Basic course:
Introduction to Linux for HPC

- 1) Overview of HPC: (30 min)
- 2) Linux (60 min)

- break (30 min)

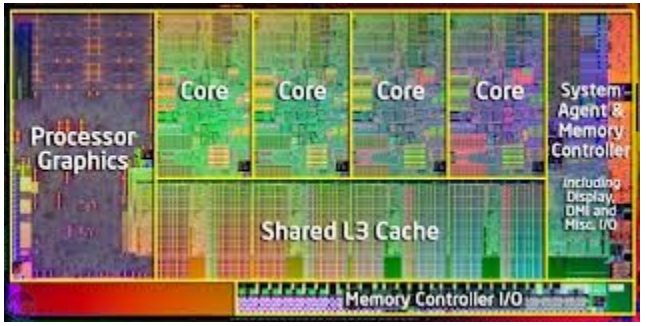
- 3) Transferring files (20 min)
- 4) Submitting jobs (40 min)

Module 1: What is HPC?

30 minutes

What is HPC?

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



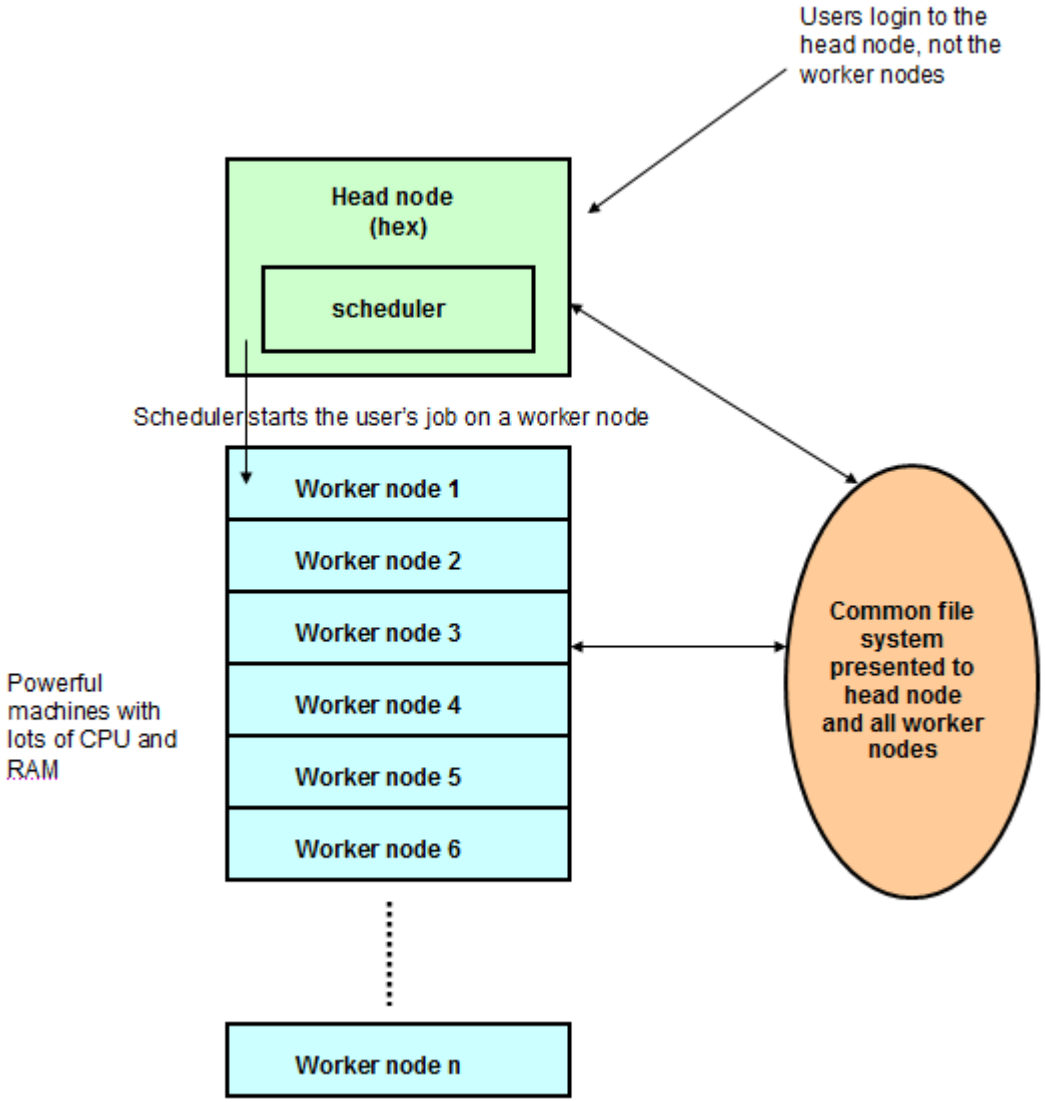
What HPC isn’t

- Linux, not Windows.
- No GUI, point and click.
- Windows software not always available.

Architecture

- Operating system: Suse Linux
- X86_64
- HPC server: Torque PBS
- Scheduler: Maui
- Worker nodes:
  - Dell C6145
  - Supermicro GPU servers
  - Virtual machines

Architecture

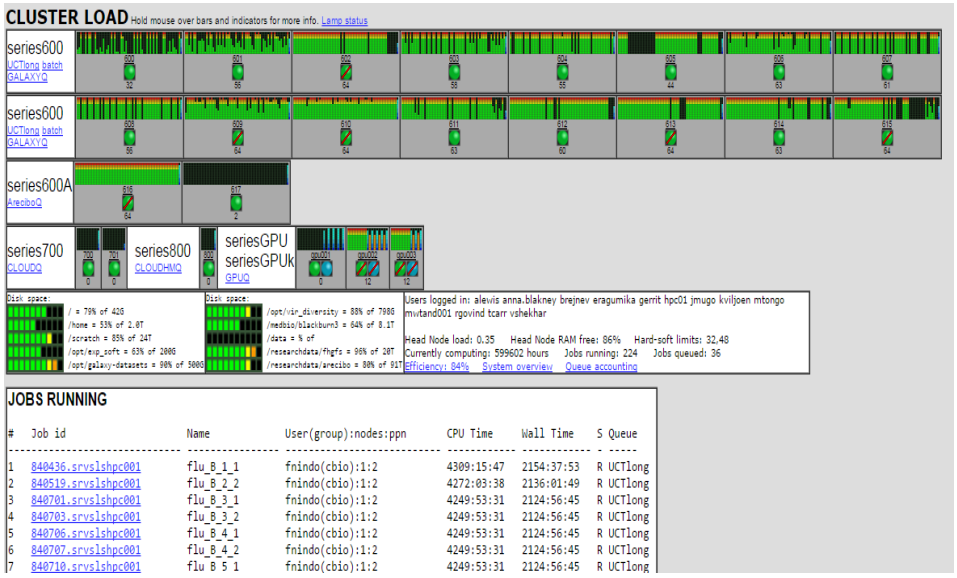


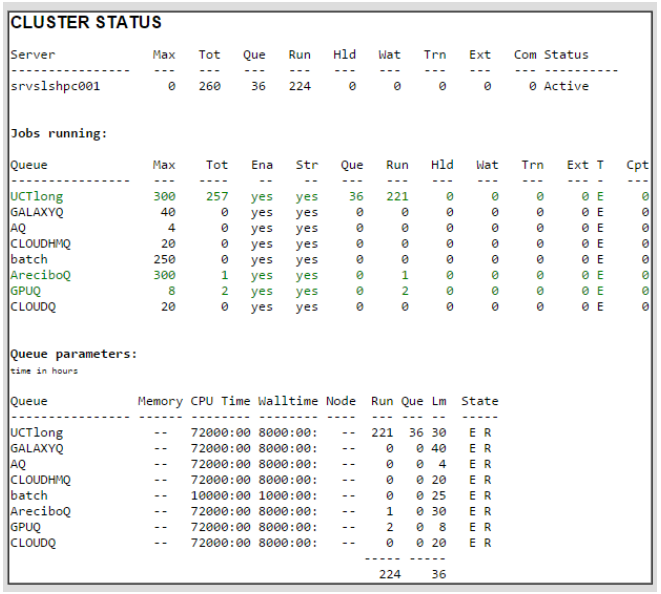
Shared file system

- 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/mydata on a worker node it will be deleted on the head node!

The dashboard

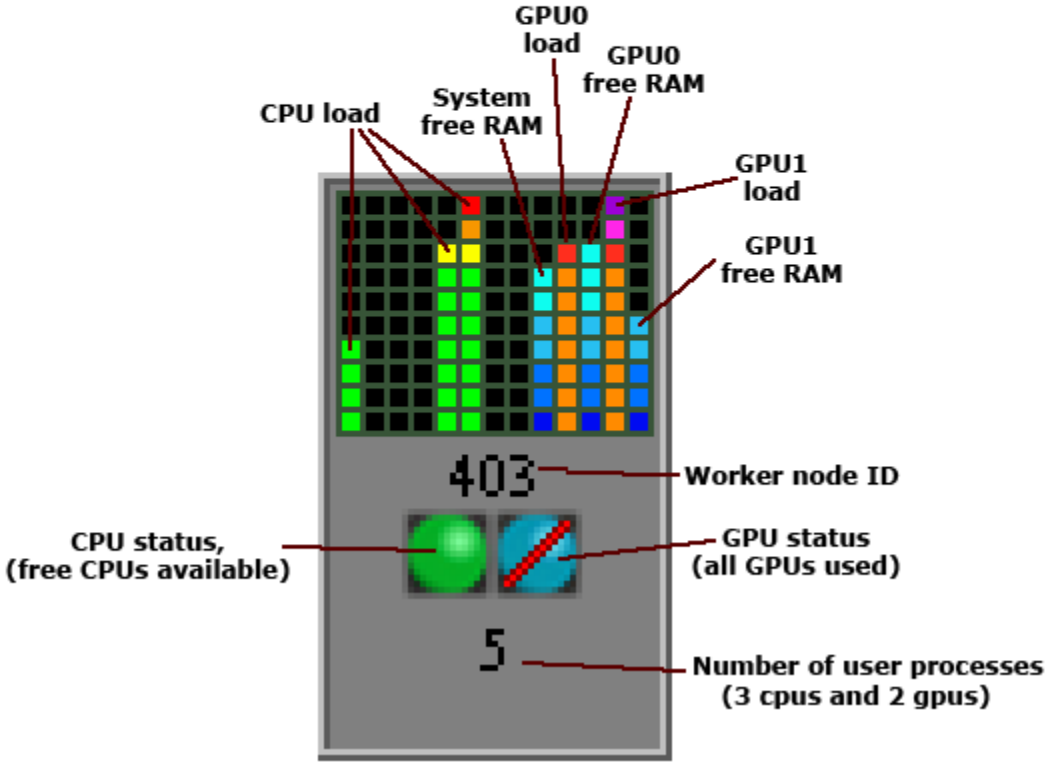
- To keep track of the cluster's status, workload and the jobs that are running go to: http://hex.uct.ac.za





The dashboard

Icon	Value	Description
	Free CPUs	There are free CPUs, jobs may be submitted to this node.
	Job-exclusive	All CPUs are busy, the node is running but no further jobs may be submitted.
	Busy	Torque mom daemon or CPUs too busy to respond to further requests. Jobs are running but may be degraded.
	Down	Node down or PBS mom daemon offline or not responding, no jobs may be submitted.
	Free GPUs	There are free GPUs, jobs may be submitted.
	Busy	All GPUs are busy, the node is running but no further jobs may be submitted.



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 to for your preference, examples “ csh, ksh, bash, zsh, pdsh, tcsh …. the list goes on “

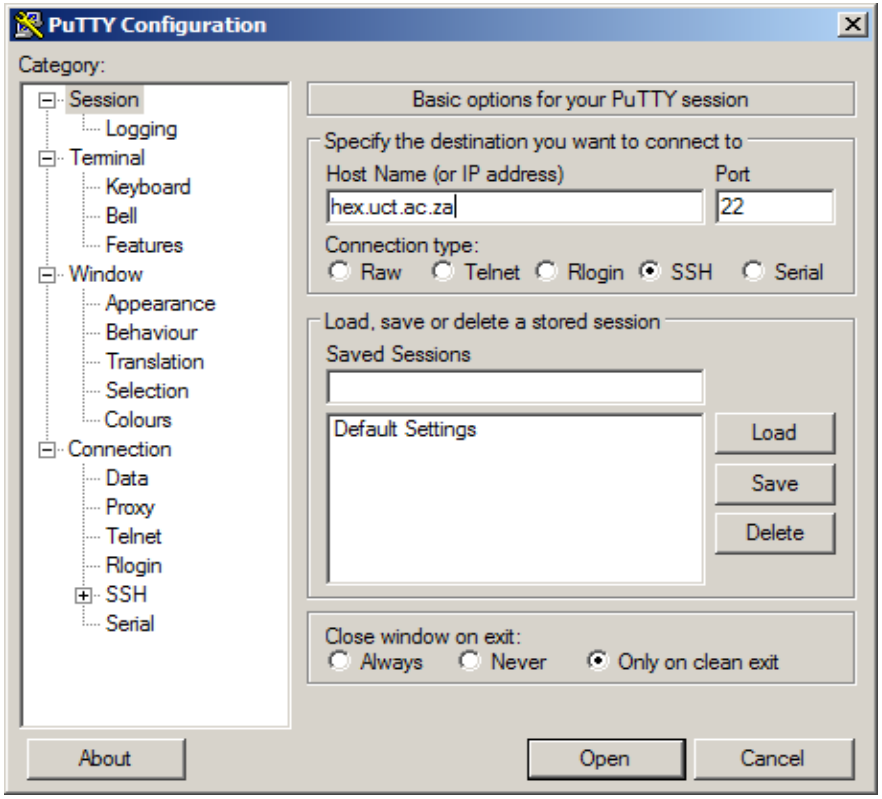
Install software

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.

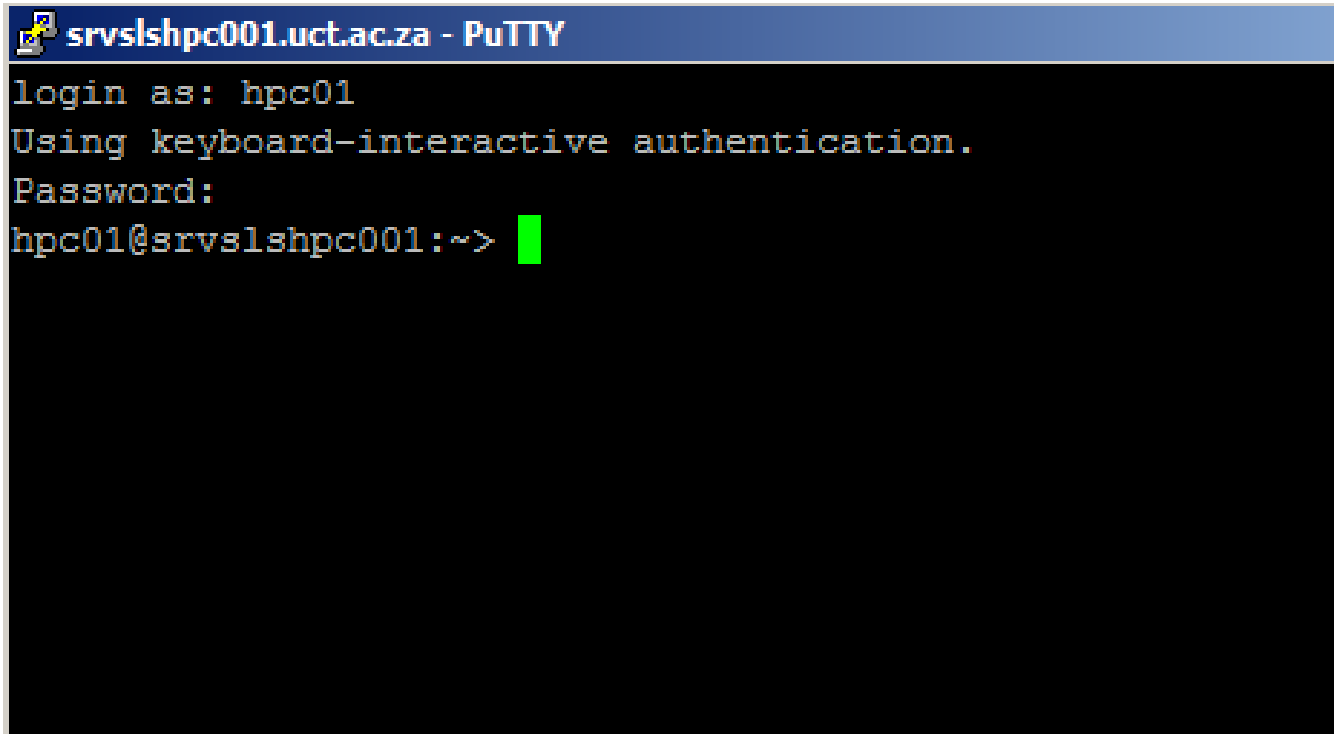
Logging On

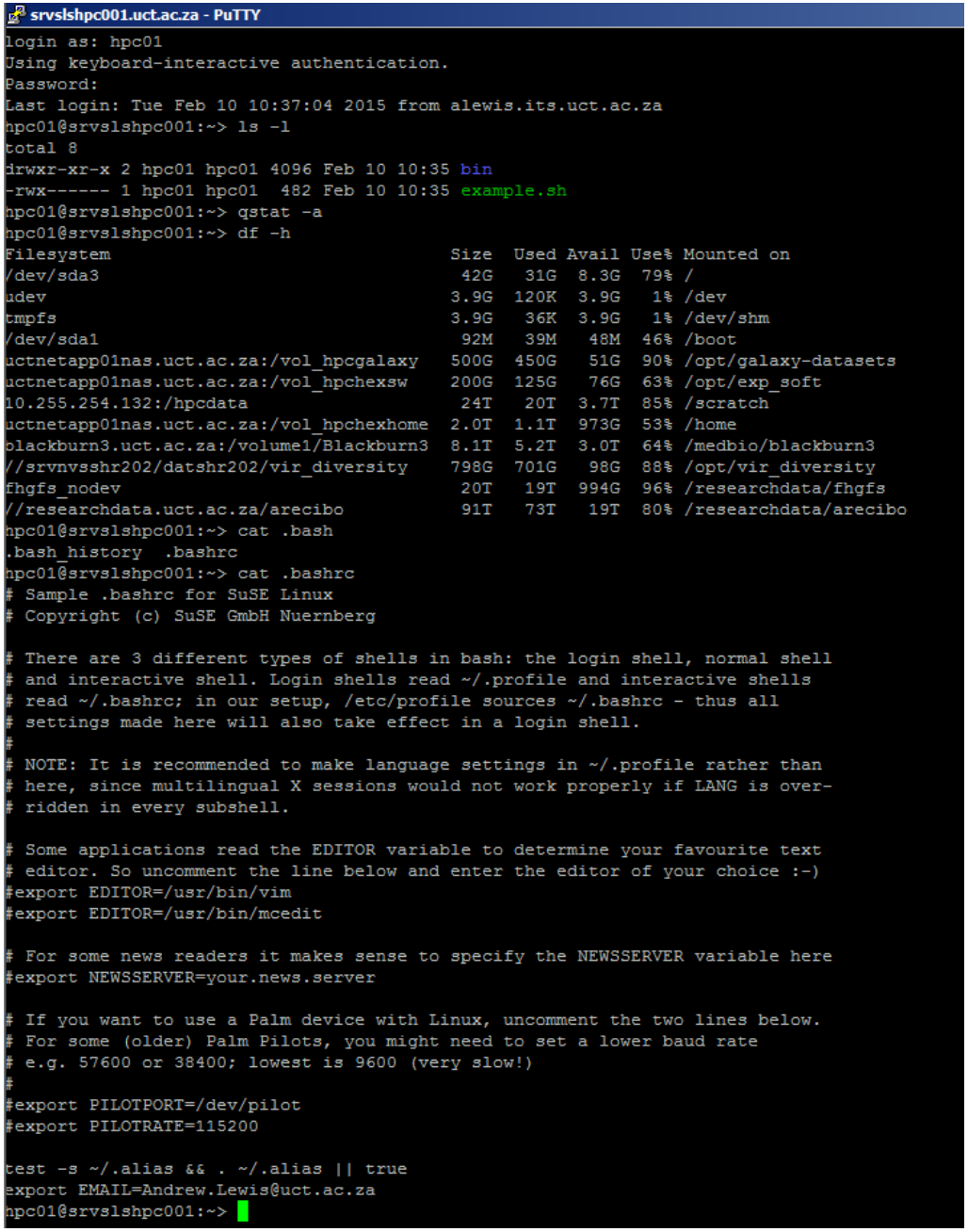
- Start the putty telnet/ssh client by double clicking on putty.exe and connect to the HPC Machine
  - Host: **`hex.uct.ac.za`**
  - Connection Type: **`ssh`**
  - Port: **22**



Log On to the HPC Machine

- Log into the training HPC system using the Test Account allocated to you, e.g.
  - **Account Name:** Test Acc
  - **Password:**




srvslshpc001.uct.ac.za - PuTTY
login as: hpc01
Using keyboard-interactive authentication.
Password:
Last login: Tue Feb 10 10:37:04 2015 from alewis.its.uct.ac.za
hpc01@srvslshpc001:~> ls -l
total 8
drwxr-xr-x 2 hpc01 hpc01 4096 Feb 10 10:35 bin
-rwx------ 1 hpc01 hpc01 482 Feb 10 10:35 example.sh
hpc01@srvslshpc001:~> qstat -a
hpc01@srvslshpc001:~> df -h
Filesystem Size Used Avail Use% Mounted on
/dev/sda3 42G 31G 8.3G 79% /
udev 3.9G 120K 3.9G 1% /dev
tmpfs 3.9G 36K 3.9G 1% /dev/shm
/dev/sda1 92M 39M 48M 46% /boot
uctnetapp01nas.uct.ac.za:/vol_hpcgalaxy 500G 450G 51G 90% /opt/galaxy-datasets
uctnetapp01nas.uct.ac.za:/vol_hpchexsw 200G 125G 76G 63% /opt/exp_soft
10.255.254.132:/hpcdata 24T 20T 3.7T 85% /scratch
uctnetapp01nas.uct.ac.za:/vol_hpchexhome 2.0T 1.1T 973G 53% /home
blackburn3.uct.ac.za:/volume1/Blackburn3 8.1T 5.2T 3.0T 64% /medbio/blackburn3
//srvnvsshr202/datshr202/vir_diversity 798G 701G 98G 88% /opt/vir_diversity
fhgfs_nodev 20T 19T 994G 96% /researchdata/fhgfs
//researchdata.uct.ac.za/arecibo 91T 73T 19T 80% /researchdata/arecibo
hpc01@srvslshpc001:~> cat .bash
.bash_history .bashrc
hpc01@srvslshpc001:~> cat .bashrc
Sample .bashrc for SuSE Linux
Copyright (c) SuSE GmbH Nuernberg
There are 3 different types of shells in bash: the login shell, normal shell
and interactive shell. Login shells read ~/.profile and interactive shells
read ~/.bashrc; in our setup, /etc/profile sources ~/.bashrc - thus all
settings made here will also take effect in a login shell.
#
NOTE: It is recommended to make language settings in ~/.profile rather than
here, since multilingual X sessions would not work properly if LANG is over-
ridden in every subshell.
Some applications read the EDITOR variable to determine your favourite text
editor. So uncomment the line below and enter the editor of your choice :-)
#export EDITOR=/usr/bin/vim
#export EDITOR=/usr/bin/mcedit
For some news readers it makes sense to specify the NEWSSERVER variable here
#export NEWSSERVER=your.news.server
If you want to use a Palm device with Linux, uncomment the two lines below.
For some (older) Palm Pilots, you might need to set a lower baud rate
e.g. 57600 or 38400; lowest is 9600 (very slow!)
#
#export PILOTPORT=/dev/pilot
#export PILOTRATE=115200
test -s ~/.alias && . ~/.alias || true
export EMAIL=Andrew.Lewis@uct.ac.za
hpc01@srvslshpc001:~>

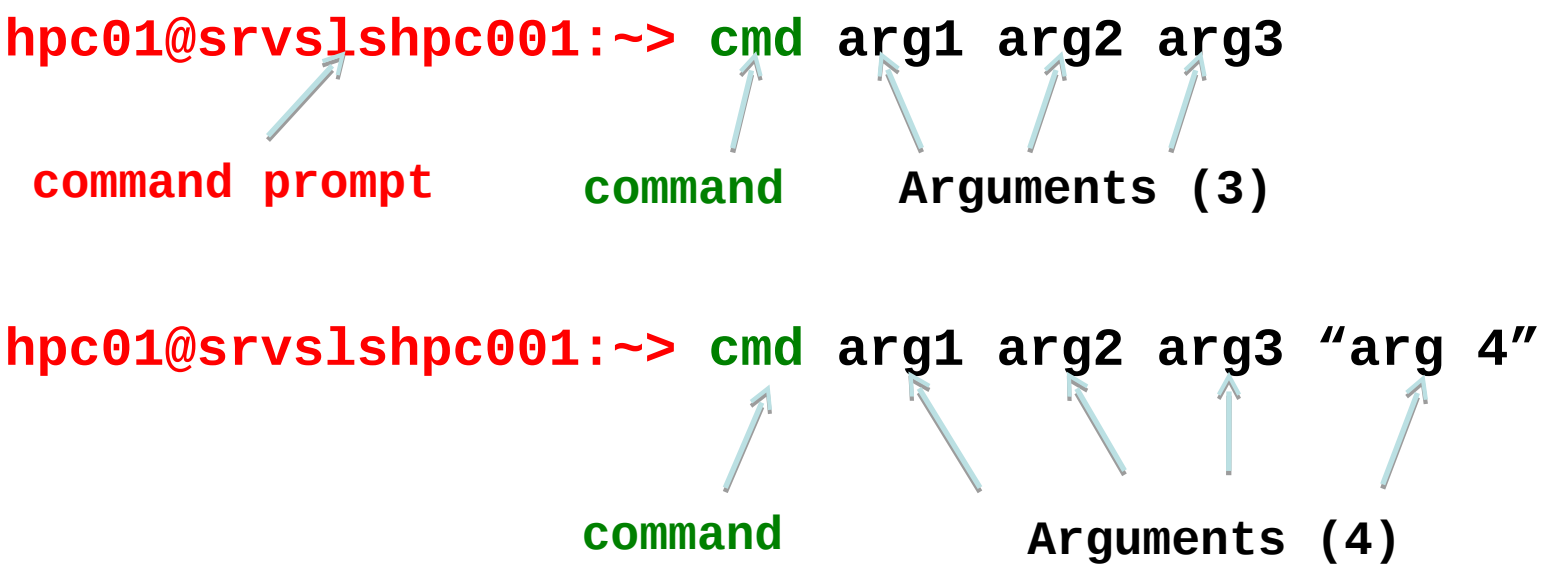
What does BASH do?

- The shell (command line interpreter) *interprets* the commands entered by the user and passes those to the Linux operating system.

- When 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



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`	*Shows a directory listing*
`w`	Shows who is logged on to the system and what they are doing
`w hpc01`	Shows login, idle time, and what user hpc22 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!

- **Short form options** start with a single hyphen "-"
  - `tail -n 20 <filename>` → tail from the last 20 lines of the file

- **Long form Options** start with a double hyphen "--"
  - `tail --lines=20 <filename>` → tail from the last 20 lines of the file

Exercise 1(b)

Flags and Parameters

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

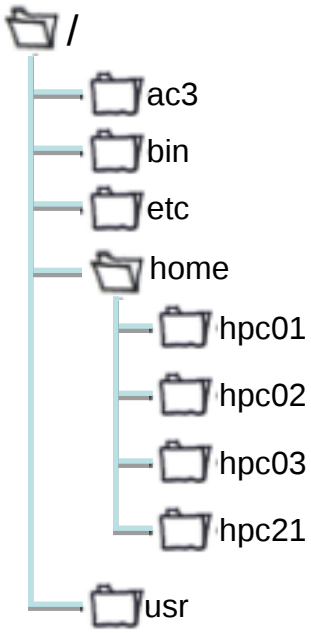
SYNOPSIS

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

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

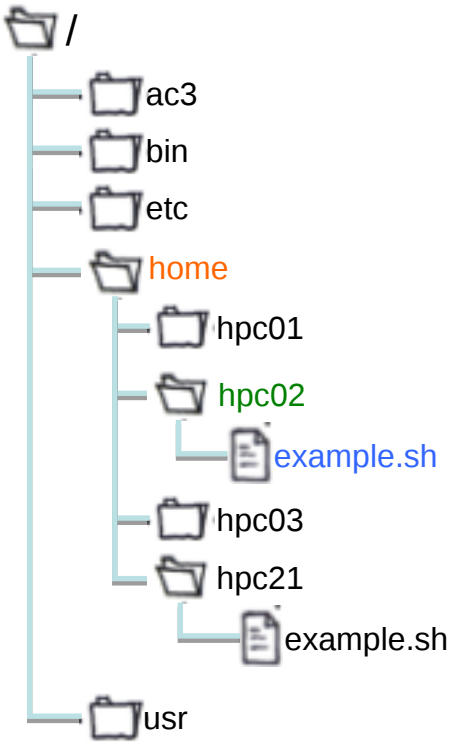
Directories

- Called ‘folders’ under windows
  - Exactly the same concept
- You run into them a LOT more under Linux than you do under windows
- A 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`



Path separator

/home/hpc02/example.sh

Where do I start from?

Path components

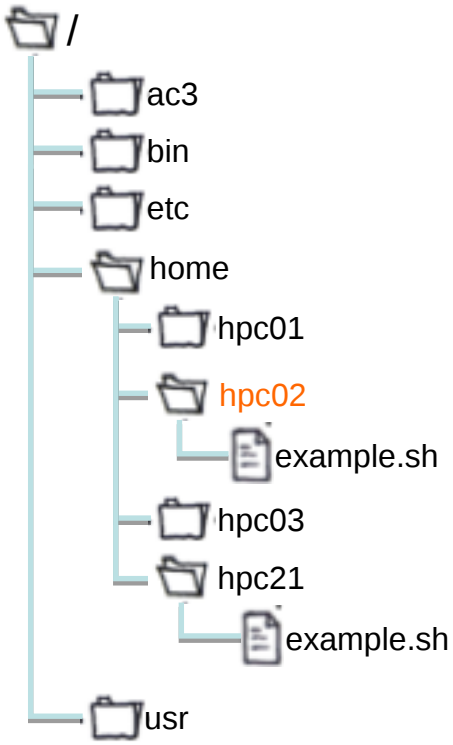
Rules So Far

- 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 directory called his/her 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”

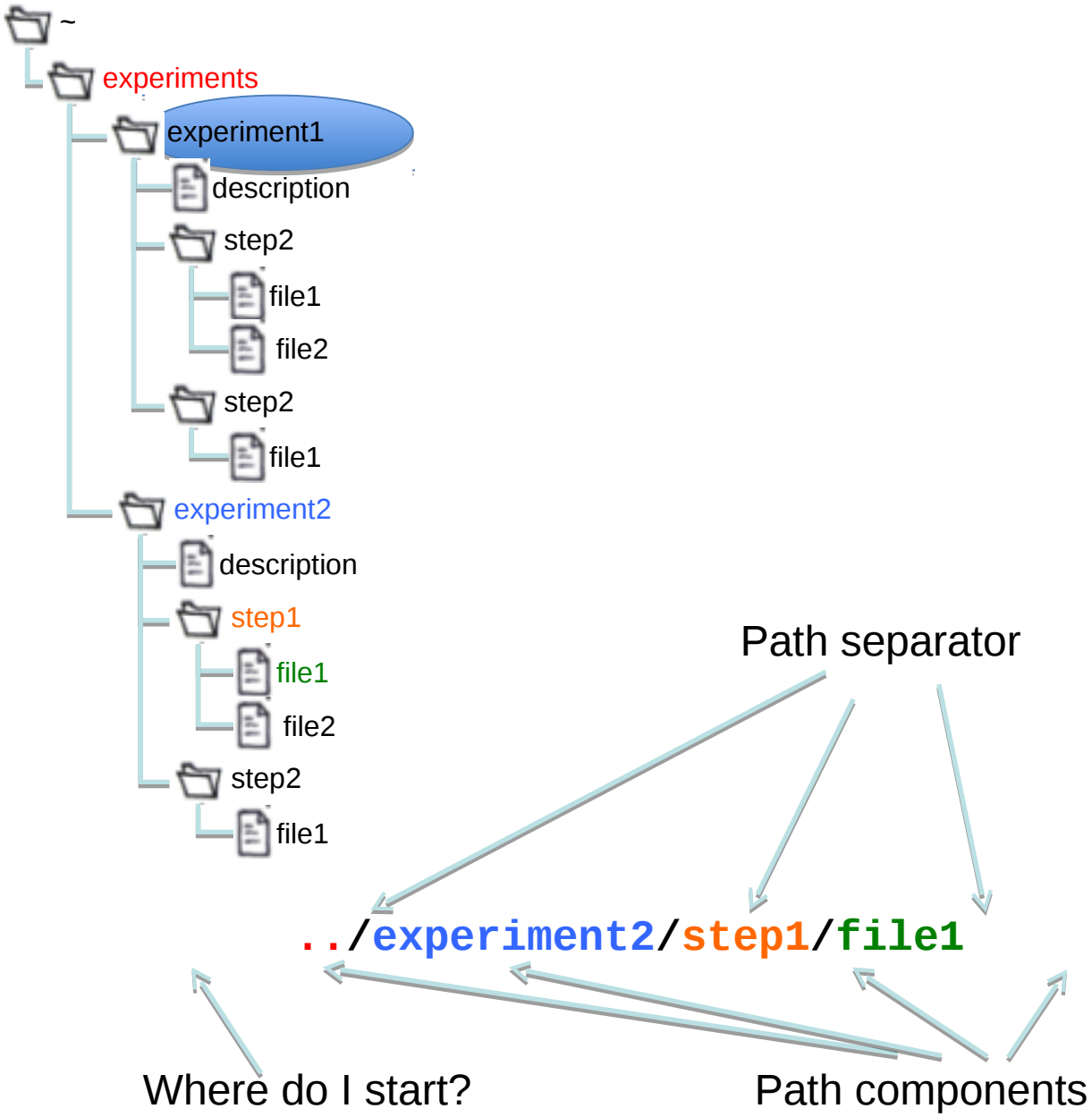
Exercise 2(a)

Finding your way around

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

Looking upwards

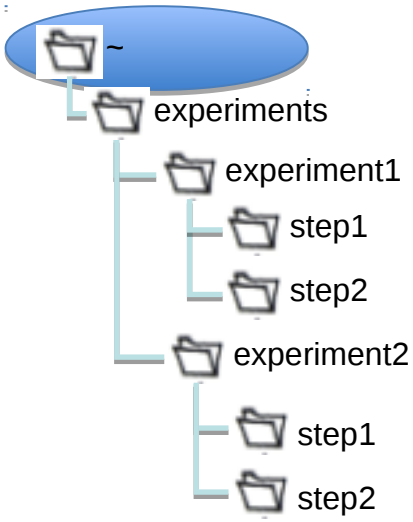
- UNIX uses “..” to denote the parent of a directory. You can use this when running commands, e.g.
  - `cd ..` → change up 1 directory
  - `cd ../hpc01` → change up 1 directory, then down 1 directory to hpc01
- UNIX understands “..” as a “normal directory” and it can appear in a path


~
experiments
experiment1
description
step2
file1
file2
step2
file1
experiment2
description
step1
file1
file2
step2
file1
Path separator
../experiment2/step1/file1
Where do I start?
Path components

Making New Directories

- You can create a directory, but only one at a time (by default)

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



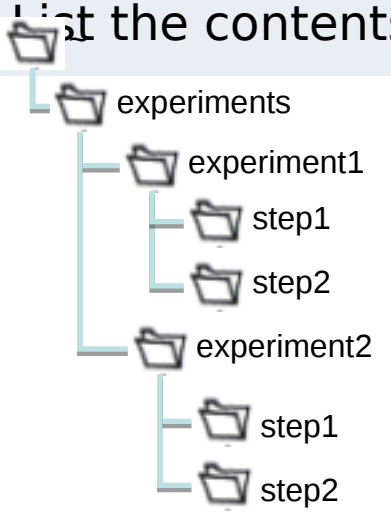
What happens when you execute the commands?

```
mkdir ~/experiments
cd ~/experiments
mkdir “experiment1”
mkdir “experiment1/step1”
mkdir “experiment1/step2”
mkdir “experiment2”
cd “experiment2”
mkdir “step1”
mkdir “step2”
```

Exercise 2(c)

Making Directories

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

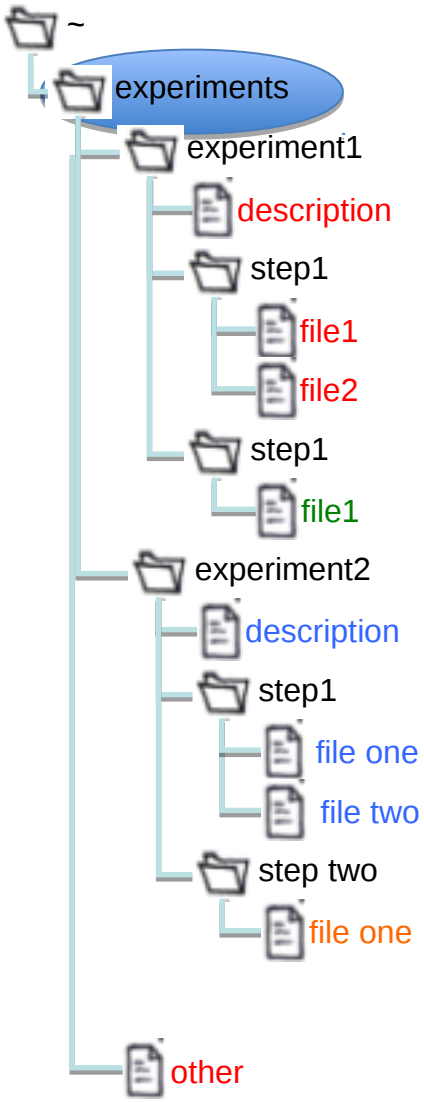


Moving/Copying Files Around

- Files live in exactly one location

- Files can be copied and moved between directories

```
cp <from path/file> <to path/file>
mv <from path/file> <to path/file>
```



What happens when you execute the commands?

```
cp “experiment1/description” “experiment2”
cd “experiment1”
cp “step1/file1” “../experiment2/step2”
cp “../experiment2/description” ../other
cd “../experiment2/step1”
mv “file1” “../step2”
```

Exercise 2(d)

Moving and Copying Files

Command	Description
cp *<from> <to>*	Copy a file from *<from>* to *<to>*
mv *<from> <to>*	Copy a file from *<from>* to *<to>* then remove *<from>*
man cp **man mv**	Print the online manual entry for *<command name>*. 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 *<file>*	Will open file *<file>* 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

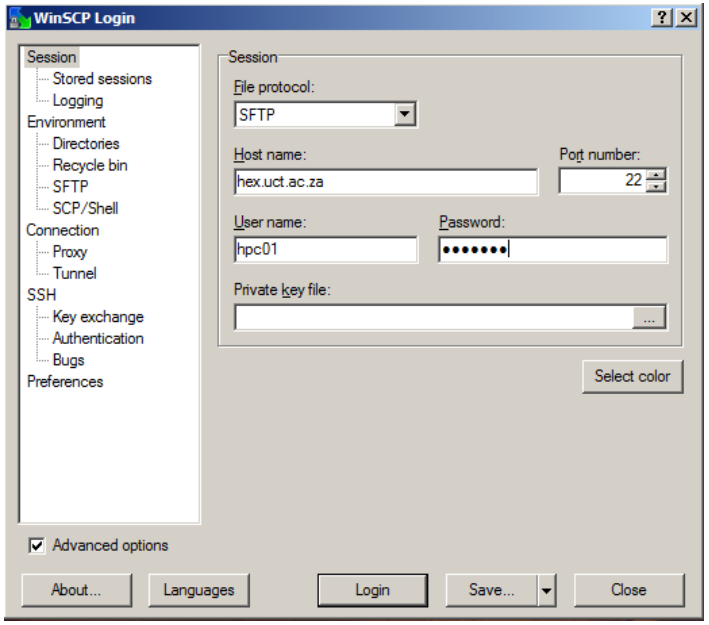
20 minutes

scp

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

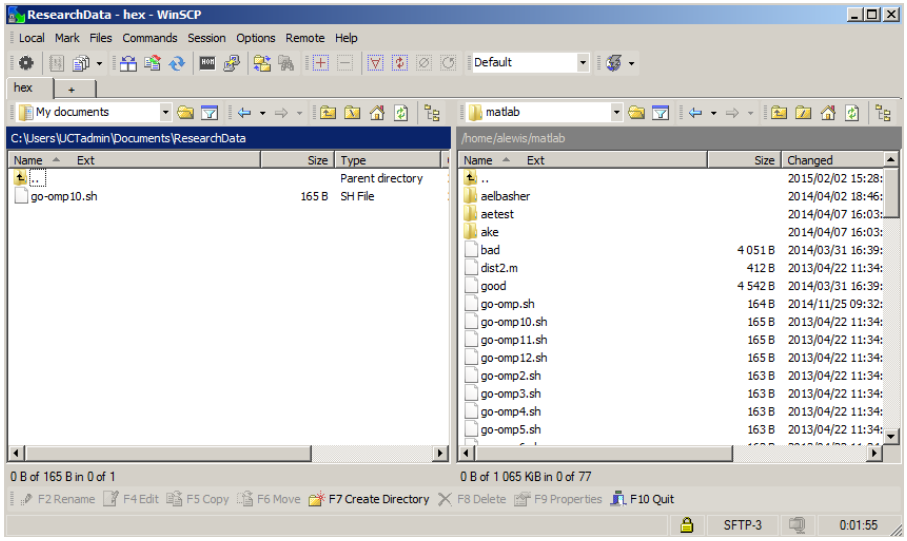
WinSCP

- 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

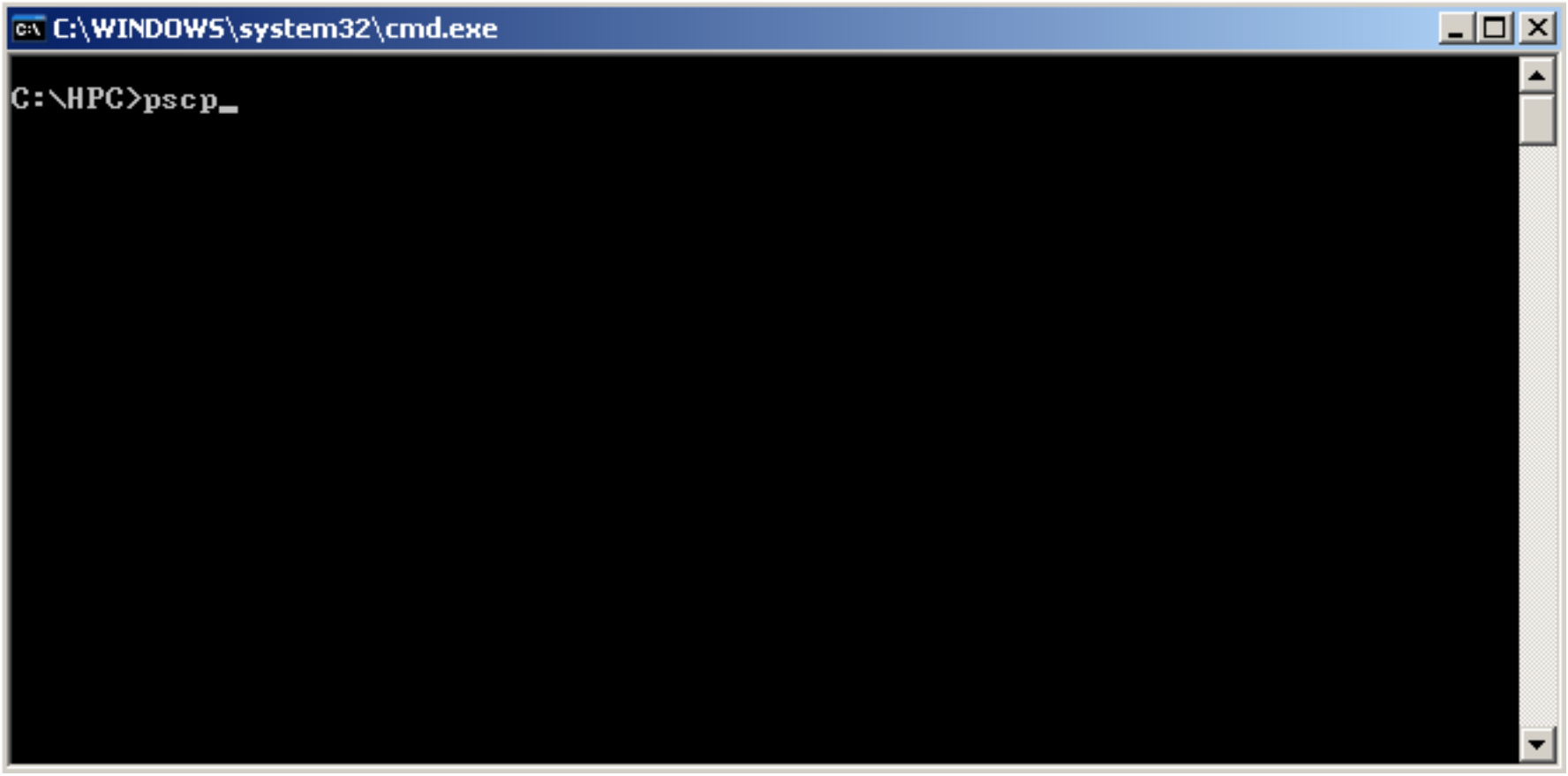
- On the left is my Windows PC, on the right is the HPC cluster.
- I can drag files between the panes.



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>@hex.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

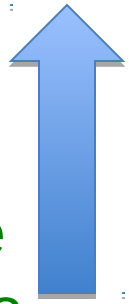
Command	Description
`echo %PATH%`	DOS command to show the value for the PATH variable
`set PATH=`*`<path>`*	DOS command to set the value for the PATH variable equal to *`<path>`*
`cd` *`<directory>`*	DOS command to change the directory to *`<directory>`*

PSCP Syntax

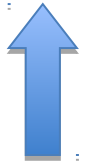
- Transfer file from the HPC machine to your local machine:

```
pscp <user_name>@hex.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

- To add a job to the queue, we write a **job script**. A job script is simply a script.
- The # symbol signifies a comment for the Shell
- It has some special comments that pass info to PBS.
- **#PBS** is a keyword for PBS and specifies that this line is for PBS. The Shell will ignore it
- When we want to queue the job, we pass its filename as a parameter to **qsub**, e.g.
  - **qsub <job-script>**
- The batch queuing system will return a number that uniquely identifies the job.

A sample PBS job script

NOTE: This script will not run on Hex as resources are assigned via groups on Hex, not via projects

```
#!/bin/bash
# Request resources
# * 10 minutes wall time to run
#PBS -l walltime=00:10:00
# * 1 node, 1 processor
#PBS -l nodes=1:ppn=1
# * 100 megabytes physical memory allocated to job
#PBS -l mem=100mb

cd myfolder
date
sleep 10
date # the job itself
 # By default you start in ~
```

You’ve got mail!

```
# Set email address
#PBS -M <name>@uct.ac.za
# Send an email when jobs
# begins (b), gets aborted (a)
# and ends (e)
#PBS -m abe
```

Exercise 2

Create a script and submit a very simple job

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

Exercise 3

Create another script and submit a more realistic sample job

Command	Description
#PBS	**#PBS** is a keyword for PBS and specifies that this line is for PBS. The Shell will ignore it
#	Signifies a comment for the Shell, e.g. **# Next line will create a job**
qsub	Submit a job to PBS
qstat	List jobs in the queue
nano *<filename>*	Will open file *<filename>* in a text file editor

Thank You
Questions?