Before You Begin

Introduction to Unix for HPC
Raijin version

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: Shell, Data, HPC
- 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 save to a local directory.

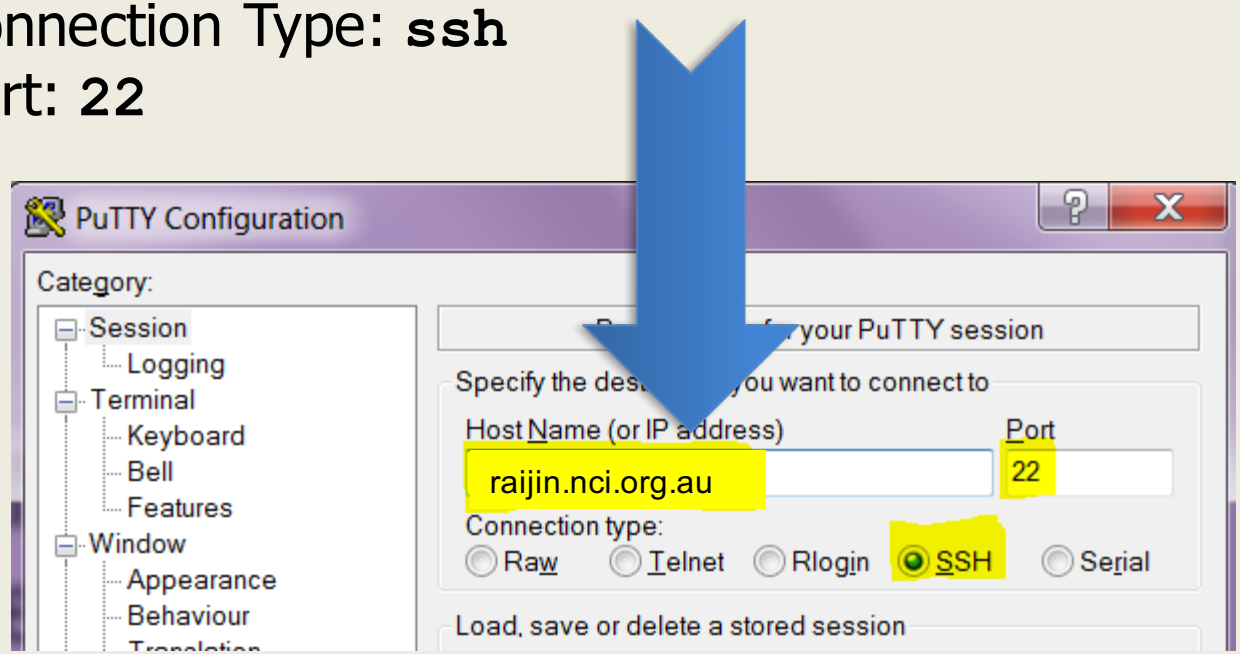
Install software

If you use Windows: use your web browser to download Putty and PuttySCP from:
http://www.putty.org

Download Filezilla from the web for Windows and Mac.
https://filezilla-project.org

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

- If you use a Mac use the terminal program:
- ssh iaa444@raijin.nci.org.au

- If you use Windows use putty. into Raijin
  - **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

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$$
```

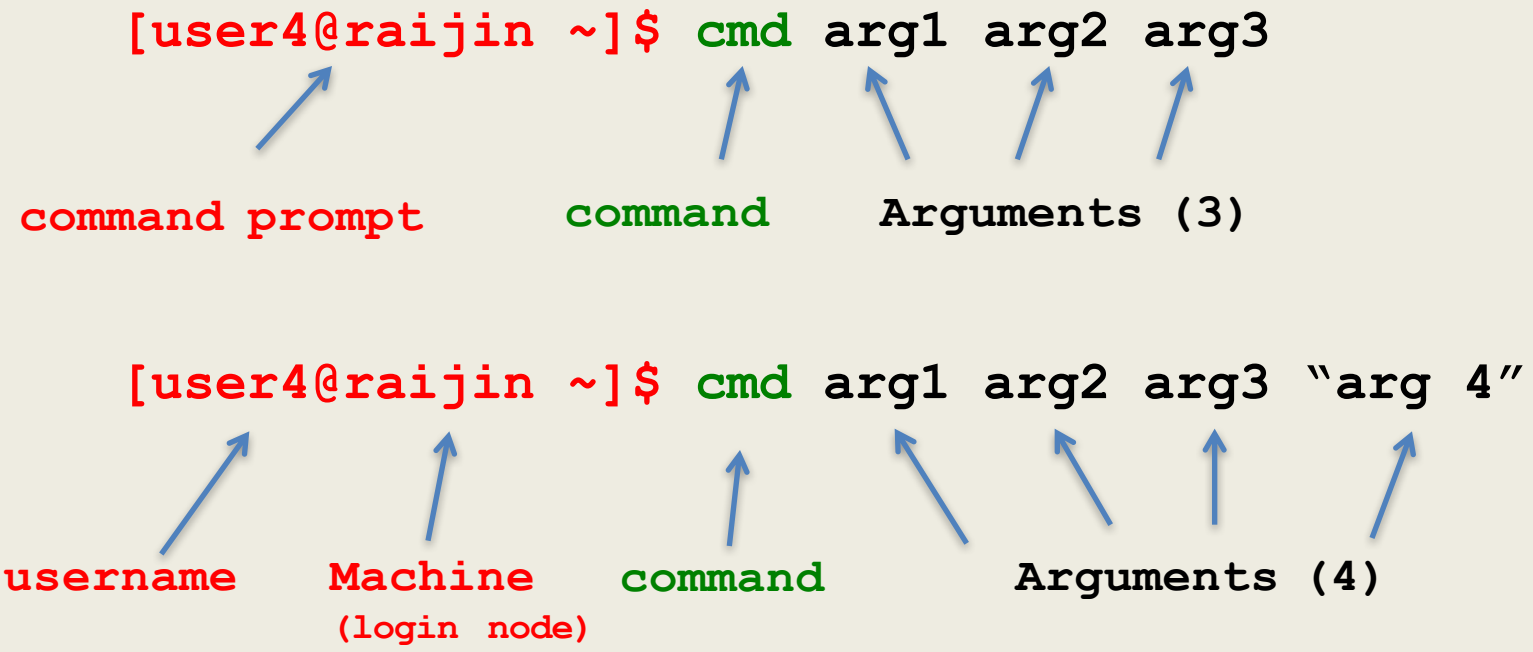

INTERSECT

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)

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)

Try these 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

INTERSECT

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 .

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.
  - 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

Exercise 1(b)

Try these 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
```

```
To quit a man command, press q.
```

Exercise 1(c)

Print a calendar from Feb-Apr 1964.

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

Unit 2: Where am I?

Goals:

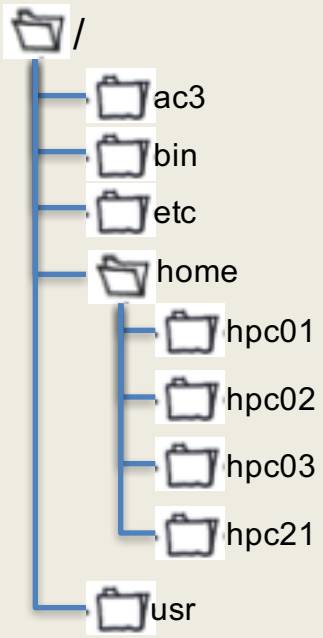
- Can explore navigate a hierarchy of directories
- Can specify files with relative and absolute paths
- Can create a hierarchy of directories
- Can copy and move files between directories

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

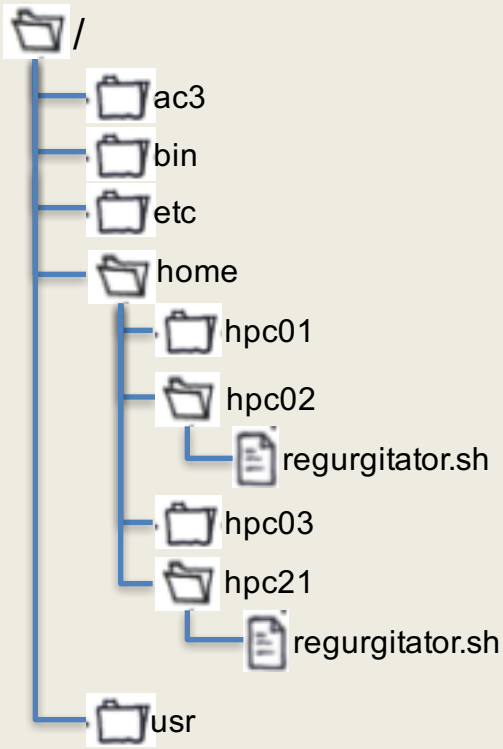
What are the names of the contents of the directory called "`/ (root folder)`"?

```
/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`

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"

Rules So Far

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

Change dir and list content

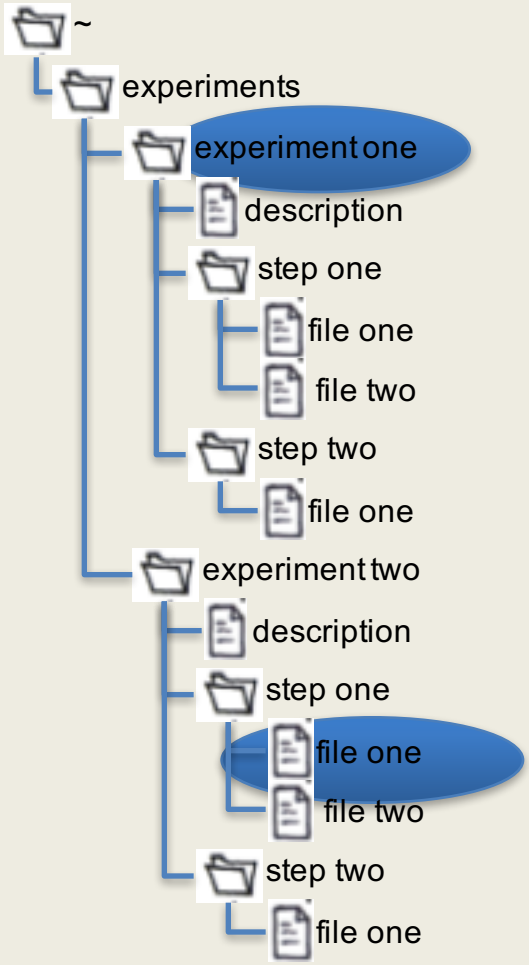
Command	Description
`cd`	Change the working directory to your home directory
`cd` *`<path>`*	Change directory to *`<path>`*
`ls`	List the contents of the working directory
`ls` *`<path>`*	List the contents of *`<path>`*

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

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?

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"

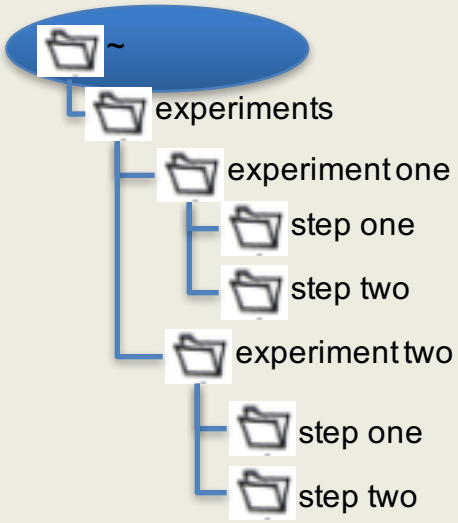
Finding your way around

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>
```



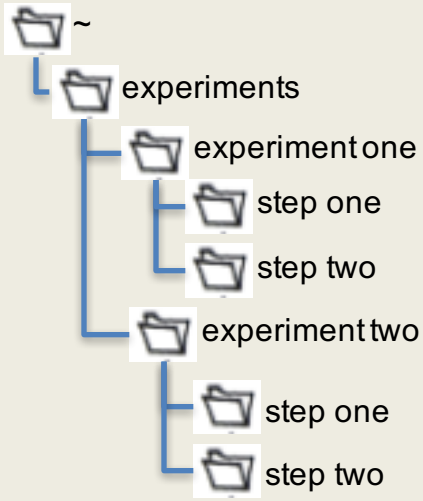
What happens when you execute the commands?

```
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(a)

Create the tree

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>`*

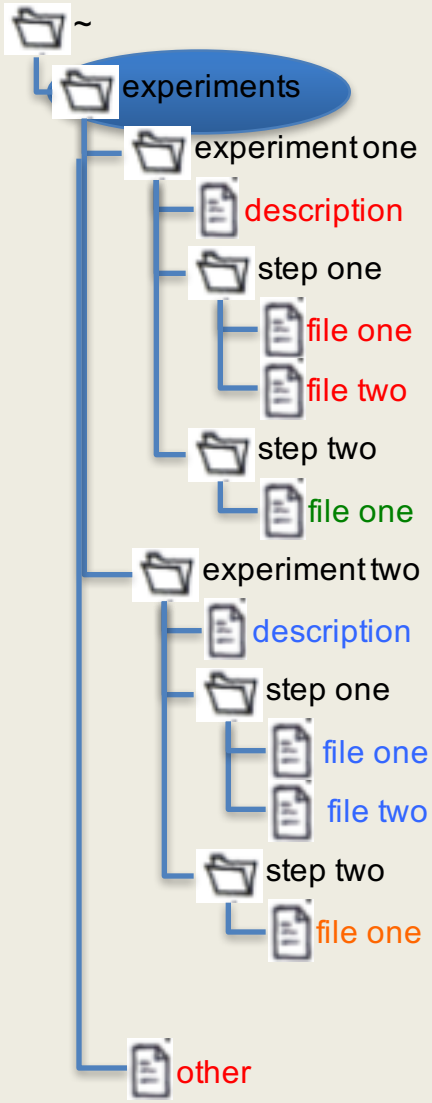


INTERSECT

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(b)

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

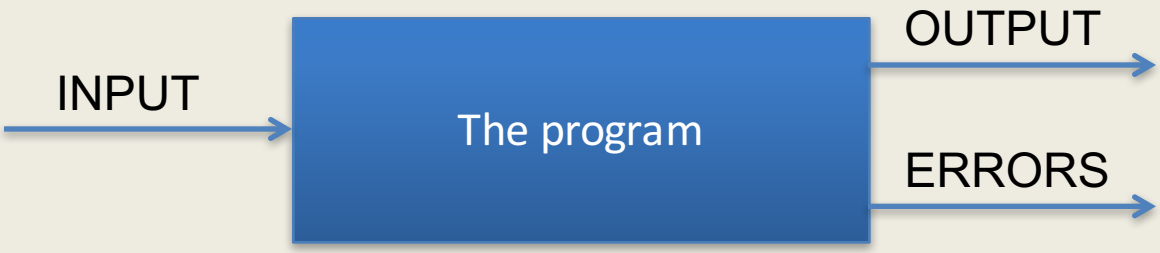
Looking into files

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



INPUT

The program

OUTPUT

ERRORS

command

Redirect OUTPUT elsewhere...

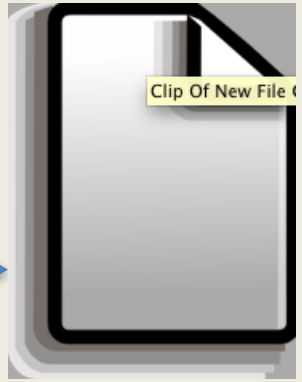
>

INPUT

The program

OUTPUT

ERRORS



```
command > filename
```


INTERSECT

Append OUTPUT elsewhere...

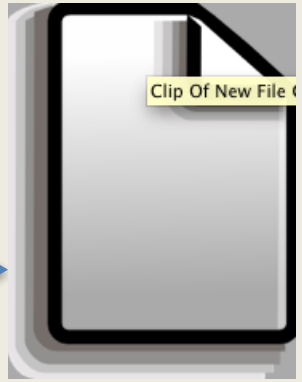
>>

INPUT

The program

OUTPUT

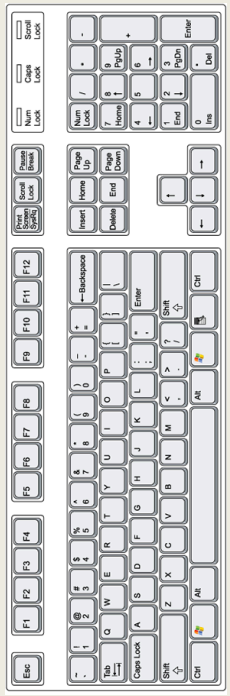
ERRORS

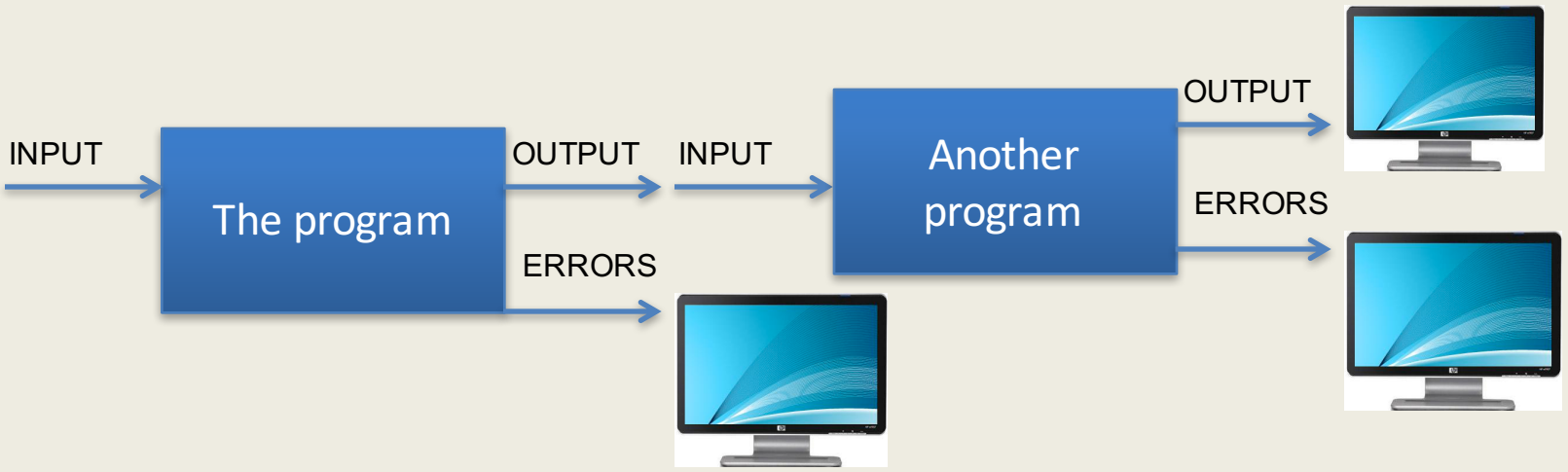


```
command >> filename
```


INTERSECT

Pipe OUTPUT to a command...






```
command | another_command
```


INTERSECT

Exercise 4(a)

Create a file and try the commands below

Command	Description
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>*

INTERSECT

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

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

Manipulating your 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

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)
-