Before You Begin

Introduction to Unix for HPC

Raijin version

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)

INTERSECT

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**

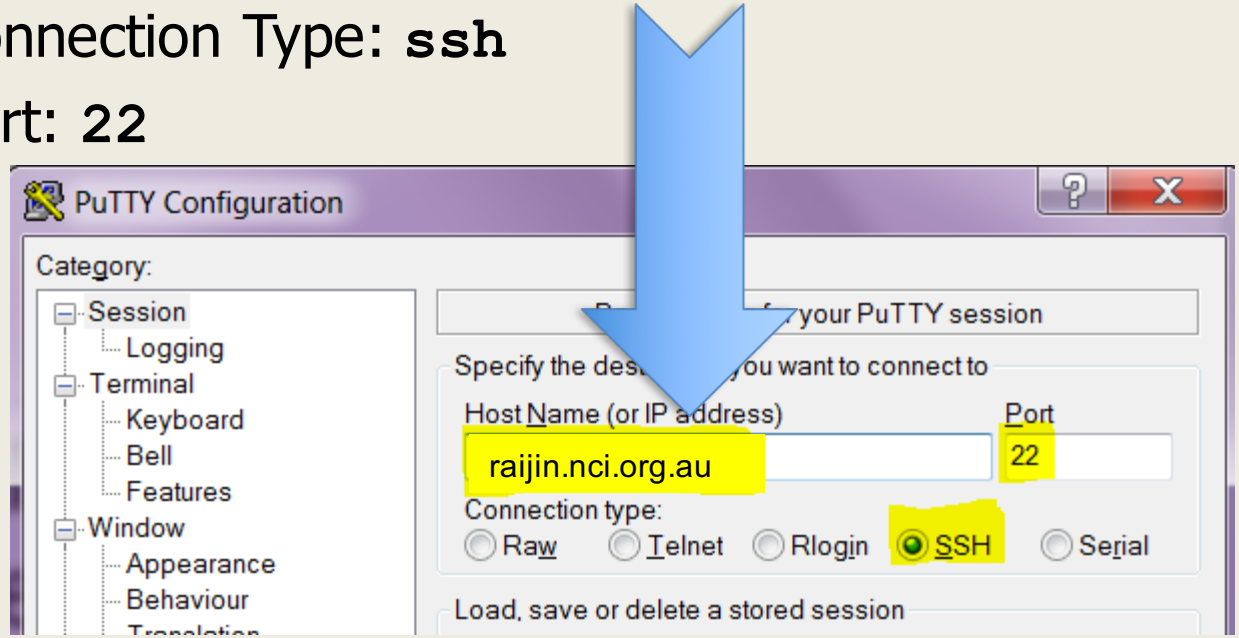
Install software

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: **`raijin.nci.org.au`**
  - Connection Type: **`ssh`**
  - Port: **`22`**



INTERSECT

Log On to Raijin

- Log into Raijin using the Test Account allocated to you, e.g.
  - **Account Name:** iaa444 – ibx444
    - iaa, iab, iac… iax, iba, ibb, ibc …ibx
  - **Password:** provided by your instructor

Shell Primer

Introduction to Unix for 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)
    - **Rajin uses CentOS**

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

```
Last login: Thu Apr 13 13:54:47 2017 from 202.7.176.94
################################################################################
#                  Welcome to the NCI National Facility!                       #
#      This service is for authorised clients only. It is a criminal           #
#      offence to:                                                             #
#                - Obtain access to data without permission                    #
#                - Damage, delete, alter or insert data without permission     #
#      Use of this system requires acceptance of the Conditions of Use         #
#      published at http://nci.org.au/conditions/                              #
################################################################################
|        raijin.nci.org.au - 82408 processor InfiniBand x86_64 cluster         |
|     User Guide:       http://help.nci.org.au   Assistance: help@nci.org.au   |
|     Information:      http://nci.org.au                                      |
|     Downtime Notices: http://nci.org.au/raijin-status/                       |
================================================================================

Jan 12  Additional Broadwell CPU Nodes Available
    NCI has installed an additional 814 Broadwell-based compute nodes in
    Raijin. For instructions on the use of these nodes, please visit:
    http://nci.org.au/broadwell

Feb 16  User Job History
    Users can now review completed jobs here:
    https://usersupport.nci.org.au/report/job_history

Mar 2  NVIDIA P100 Compute Nodes Online
    NCI has installed new compute nodes with 4x NVIDIA P100 GPUs in each.
    These are available under the 'gpupascal' queue.

    To use these new nodes, your application would need to have been built
    against CUDA 8.0 (or newer). Details: https://nci.org.au/gpu

================================================================================
[aw7523@raijin6 ~]$ 4$$
```


INTERSECT

More on UNIX Shells

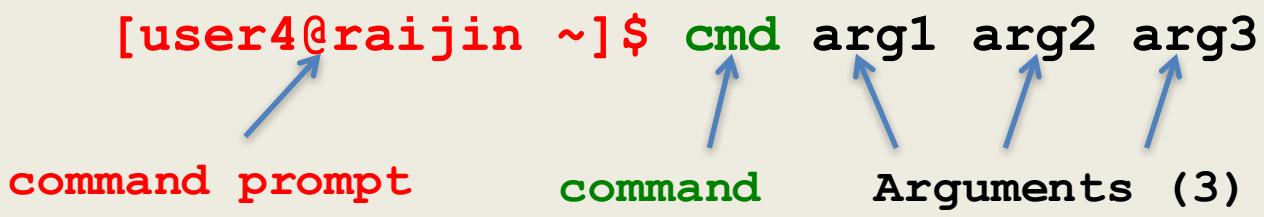
- This course uses BASH
  - BASH = the Bourne-Again Shell
  - Default shell for most Linux systems
- Other common Shells include:
  - C Shell (csh)
  - K Shell (ksh)
  - TENEX C Shell (tcsh)

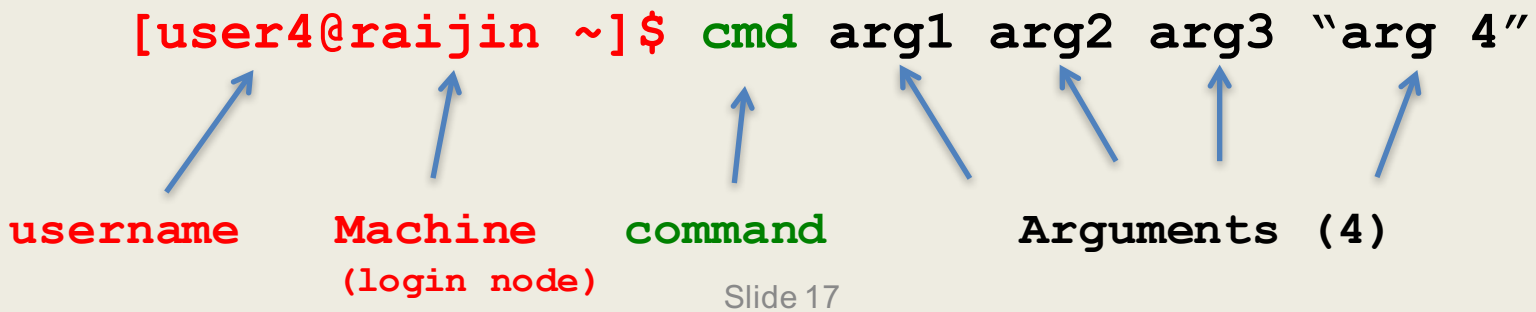
What does BASH do?

- 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





INTERSECT

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 iaa444`	Shows login, idle time, and what user iaa444 is doing
`finger`	Shows login, username & login details of users on the system
`finger iaa444`	Shows user iaa444's login details including name, idle time, login time as well as some of their environment settings
`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!

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
`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
`ls --format=single-column`	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 *<command name>*	Print the online manual entry for *<command name>*
cal	Print a calendar

Unit 2: Where am I?

Goals:

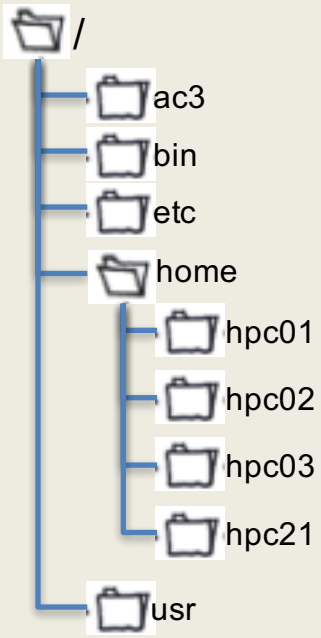
- Can explore navigate a hierarchy of directories
- Can specify files with relative and absolute paths
- Can create a hierarchy of directories
- Can copy and move files between directories

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

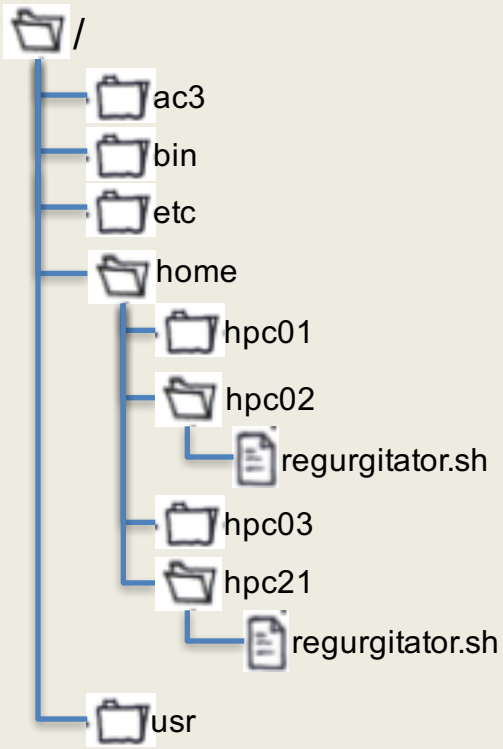
What are the names of the contents of the directory called "**`/" (root folder)`**?

`/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`

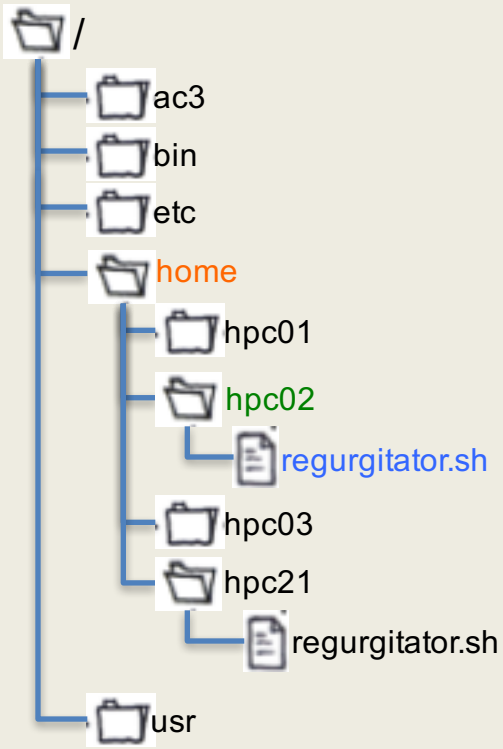


What are the names of the contents of the directory called **`/home/hpc02`**?

`/home/hpc02/regurgitator.sh`

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

`/home/hpc21/regurgitator.sh`



Path separator

/home/hpc02/regurgitator.sh

Where do I start from?

Path components

INTERSECT

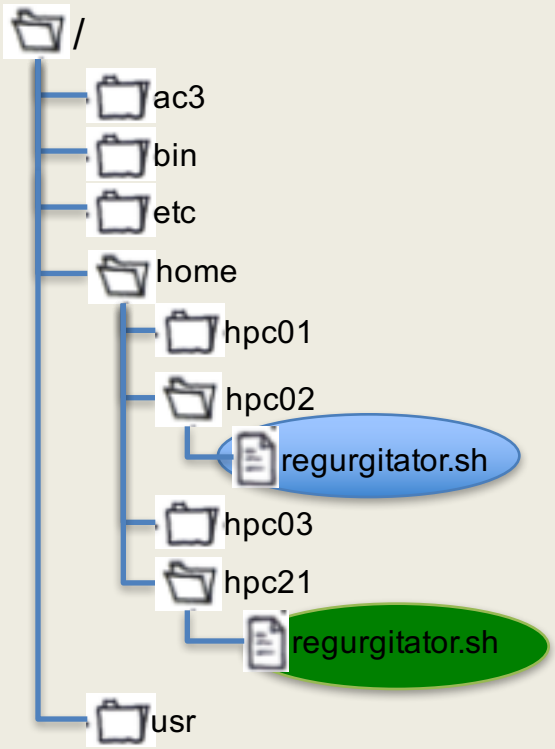
Rules So Far

- 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"
  - `~iaa444` "iaa444's home directory"



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

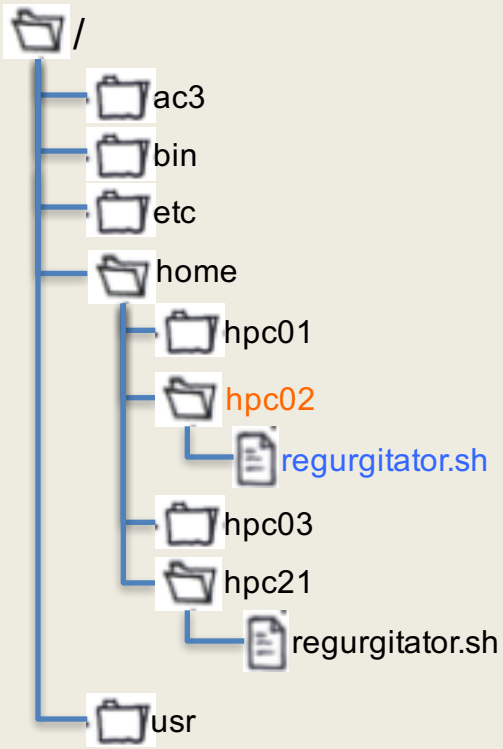
It depends who asks: If the user "hpc02" asks, then the answer is `/home/hpc02/regurgitator.sh`

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

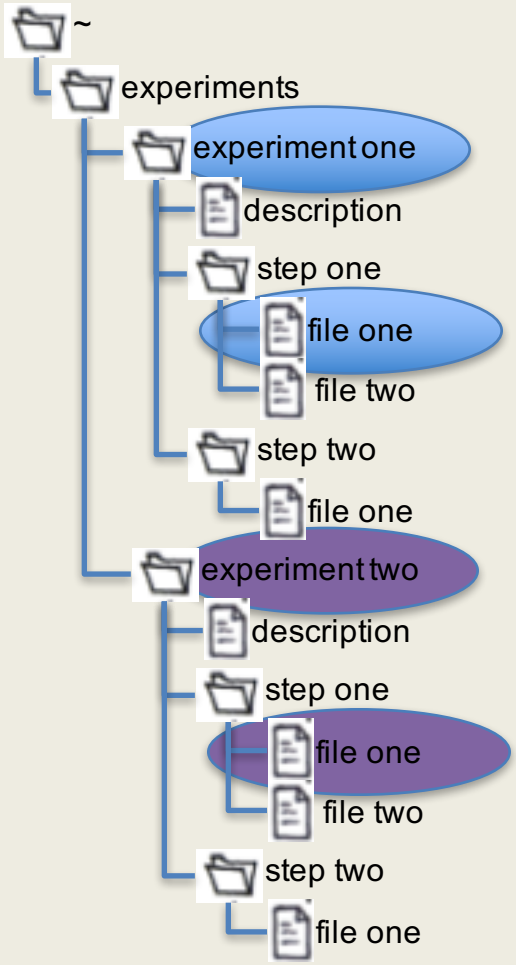
Command	Description
`cd`	Change the working directory to your home directory
`cd` *`<path>`*	Change directory to *`<path>`*
`ls`	List the contents of the working directory
`ls` *`<path>`*	List the contents of *`<path>`*

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

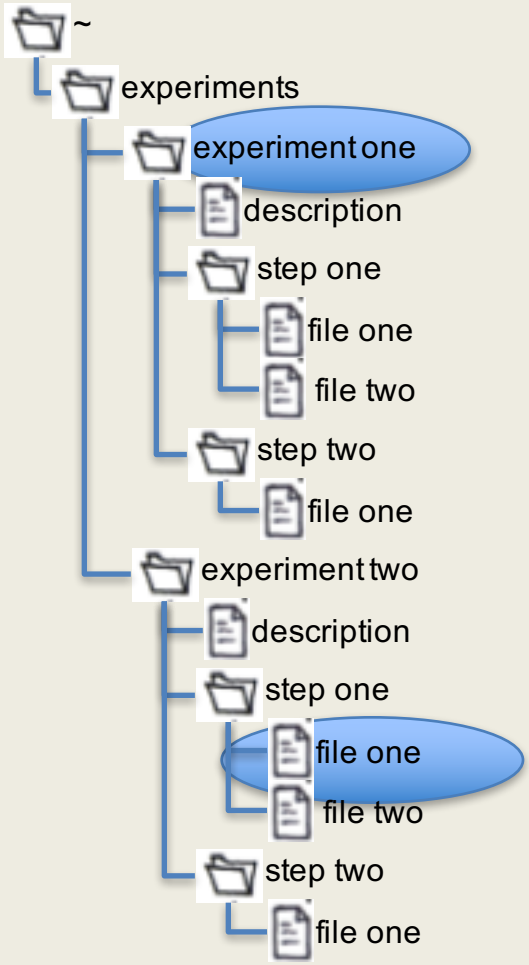


What is the relative path of "**`step one/file one`**"

What is the working directory?

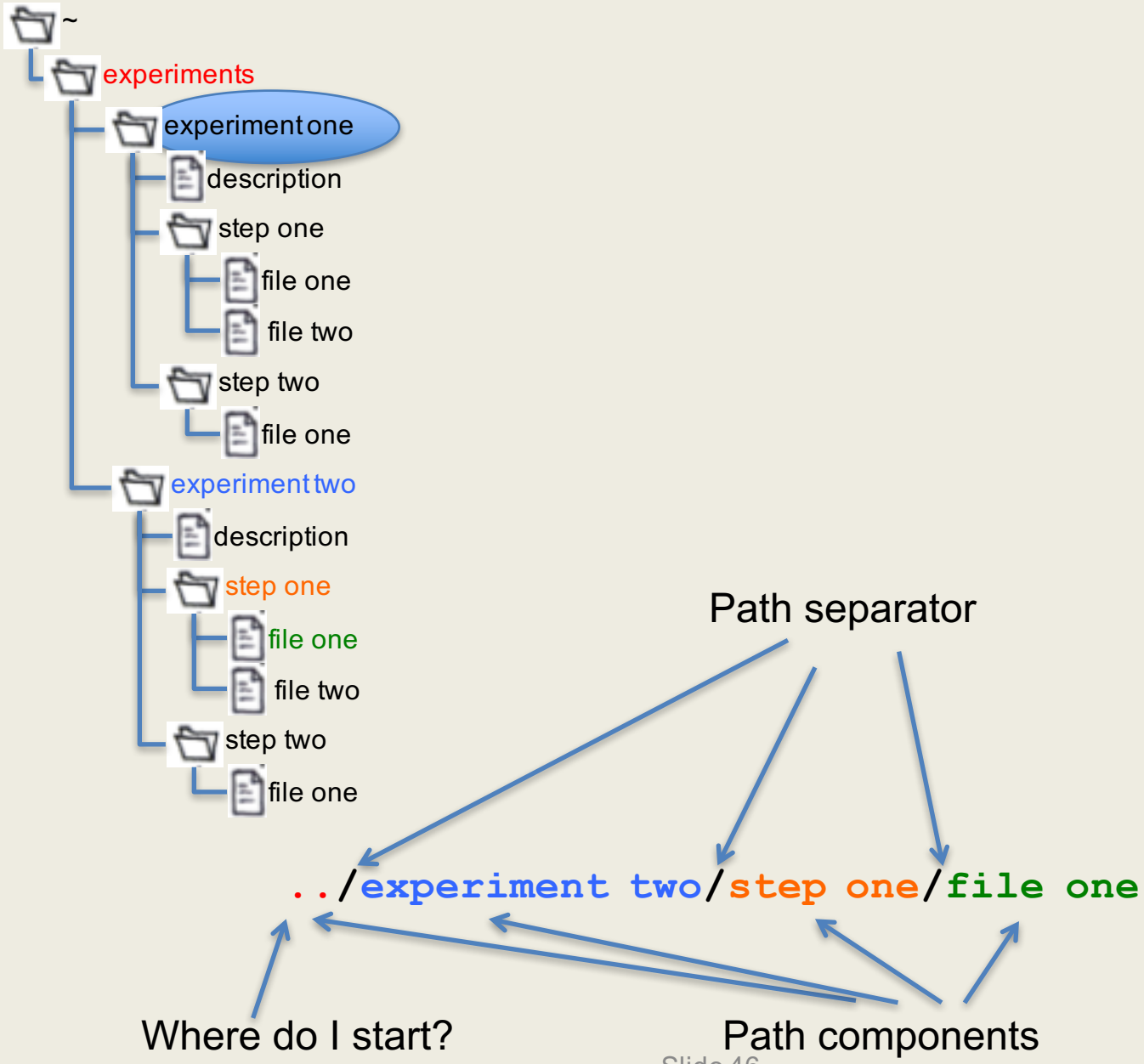
Looking upwards

- 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 relative path of
`../experiment two/step one/file one`

What is the working directory?


~
experiments
experiment one
description
step one
file one
file two
step two
file one
experiment two
description
step one
file one
file two
step two
file one
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)

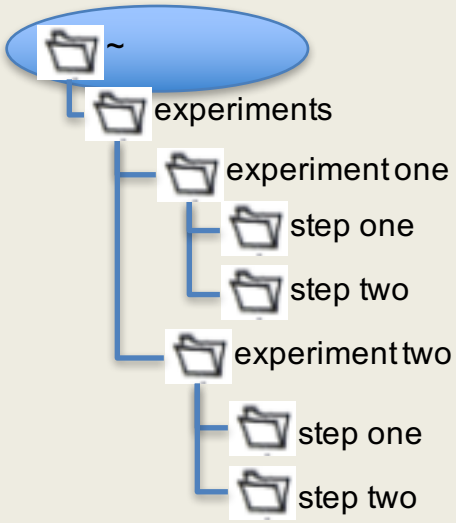
Finding your way around

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



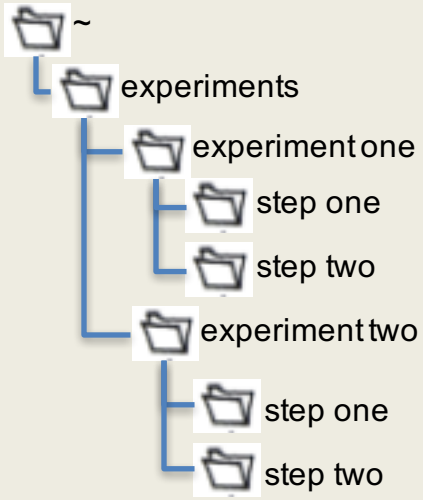
What happens when you execute the commands?

```
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` *<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>*

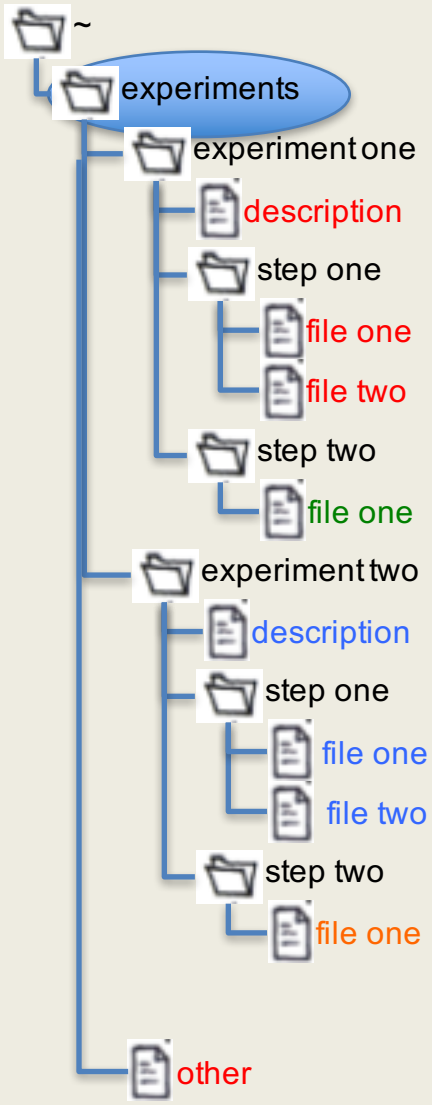


INTERSECT

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 *<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**?

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: 01101101011110010010000001100100011000010110110 1  my dam
0000006: 01100101001000000110100001100001011100110010000 0  e has
000000c: 01100001001000000110110001100001011011010110010 1  a lame
0000012: 00100000011101000110000101101101011001010010000 0   tame
0000018: 01100011011100100110000101101110011001010000101 0  crane.
000001e: 01101101011110010010000001100100011000010110110 1  my dam
0000024: 01100101001000000110100001100001011100110010000 0  e has
000002a: 01100001001000000110001101110010011000010110111 0  a cran
0000030: 01100101001000000111010001101000011000010111010 0  e that
0000036: 00100000011010010111001100100000011011000110000 1   is la
000003c: 01101101011001010000101001110000011100100110000 1  me.pra
```

What's in a file?

This power-point presentation.

```
0000000: 0101000001001011000000110000010000010100000000000  PK....
0000006: 0000011000000000000010000000000000000000000000000  ......
000000c: 0010000100000000110101111100111100100001111011110  !...!.
0000012: 0010001000000011000000000000000000000010100011110  ".....
0000018: 0000000000000000000100110000000000000100000000010  ......
000001e: 0101101101000011011011110110111001110100011001010  [Conte
0000024: 0110111001110100010111110101010001111001011100000  nt_Typ
000002a: 0110010101110011010111010010111001111000011011010  es].xm
0000030: 0110110000100000101000100000010000000010001010000  l ...(
0000036: 1010000000000000000001000000000000000000000000000  ......
000003c: 0000000000000000000000000000000000000000000000000  ......
```


INTERSECT

What is a ‘text file’?

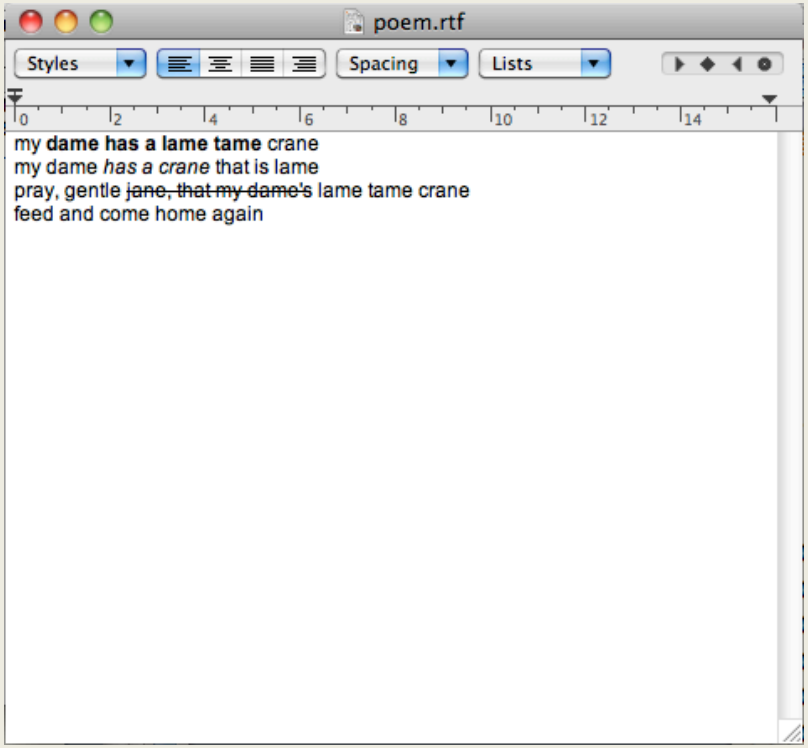
109 104 10

```
0000000: 01101101011110010010000001100100011000010110110 1  my dam
0000006: 0110010100100000 01101000 0110000101110011001000 00  e has
000000c: 0110000100100000011011000110000101101101011001 01  a lame
0000012: 0010000001110100011000010110110101100101001000 00   tame
0000018: 0110001101110010011000010110111001100101 00001010  crane.
000001e: 0110110101111001001000000110010001100001011011 01  my dam
0000024: 0110010100100000011010000110000101110011001000 00  e has
000002a: 0110000100100000011000110111001001100001011011 10  a cran
0000030: 0110010100100000011101000110100001100001011101 00  e that
0000036: 0010000001101001011100110010000001101100011000 01   is la
000003c: 0110110101100101000010100111000001110010011000 01  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	&#127;	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\
}
```


INTERSECT

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

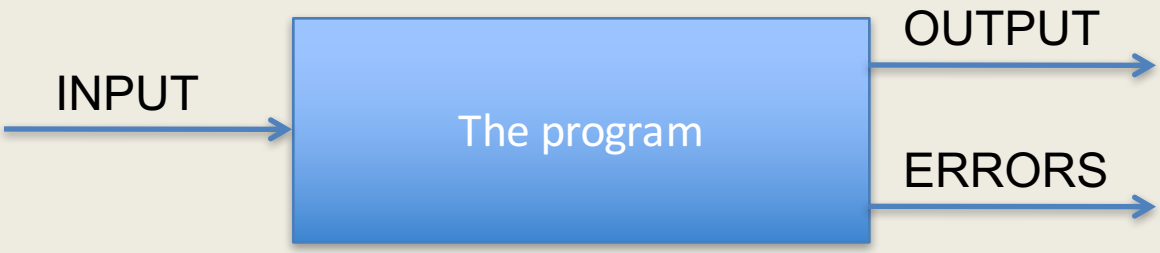
Unit 4: It's all just files

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

OUTPUT

ERRORS

command

Redirect OUTPUT elsewhere...

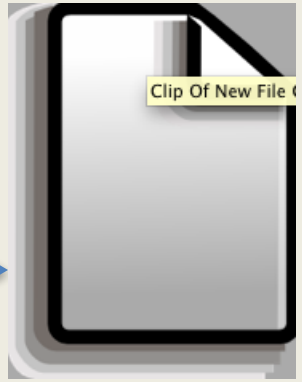
>

INPUT

The program

OUTPUT

ERRORS



```
command > filename
```

Append OUTPUT elsewhere...

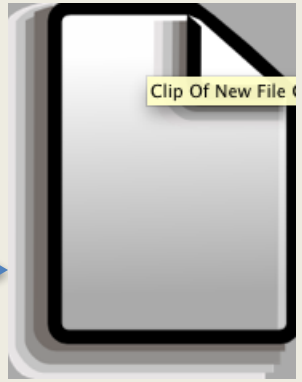
>>

INPUT

The program

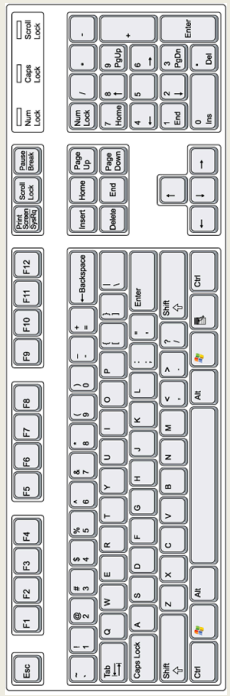
OUTPUT

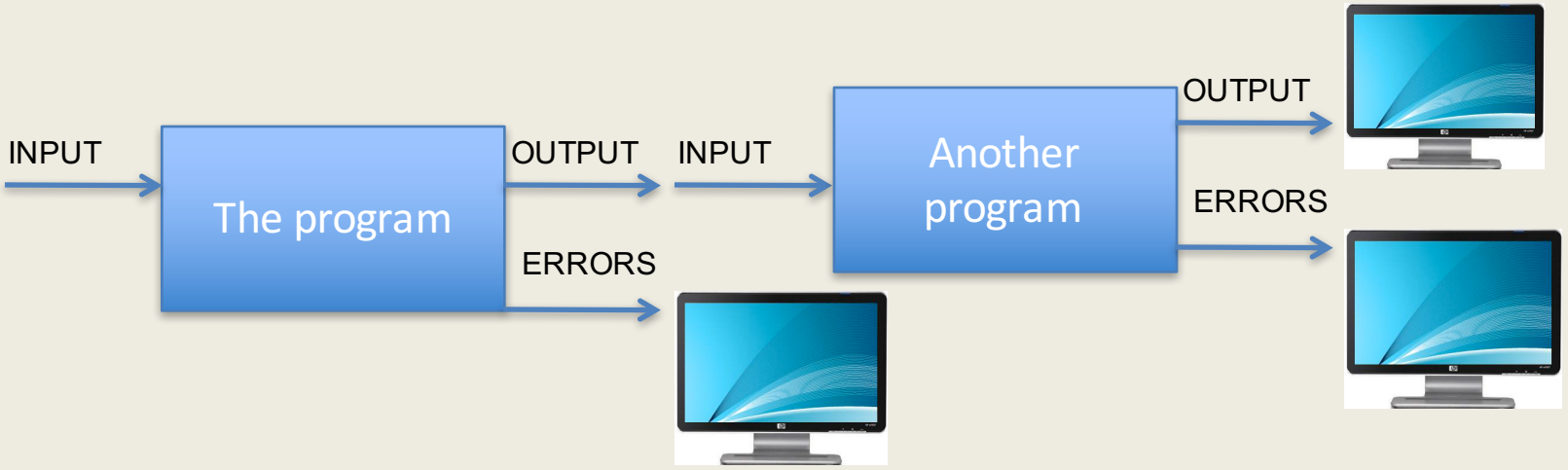
ERRORS



```
command >> filename
```

Pipe OUTPUT to a command...






```
command | another_command
```


INTERSECT

Exercise 4(a)

Looking into files

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

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
$<*VARIABLE*>	Evaluates to the contents of the variable <*VARIABLE*>.
env	Prints all environment variables and their values.
echo <*arg1*> <*arg2*> …	Prints out its arguments, with no additional information.
grep <*pattern*> <*file*>	Prints out all occurrences of <*pattern*> in <*file*>. If no file is specified, processes standard input. For example **env | grep PA** will print all variables containing 'PA' in them or their value.

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 *<VAR>*=*<VAL>*	Set a variable called *VAR* and give it contents *VAL*. 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.