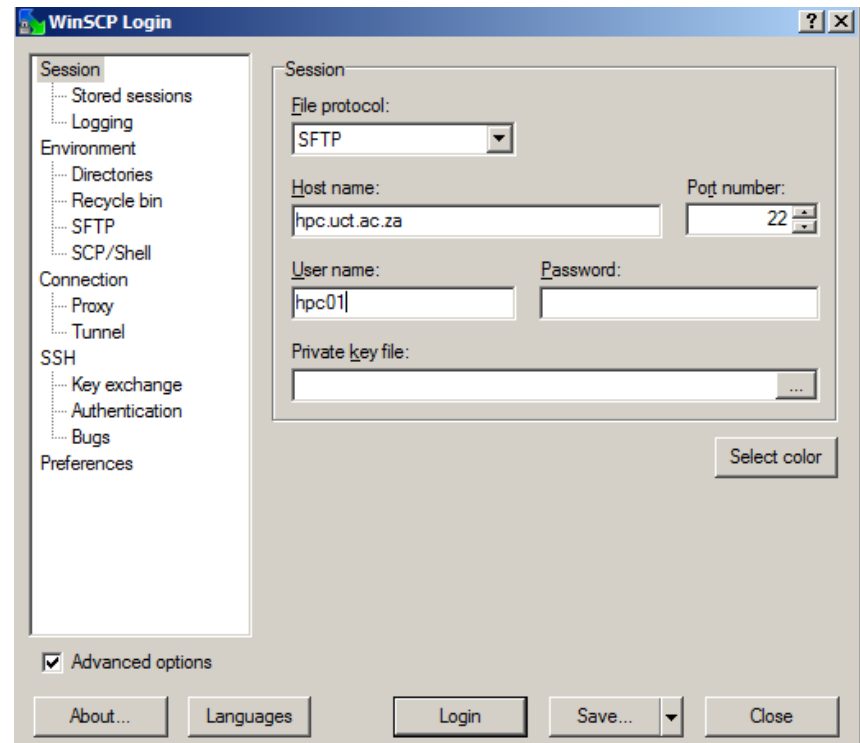
20 minutes

scp

- Secure Copy (**scp**) is another way to transfer files to and from the HPC

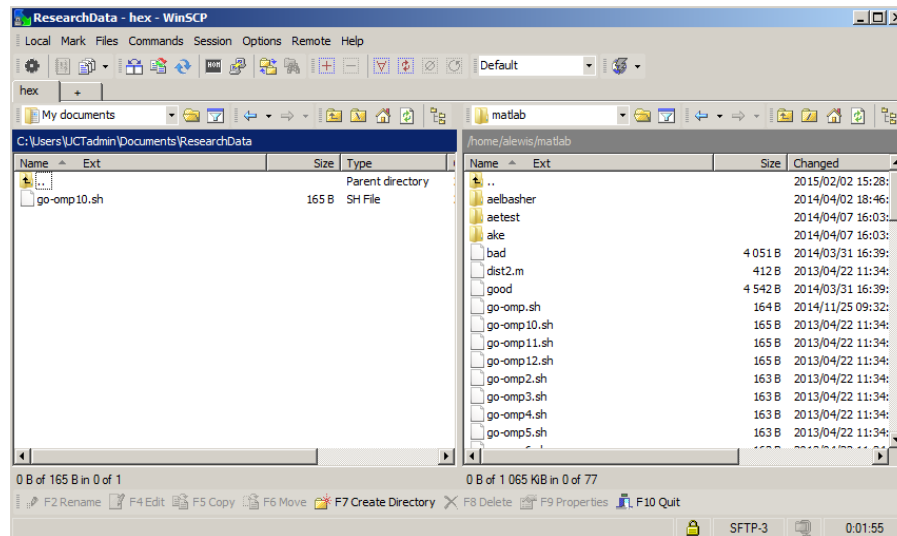
WinSCP

- WinSCP is a graphical Windows based tool that can move data between your desktop and a linux server.
- Free, download from:
 - <http://winscp.net/eng/download.php>
- Login much like putty



WinSCP

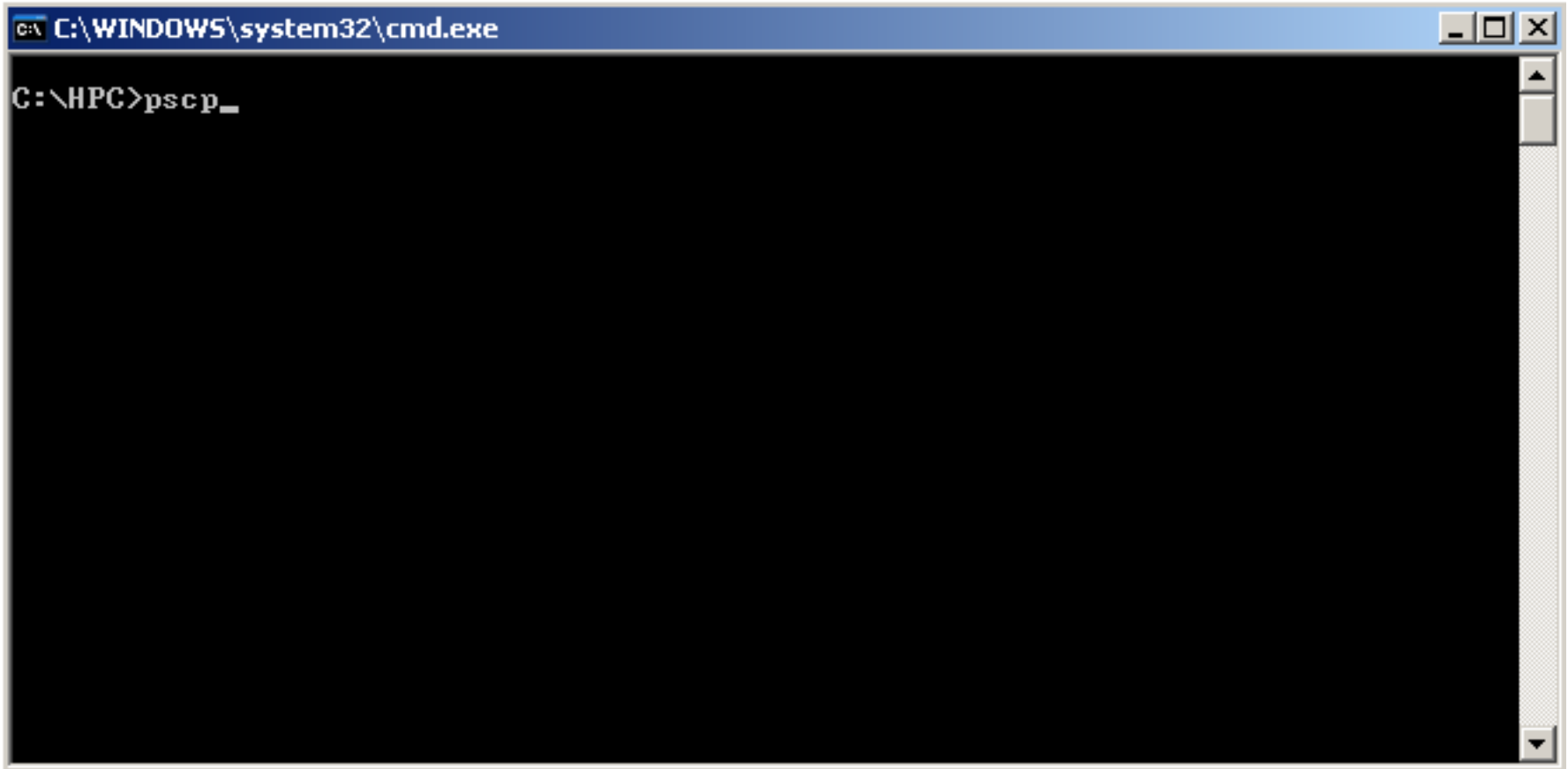
- On the left is my Windows PC, on the right is the HPC cluster.
- I can drag files between the panes.



PSCP – An SCP Client

- Putty comes with an SCP client `pscp.exe` – Putty Secure Copy.
- We'll be using PSCP in the exercises.
- To use it we need to open a **Windows Command Prompt**.
- An easy way to do this is to select *Run...* from the Start menu and type `cmd`. Then click *OK*.

Windows command prompt



PSCP Syntax

- Transfer file from local machine to the HPC machine:

```
pscp <file_name.ext> <user_name>@hpc.uct.ac.za:<dest_dir>
```



The local file
you want to
copy



Your training account user
name



Where you
want the file to
go on the HPC
machine

Exercise 2(a)

Transferring files from your local machine to the training HPC machine using PSCP

Command	Description
echo %PATH%	DOS command to show the value for the PATH variable
set PATH=<path>	DOS command to set the value for the PATH variable equal to <path>
cd <directory>	DOS command to change the directory to <directory>

PSCP Syntax

- Transfer file from the HPC machine to your local machine:

```
pscp <user_name>@hpc.uct.ac.za:<path/to/file_name.txt> .
```



Your training
account
user name



The path on the
HPC machine to
the file you want to
copy



Where to put
the file locally.
In this case "."
for the current
working
directory

Module 4: Submitting jobs

40 minutes

Add a job to the queue

- The scheduler is called SLURM:
 - Simple Linux Utility for Resource Management
- To add a job to the SLURM queue, we write a **job script**. A job script is simply a shell script (text file).
- The `#` symbol signifies a comment for the Shell
- However it has some special comments that pass info to SLURM.

Add a job to the queue

- **#SBATCH** is a keyword for SLURM and specifies that this line is for the scheduler. The linux shell will ignore it.
- When we want to queue the job, we pass its filename as a parameter to **sbatch**, e.g.
 - **sbatch <job-script>**
- The batch queuing system will return a number that uniquely identifies the job.

```
alewis@srvcnthpc001 ~/test$ sbatch test.sh
Submitted batch job 235294
alewis@srvcnthpc001 ~/test$
```

A sample SBATCH job script

```
#!/bin/bash
#SBATCH --account icts
#SBATCH --partition=ada
#SBATCH --nodes=1 --ntasks=1
#SBATCH --time=10:10:00
# By default you start in ~
cd myfolder
pwd
date
hostname
sleep 10
```

Rules!!!

No space before #SBATCH

No space between # and SBATCH

A sample SBATCH job script

Output...

```
#!/bin/bash
#SBATCH --account icts
#SBATCH --partition=ada
#SBATCH --nodes=1 --ntasks=1
#SBATCH --time=10:10:00
# By default you start in ~
cd myfolder
pwd
date
hostname
sleep 10
```

```
cat slurm-235294.out
/home/alewis/myfolder
Tue Aug  6 10:37:52 SAST 2019
srvcnthpc113
```

You've got mail!

```
# Set email address
```

```
#SBATCH --mail-user=USERSEMAIL
```

```
# Send an email when jobs
```

```
# begins, gets fails or ends
```

```
#SBATCH --mail-type=BEGIN,END,FAIL
```


Example SBATCH job script

```
#!/bin/sh
# This example submission script contains several important directives, please examine it thoroughly

# The line below indicates which accounting group to log your job against
#SBATCH --account=$USERSACCOUNT

# The line below selects the group of nodes you require
#SBATCH --partition=$USERSPARTITION

# The line below means you need 1 worker node and a total of 2 cores
#SBATCH --nodes=1 --ntasks=2

# The line below indicates the wall time your job will need, 10 hours for example. NB, mandatory directive!
#SBATCH --time=10:00:00

# A sensible name for your job, try to keep it short
#SBATCH --job-name="MyJob"

# Modify the lines below for email alerts. Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL
#SBATCH --mail-user=$USERSEMAIL
#SBATCH --mail-type=BEGIN,END,FAIL

# The cluster is configured primarily for OpenMPI and PMI. Use srun to launch parallel jobs if your code is
# parallel aware. To protect the cluster from code that uses shared memory and grabs all available cores the
# cluster has the following environment variable set by default: OMP_NUM_THREADS=1
# If you feel compelled to use OMP then uncomment the following line:
# export OMP_NUM_THREADS=$SLURM_NTASKS

# NB, for more information read https://computing.llnl.gov/linux/slurm/sbatch.html

# Use module to gain easy access to software, typing module avail lists all packages.
# Example:
# module load python/anaconda-python-3.7

# Your science stuff goes here...
```

Exercise 2

Create a script and submit a very simple job

Command	Description
#SBATCH	#SBATCH is a keyword for SLURM and specifies that this line is for the scheduler. The Shell will ignore it
#	Signifies a comment for the Shell, e.g. # Next line will create a job
sbatch	Submit a job to SLURM
squeue	List jobs in the queue
cat <i><filename></i>	Print a file to the terminal (catenate)
less <i><filename></i>	Like cat , but less at a time
nano <i><filename></i>	Will open file <i><filename></i> in a text file editor

Thank You