



Before You Begin

Introduction to Unix and HPC

Agenda – Day 1 – Shell Primer

9:45	Welcome
10:00 – 11:15	Session 1 (Intro & Unit 1)
11:15 – 11:30	Short break
11:30 – 12:30	Session 2 (Unit 2+3)
12:30 – 1:30	Lunch
13:30– 14:45	Session 3 (Unit 4)
14:45 – 15:00	Short Break
15:00 – 16:15	Session 4 (Unit 5)

Agenda – Day 2 – Data & HPC

9:45 – 11:15	Session 1 (Data – Units 1+2+3)
11:15 – 11:30	Short break
11:30 – 12:30	Session 2 (Data – Units 4+5)
12:30 – 1:30	Lunch
13:30 – 15:30	Session 3 (HPC)

General Introduction

- Intersect <http://www.intersect.org.au/>
 - Who we are?
 - Your Trainer
- Your University IT Contacts
- General Housekeeping
 - Toilets
 - Coffee & Water Facilities
 - Emergency Exits

Course Information

- Who is the course intended for?
- What will be covered during the 2 days?
 - More details in the course outline at <http://www.intersect.org.au/course-resources>
- What level will you be at upon completion of the course?
- **Remember – It's YOUR course, so ask questions!**

Download training materials

Training Material available from:

<http://www.intersect.org.au/course-resources>

- Download the Slides and Exercises and save to a local directory, e.g. C:\HPC:
 - **Course Outlines:** Shell Primer, Data, HPC
 - **Slides:** Shell Primer, Data, HPC Slides
 - **Exercises**

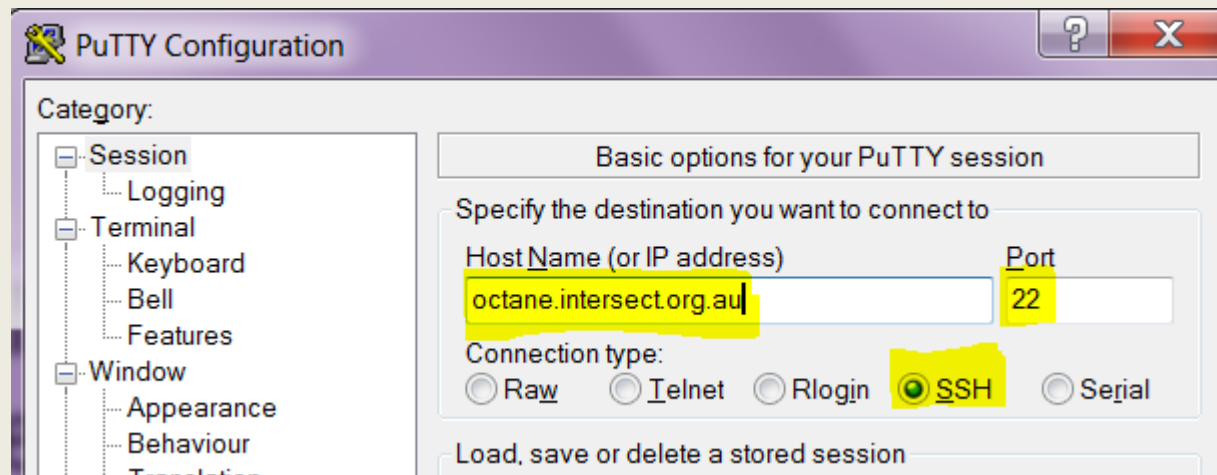
Install software

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)
- Double click to install on your PC.

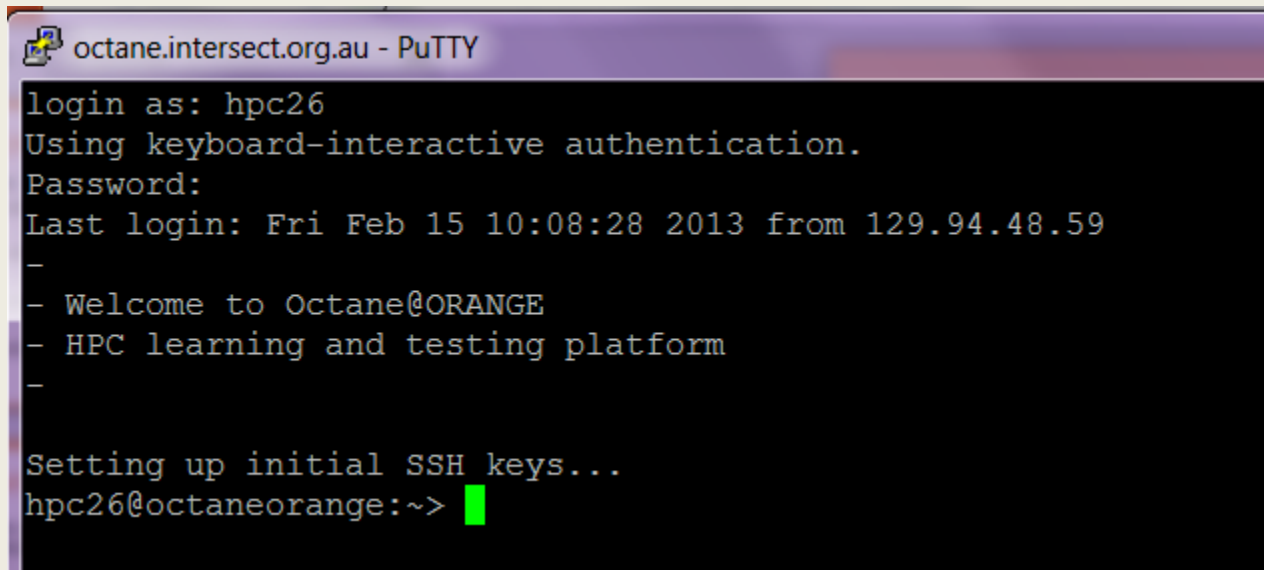
Logging On

- Start the putty telnet/ssh client by double clicking on putty.exe and connect to the HPC Machine
 - Host: `octane.intersect.org.au`
 - 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:** hpc26
 - **Password:** train26



```
octane.intersect.org.au - PuTTY
login as: hpc26
Using keyboard-interactive authentication.
Password:
Last login: Fri Feb 15 10:08:28 2013 from 129.94.48.59
-
- Welcome to Octane@ORANGE
- HPC learning and testing platform
-
Setting up initial SSH keys...
hpc26@octaneorange:~>
```



Shell Primer

Introduction to Unix and HPC

Unit 1: Run Spot, Run

Goals:

- Can run commands, including when there are spaces in arguments
- Can use 'man' to find out more about commands, including arguments and parameters

UNIX Operating System

- UNIX was initially developed by AT&T employees at Bell Labs in 1969
- Common Variants today: Linux, OpenSolaris, FreeBSD, HP/UX, AIX
- 100's of Major Linux Distributions: Debian-based (e.g. Ubuntu), Red Hat Package Manager (RPM)-based (SUSE, Red Hat & Fedora)
 - **Octane uses SUSE LINUX**

What is a Shell?

- The shell is a command line interpreter or shell that provides an interface to the UNIX 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

```

    jat569@vayu1:~ — ssh — 125x65
Last login: Thu May 12 09:13:48 on console
joes-macbook-pro:~ joe$ ssh jat569@vayu.nci.org.au
#####
#               NCI National Facility               #
# This service is for authorised clients only. It is a criminal #
# offence to:                                     #
# 1. Obtain access to data without permission          #
# 2. Damage, delete, alter or insert data without permission #
# Use of this system requires acceptance of the Conditions of Use #
# published at http://nf.nci.org.au/policies/ #
#####
| Welcome to the NCI National Facility! |
| vayu.nci.org.au - 11936 processor InfiniBand x86_64 cluster |
| Assistance: help@nf.nci.org.au   Information: http://nf.nci.org.au |
#####

Jul 17 Automatic expiry of /short files is now in effect. The list of files
subject to deletion can be found in /short/file_expiry/.
*** Disabled until Mar 2011 due to filesystem bug ***

Dec 21 The default version of OpenMPI is now 1.4.3 (was 1.3.3). Although
it is not necessary to rebuild your MPI applications, we strongly
recommend that you do so. Application with non-standard process
layout or imbalanced memory may need specific mpirun options.

Jan 12 Use mpirun --debug ./. to use totalview with MPI jobs.

Mar 18 All vayu nodes have been running Linux 2.6.32 for a couple of weeks.
Anyone using performance counters can now remove requests for
-lother=perf_events from their batch jobs.

Data Transfers: Please use copyq jobs for data transfers in preference to
interactive commands on the login nodes. Also transfers
initiated remotely should be directed to vu-dm.nci.org.au
which will use faster data mover nodes.

#####
# Tuesday, May 10, 10pm: A storage hardware problem on a fileserver #
# hung the filesystem for about 2 hrs. Some jobs may have hit #
# their walltime limit in this time, others may have failed. #
#####

Project being charged for this session is q77
rash: You have no active project, Your per-process limit will be
reduced to 60 seconds.
09:55:09 jat569@vayu1 ~>

```

More on UNIX Shells

- This course uses BASH
 - BASH = the Bourne-Again Shell
 - Default shell for most Linux systems
- Other common Shells include:
 - C Shell (csh)
 - K Shell (ksh)
 - TENEX C Shell (tcsh)

What does BASH do?

- The shell (command line interpreter) interprets the commands entered by the user and passes those to the UNIX 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

hpc85@octaneorange:~> cmd arg1 arg2 arg3

command prompt command Arguments (3)

hpc85@octaneorange:~> cmd arg1 arg2 arg3 "arg 4"

command Arguments (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
<code>ls</code>	<i>Shows a directory listing</i>
<code>w</code>	Shows who is logged on to the system and what they are doing
<code>w hpc22</code>	Shows login, idle time, and what user hpc22 is doing
<code>finger</code>	Shows login, username & login details of users on the system
<code>finger hpc22</code>	Shows user hpc22's login details including name, idle time, login time as well as some of their environment settings
<code>date</code>	Prints the system time and date
<code>uptime</code>	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!

Sample Command & Flags

- `ls -a`
 - short form flag “-a”
 - Shows all files incl. files starting with .
- `ls --all`
 - long form flag “--all”
 - Shows all files incl. files starting with .

More Commands & Flags

– `ls -l`

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

Combining Flags

- Flags can be **combined in any order** but still mean the same thing
- The following command and flags all show the directory contents and show one file per line:

- `ls -a -l`
- `ls -l -a`
- `ls -la`
- `ls -al`
- `ls --all -l`
- `ls -l -all`

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
<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 -a</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
<code>ls --format=single-column</code>	Lists directory contents of current directory in a single column

Using the “man” pages

- All Unix systems come equipped with manual or “man” pages which contain documentation about how each in-built command works
- Man pages can be accessed using the following command format:

man program, e.g. **man** ls

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)

Exercise 1(c)

The Calendar

Command	Description
man <i><command name></i>	Print the online manual entry for <i><command name></i>
cal	Print a calendar

Unit 2: Where am I?

Goals:

- Can explore navigate a hierarchy of directories
- Can specify files with relative and absolute paths
- Can create a hierarchy of directories
- Can copy and move files between directories

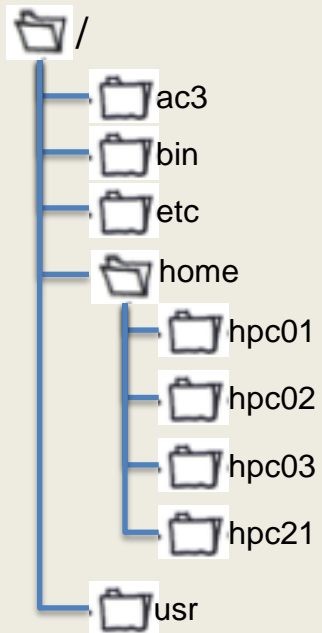
UNIX Directories

- Called 'folders' under windows
 - Exactly the same concept
- You run into them a LOT more under Unix 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 “/” (**root folder**)?

/ac3
/bin
/etc
/home
/usr



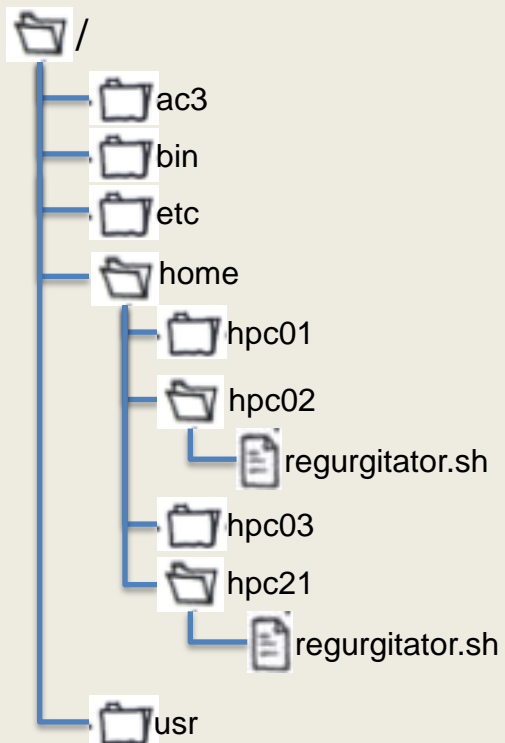
What are the names of the contents of the directory called **/home**?

/home/hpc01

/home/hpc02

/home/hpc03

/home/hpc21

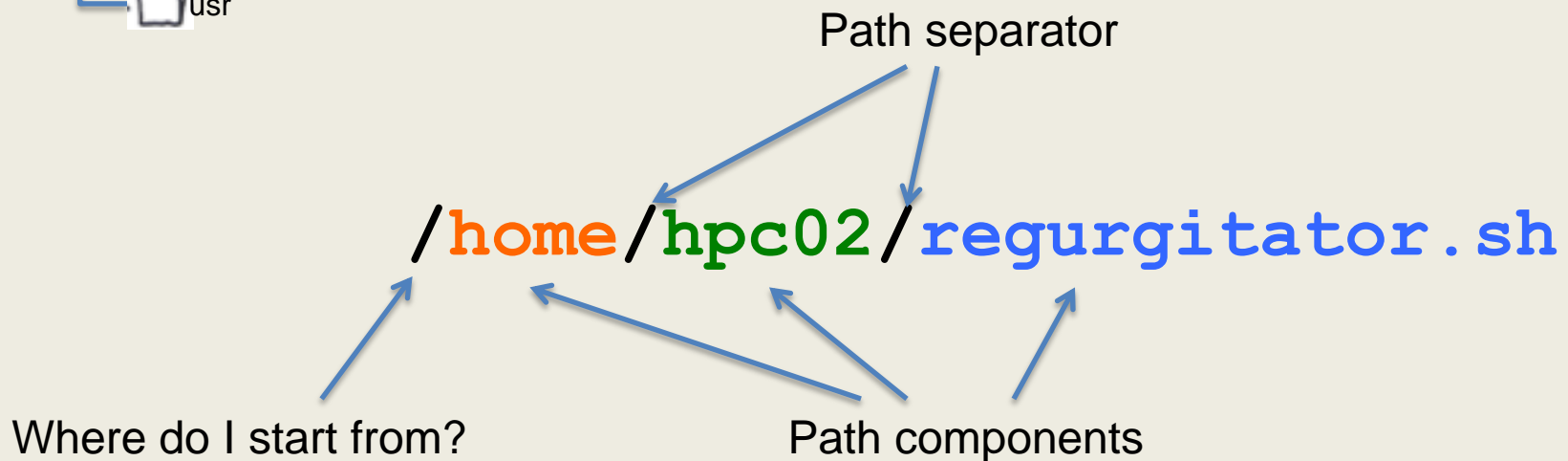
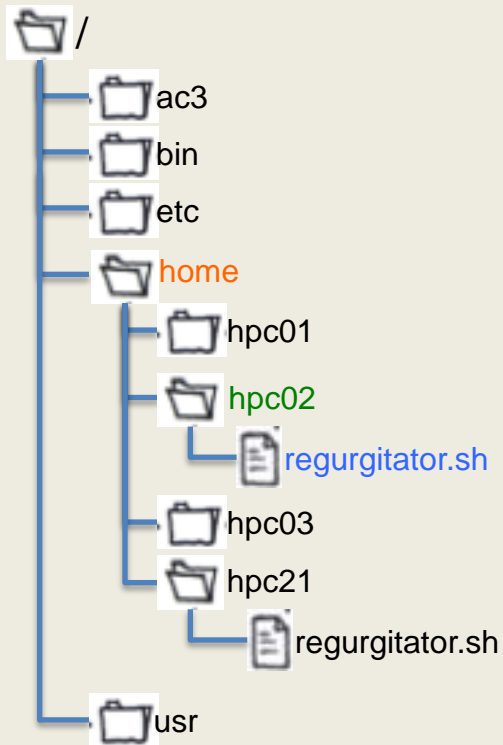


What are the names of the contents of the directory called **/home/hpc02**?

`/home/hpc02/regurgitator.sh`

What are the names of the contents of the directory called **/home/hpc21**?

`/home/hpc21/regurgitator.sh`

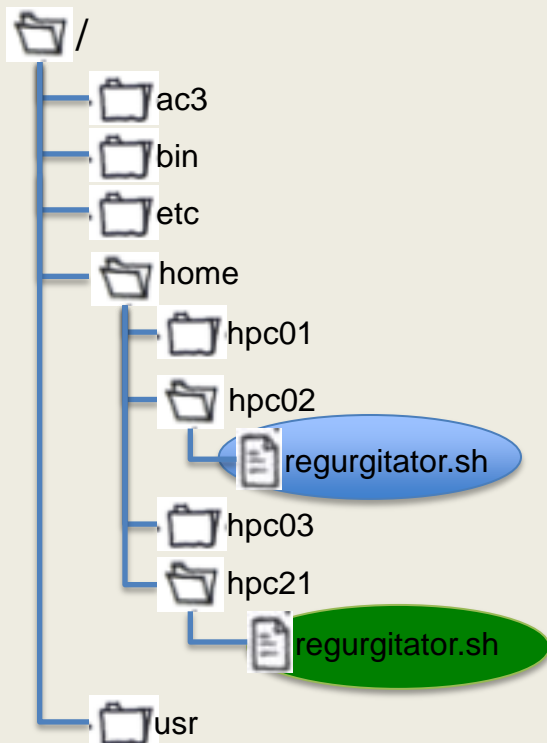


Rules So Far

- 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"



What are the names of the contents of the directory called ~

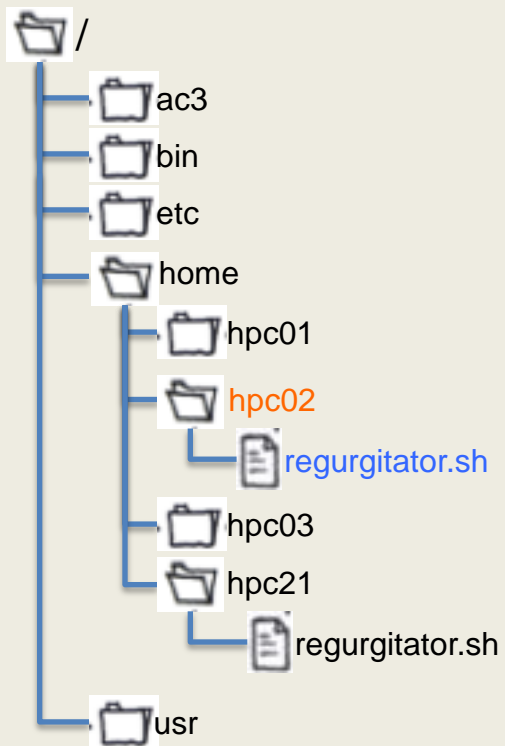
It depends who asks: If the user “hpc02” asks, then the answer is
`/home/hpc02/regurgitator.sh`

What are the names of the contents of the directory called ~hpc21

`/home/hpc21/regurgitator.sh`

What is the absolute path of the file ~hpc02/regurgitator.sh

`/home/hpc02/regurgitator.sh`



Path separator

`~hpc02/regurgitator.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”

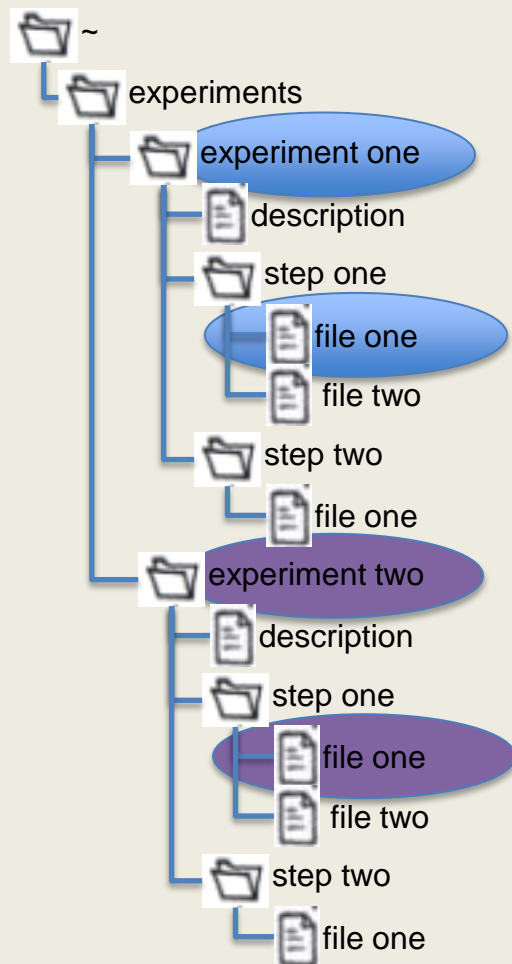
Exercise 2(a)

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>

The Working Directory

- BASH maintains a “working” or “current” directory. That is “the directory that you are currently in”
 - Your prompt will show you your current working directory
- Linux uses the “.” character to denote the working directory. You can use this when running commands
 - `ls -la .` → show the listing for the current dir
- The command “**pwd**” will list the current working directory

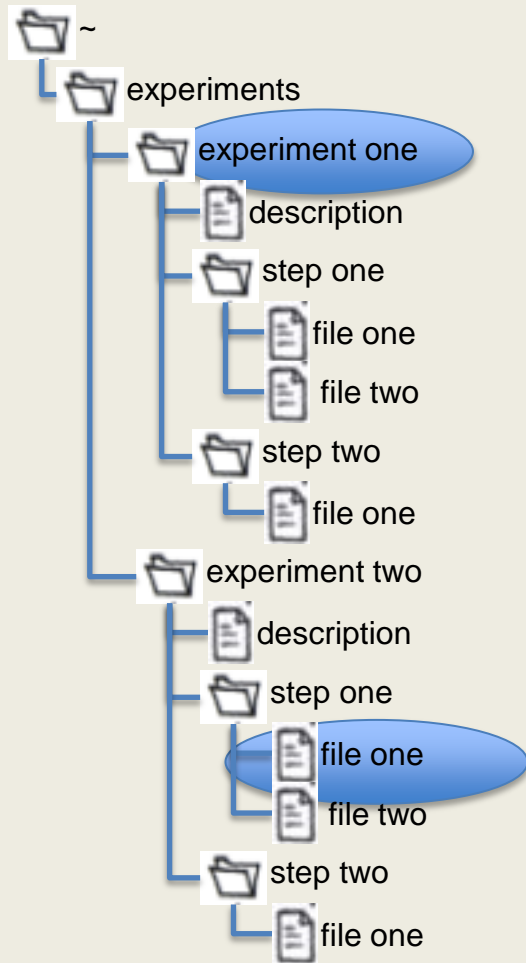


What is the absolute path of
“**step one/file one**”

What is the working directory?

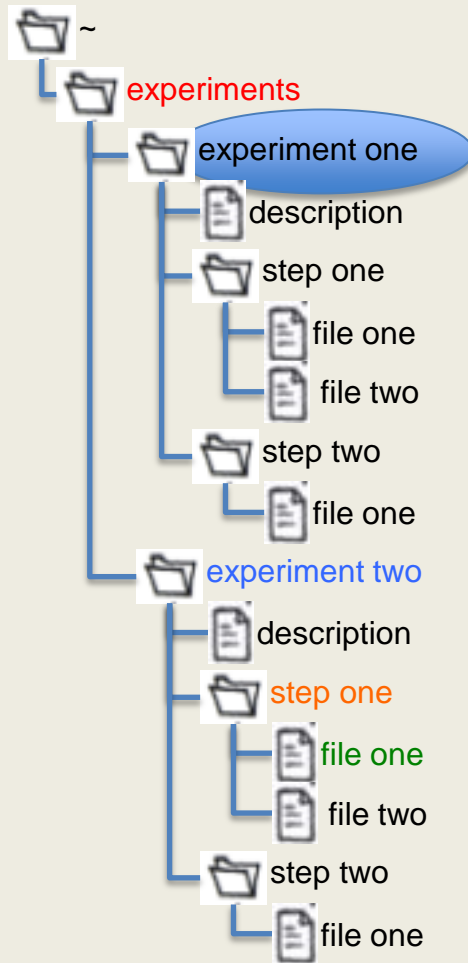
Looking upwards

- 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



What is the absolute path of
`../experiment two/step one/file one`

What is the working directory?



Path separator

../experiment two/step one/file one

Where do I start?

Path components

Everything about directories

- There is one root, called “/”
 - A path starting with “/” means “from the root”
 - A path starting with “~” means “from the home”
 - A path starting with anything else means “from the working directory”
-
- Note – the last and second last rules assume a hidden context (current user and current directory)

Exercise 2(b)

Finding your way around

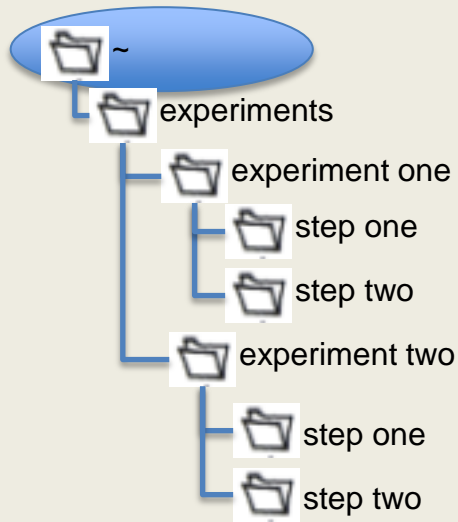
Command	Description
<code>pwd</code>	Print the name of the current working directory
<code>cd ..</code>	Change directory to the parent directory
<code>ls</code>	List the contents of the working directory
<code>ls .</code>	List the contents of the working directory
<code>ls ..</code>	List the contents of the parent directory

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>
```



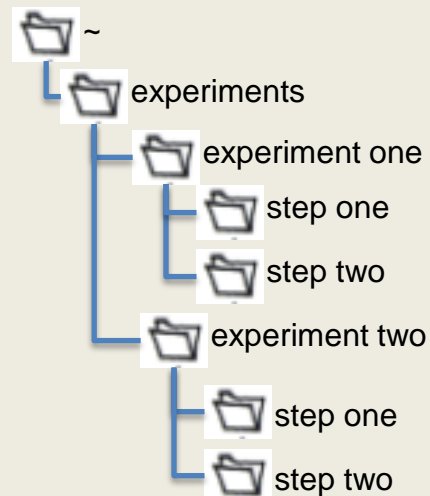
What happens when you execute the commands?

```
mkdir ~/experiments
cd ~/experiments
mkdir "experiment one"
mkdir "experiment one/step one"
mkdir "experiment one/step two"
mkdir "experiment two"
cd "experiment two"
mkdir "step one"
mkdir "step two"
```

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>

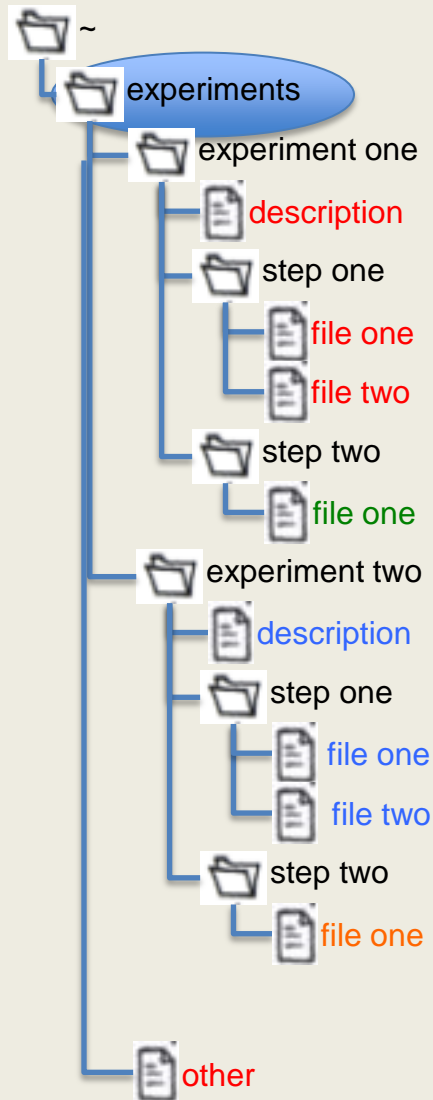


Moving/Copying Files Around

- 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 "experiment one/description" "experiment two"
cd "experiment one"

cp "step one/file one" "../experiment two/step two"
cp "../experiment two/description" ../other
cd "../experiment two/step one"
mv "file one" "../step two"
```

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 ?

Unit 3: It's what's inside that counts

Goals:

- Can look inside files (even really big ones)
- Can obtain rudimentary information about files

What's in a file?

my dame has a lame tame crane
my dame has a crane that is lame
pray, gentle jane, that my dame's lame tame crane
feed and come home again

0000000:	011011010111100100100000011001000110000101101101	my dam
0000006:	011001010010000001101000011000010111001100100000	e has
000000c:	011000010010000001101100011000010110110101100101	a lame
0000012:	001000000111010001100001011011010110010100100000	tame
0000018:	011000110111001001100001011011100110010100001010	crane.
000001e:	011011010111100100100000011001000110000101101101	my dam
0000024:	011001010010000001101000011000010111001100100000	e has
000002a:	011000010010000001100011011100100110000101101110	a cran
0000030:	011001010010000001110100011010000110000101110100	e that
0000036:	001000000110100101110011001000000110110001100001	is la
000003c:	011011010110010100001010011100000111001001100001	me.pra

What's in a file?

This power-point presentation.

```
0000000: 010100000100101100000011000001000001010000000000 PK....
0000006: 000001100000000000000100000000000000000000000000 .....
```

000000c: 001000010000000011010111110011110010000111101110 !...!.
0000012: 00100010000000110000000000000000000000010100011110 ".....
0000018: 000000000000000000000100110000000000000100000000010
000001e: 0101101101000011011011110110111100111010001100101 [Conte
0000024: 011011100111010001011111010101000111100101110000 nt_Typ
000002a: 011001010111001101011101001011100111100001101101 es].xm
0000030: 011011000010000010100010000001000000001000101000 l ... (
0000036: 1010000000000000000000001000000000000000000000000
000003c: 000

What is a 'text file'?

109

104

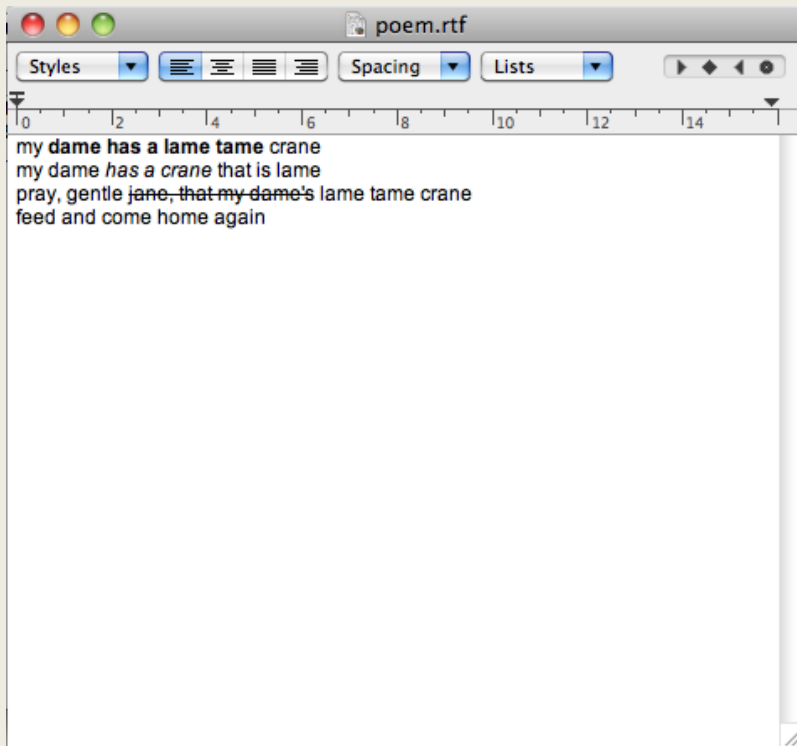
10

0000000:	01101101 0111100100100000011001000110000101101101	my dam
0000006:	0110010100100000 01101000 011000010111001100100000	e has
000000c:	011000010010000001101100011000010110110101100101	a lame
0000012:	001000000111010001100001011011010110010100100000	tame
0000018:	0110001101110010011000010110111001100101 00001010	crane.
000001e:	011011010111100100100000011001000110000101101101	my dam
0000024:	011001010010000001101000011000010111001100100000	e has
000002a:	011000010010000001100011011100100110000101101110	a cran
0000030:	011001010010000001110100011010000110000101110100	e that
0000036:	001000000110100101110011001000000110110001100001	is la
000003c:	011011010110010100001010011100000111001001100001	me.pra

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com

Two sides of the same coin



```
{\rtf1\ansi\ansicpg1252\cocoartf1038\cocoa  
subrtf350
```

```
{\fonttbl{\f0\fswiss\fcharset0 ArialMT;}
```

```
{\colortbl;\red255\green255\blue255;}
```

```
\paperw11900\paperh16840\margl1440\ma  
rgr1440\vieww9000\viewh8400\viewkind0  
\deftab720
```

```
\pard\pardeftab720\ql\qnatural
```

```
\f0\fs24 \cf0 my
```

```
\b dame has a lame tame
```

```
\b0 crane\
```

```
my dame
```

```
\i has a crane
```

```
\i0 that is lame\
```

```
pray, gentle \strike \strikec0 jane, that my  
dame's\strike0\striked0 lame tame crane\
```

```
feed and come home again\
```

```
}
```

Why do we care?

- BASH is more or less committed to ASCII text
- In particular
 - your keystrokes result in ASCII codes being sent to the shell
 - when a program generates output, it is interpreted by the shell as ASCII codes and displayed on the screen (even some non-printables)

Exercise 3(a)

Looking into files

Command	Description
<code>ls</code>	You've met before
<code>cat <filename></code>	Print a file to the terminal (cat enate)
<code>less <filename></code>	Like <code>cat</code> , but <code>less</code> at a time
<code>head <filename></code>	Like <code>cat</code> , but just the top of the file
<code>tail <filename></code>	Like <code>cat</code> , but just the end of the file
<code>wc <filename></code>	W ord c ount – count the words in a file
<code>du</code>	D isk u sage – how much space is used

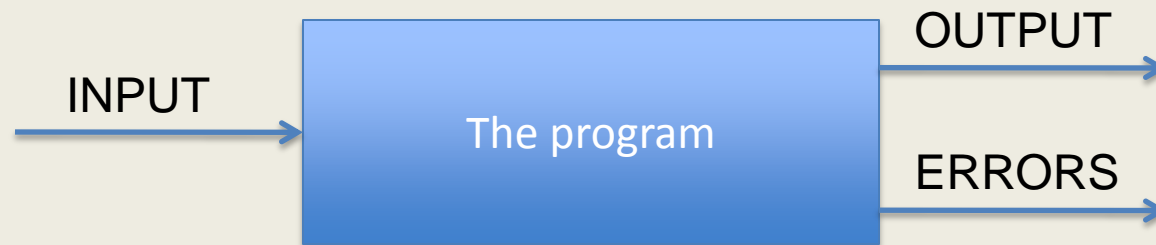
Unit 4: It's all just files

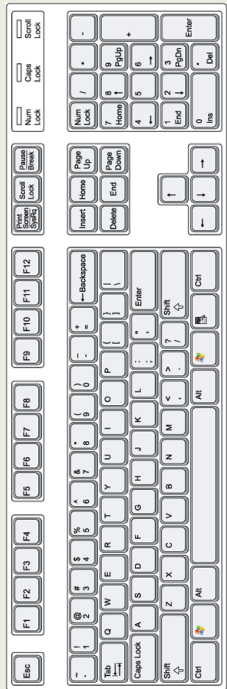
Goals:

- Can save the output of programs into files
- Can chain commands together

The core magic of the Linux command line

- The next bit is the core magic of Linux, as far as this module goes.
 - The keyboard is just a file
 - The console is just a file
- Define – What is input?
- Define – What is output?





INPUT

The program

command

OUTPUT

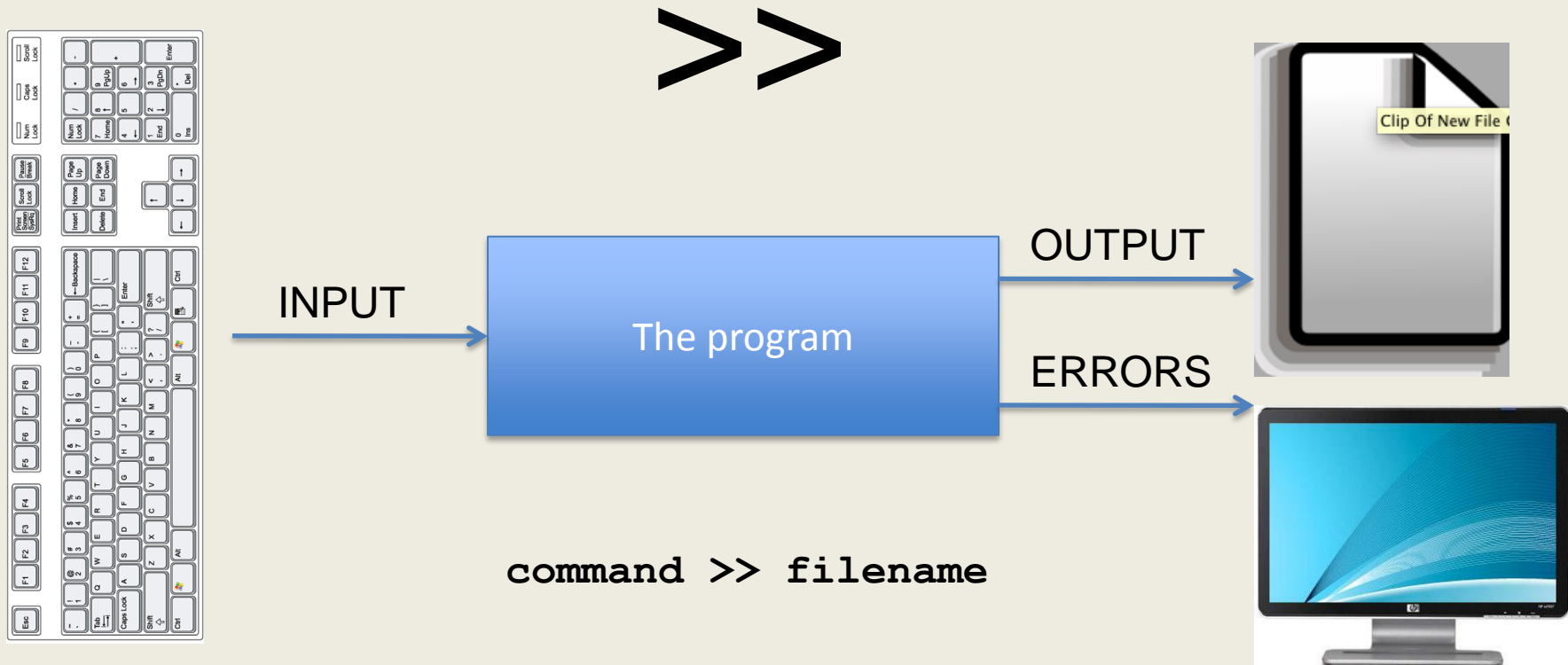
ERRORS



Redirect OUTPUT elsewhere...

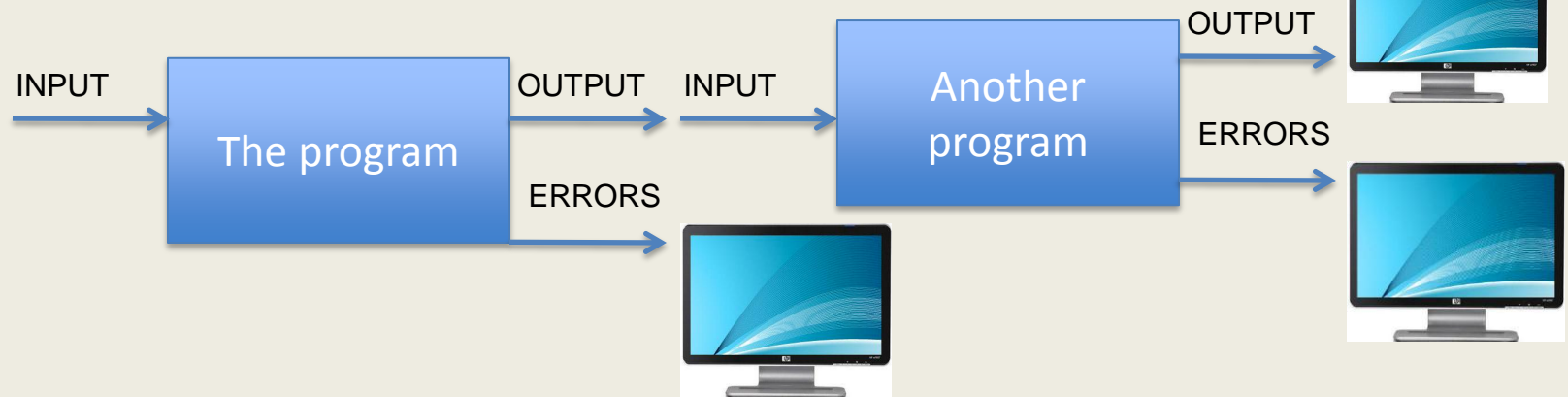


Append OUTPUT elsewhere...



Pipe OUTPUT to a command...

|



`command | another_command`

Exercise 4(a)

Looking into files

Command	Description
fortune	Tells your fortune – really! Try it!
sort <i><file></i>	Sorts the contents of <i><file></i> , or input if no file is specified.
uniq <i><file></i>	Removes any redundant lines in <i><file></i> , or input if no file is specified. Redundant lines have to be adjacent to be considered redundant.
cat <i><file></i>	Prints the contents of <i><file></i>
<i><command></i> > <i><file></i>	Run <i><command></i> but rather than print the output to the console, redirect the output to <i><file></i> .
<i><command></i> >> <i><file></i>	Run <i><command></i> but rather than print the output to the console, append the output to <i><file></i> .
<i><command></i> <i><command></i>	Pipe one <i><command></i> to another <i><command></i>

Unit 5: Products of our environment

Goals:

- Can set and read environment variables
- Can add programs to the PATH
- Can query the environment for which variables exist

Case study – finding programs

- Recall from previous units
 - Sometimes you have to specify the path of a program

```
regurgitator.sh
```

```
./regurgitator.sh
```

```
../regurgitator.sh
```

```
/bin/regurgitator.sh
```


PATH

- There is a 'special' thing called the PATH
 - Part of what makes it special is encoded
 - Part of what makes it special is convention
- There is an environment full of variables
 - One of those variables is called PATH
 - Use '\$' to tell BASH you're talking about a variable
- To check what an environment variable is set to, use `$<ENVIRONMENT_VARIABLE>`
 - e.g. `$PATH`

A sample path

```
/bin:/usr/bin:/usr/local/bin:.
```

What does BASH do?

- When you hit return
 - BASH parses the command line into tokens
 - Any 'special' tokens are expanded
 - The first token is treated as the command
 - The remaining tokens are passed to the command as arguments

A simple example

- Your home directory is stored in the environment, you can be taken there directly

```
cd $HOME
```

Exercise 5(a)

Realistic Evaluation of the Environment

Command	Description
<code>\$<VARIABLE></code>	Evaluates to the contents of the variable <code><VARIABLE></code> .
<code>env</code>	Prints all environment variables and their values.
<code>echo <arg1> <arg2> ...</code>	Prints out its arguments, with no additional information.
<code>grep <pattern> <file></code>	<p>Prints out all occurrences of <code><pattern></code> in <code><file></code>. If no file is specified, processes standard input. For example</p> <p style="text-align: center;"><code>env grep PA</code></p> <p>will print all variables containing 'PA' in them or their value.</p>

Manipulating the Environment

- You can use the environment to your advantage.
 - Update the path
 - Change your prompt
 - Use the environment to share information between your scripts

The export command

The **export** command is used to set the value of an environment variable:

```
export <VARIABLE_NAME>=<VALUE>
```

Exercise 5(b)

Manipulating the Environment

Command	Description
export <i><VAR>=<VAL></i>	Set a variable called <i>VAR</i> and give it contents <i>VAL</i> . The variable will be available to programs invoked from the shell.

Summing up – Your Skills

- You have now got the rudimentary skills to move around, make and view files, chain commands together and manipulate the environment

Summing up – Hard Lessons

- BASH is wildly picky
- Silent success
- Command-line work is flexible but error prone
- There's a lot of hidden context
 - ENVIRONMENT, WORKING DIRECTORY, STDOUT/STDIN, COMMAND LINE EXPANSION
- BASH is wildly picky

Summing up – the positives

- Command line gives you a more **powerful** means to control a program
- Allows **direct interaction** with shell commands (no GUI needed)
- Allow for **ease of automation** via scripting
- UNIX is more **flexible** and can be installed on many different types of machines (e.g. mainframes/supercomputers)
- Unix possesses much greater **processing power** than Windows.