An Introduction to Shell Scripting

Paul Brown

SBIDER & Bioinformatics Research Technology Platform

p.e.brown@warwick.ac.uk

What is the shell?

- A command line user interface for Unix-like operating systems.
- Interactive and scripting modes

What is the Bash Shell?

- Bourne Again SHell, replacing the older Bourne shell in 1989
- Default shell on most Linux systems and MacOS
- Now available on Windows

  https://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/

When to use the shell

- As a wrapper for a workflow
- When doing lots of filesystem access
- When low level access to hardware is required

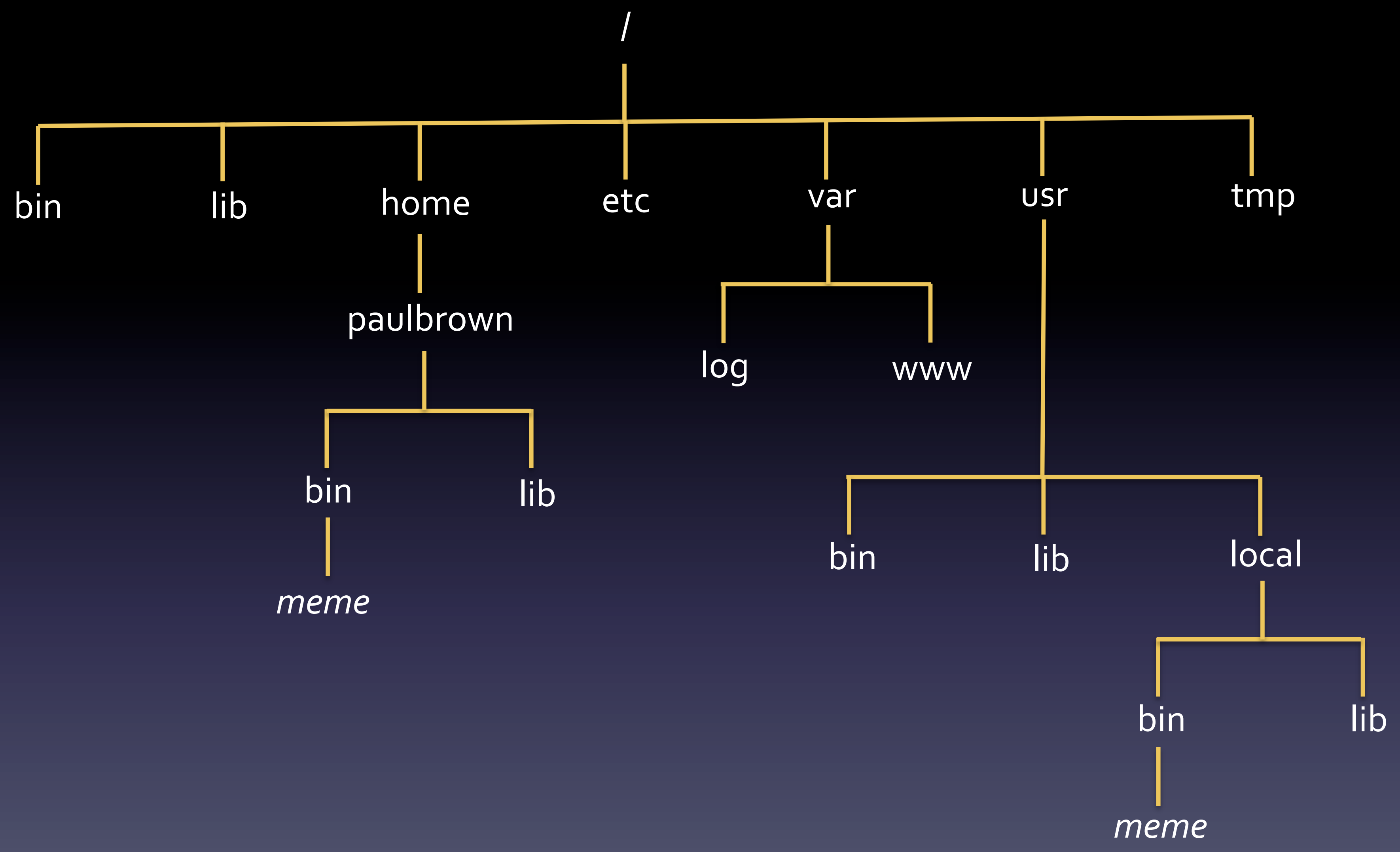
When not to use the shell

Shell scripting is of much less use when any of the following are required

- Complex calculations
- A graphical user interface
- Any kind of debugging beyond very basic

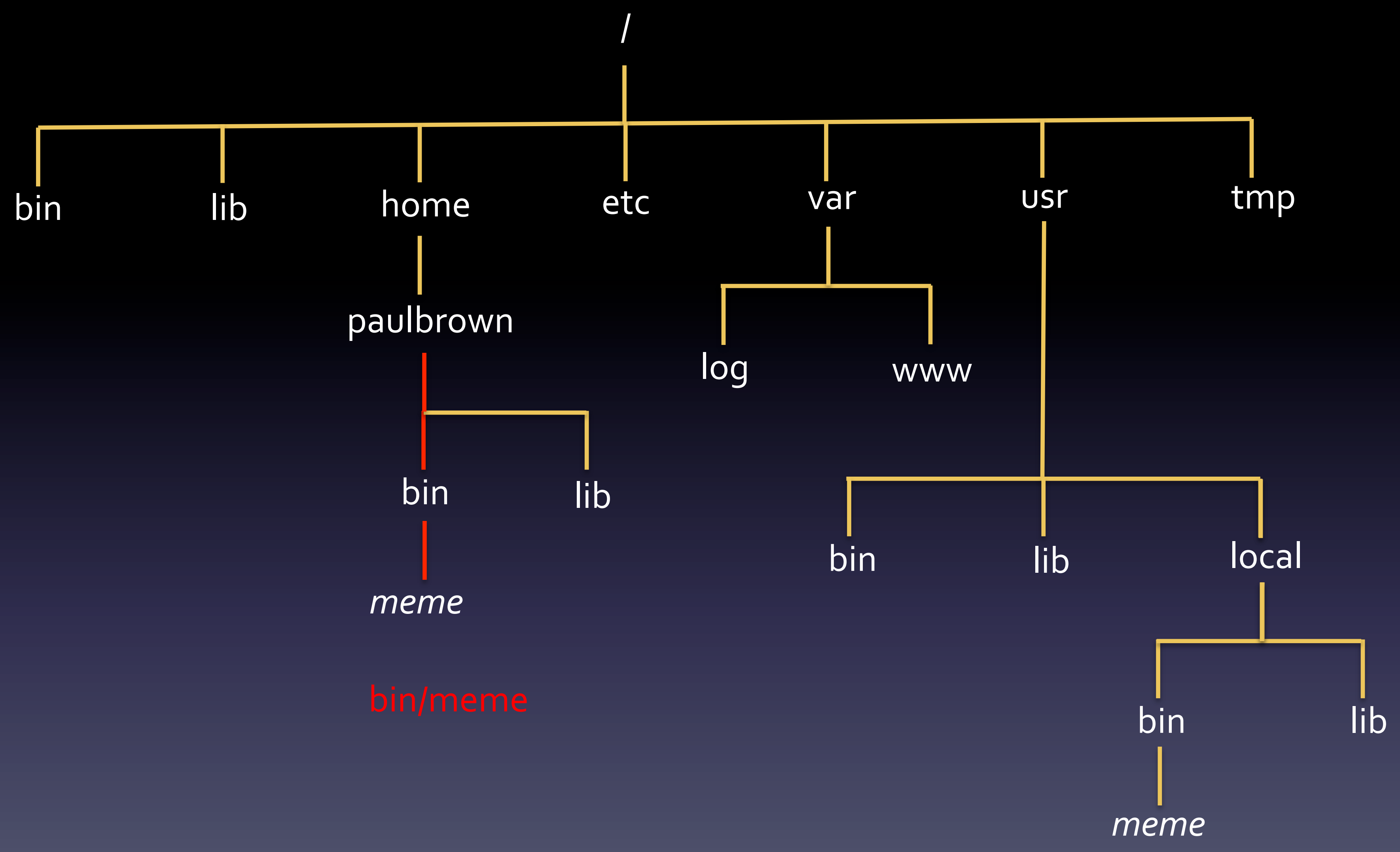
Starting up

- Often opened via the graphical desktop
- Startup files are read to provide user customisations, eg .bash_profile, .bashrc

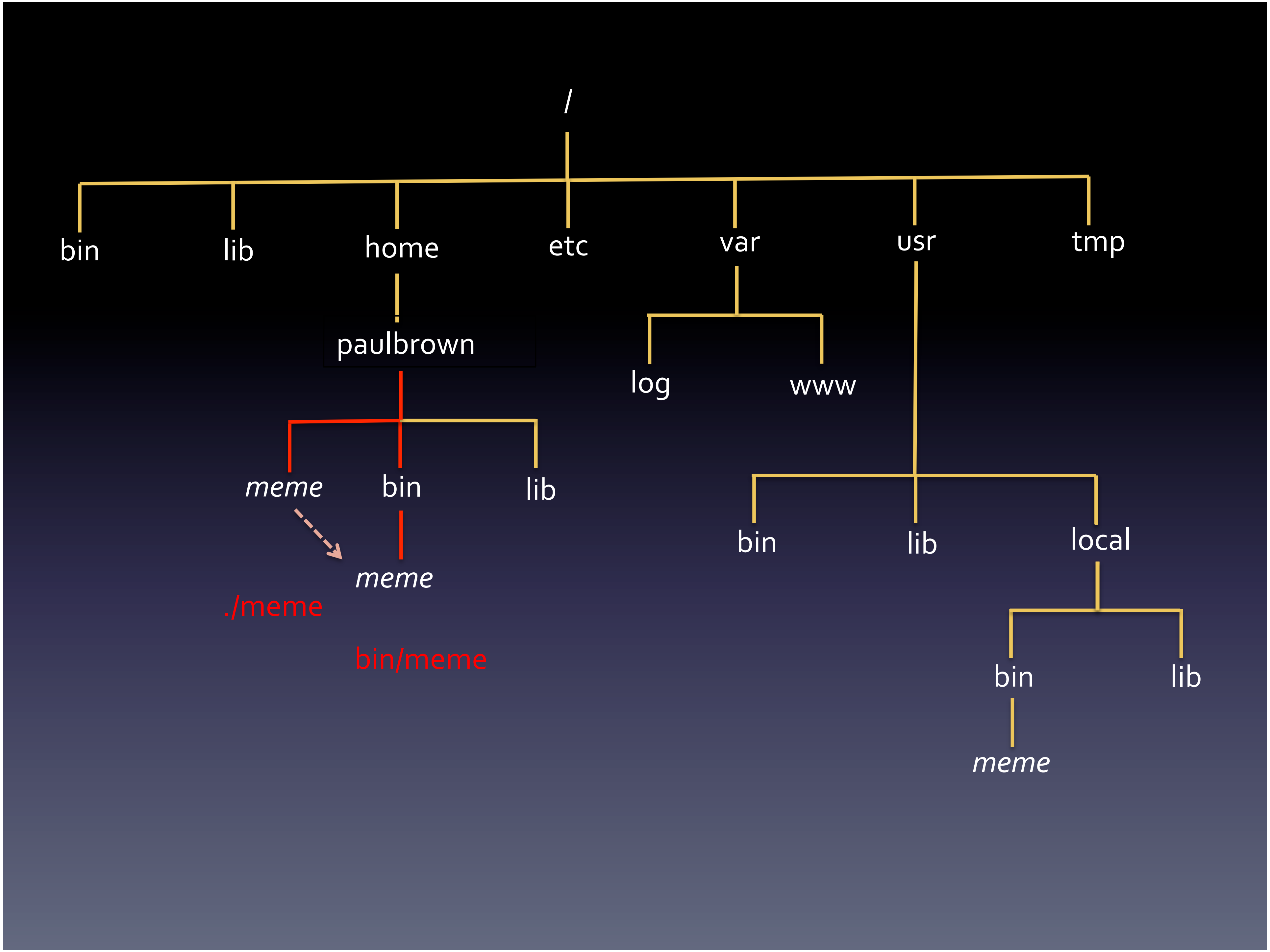

/
bin
lib
home
etc
var
usr
tmp
paulbrown
log
www
bin
lib
meme
bin
lib
local
bin
lib
meme

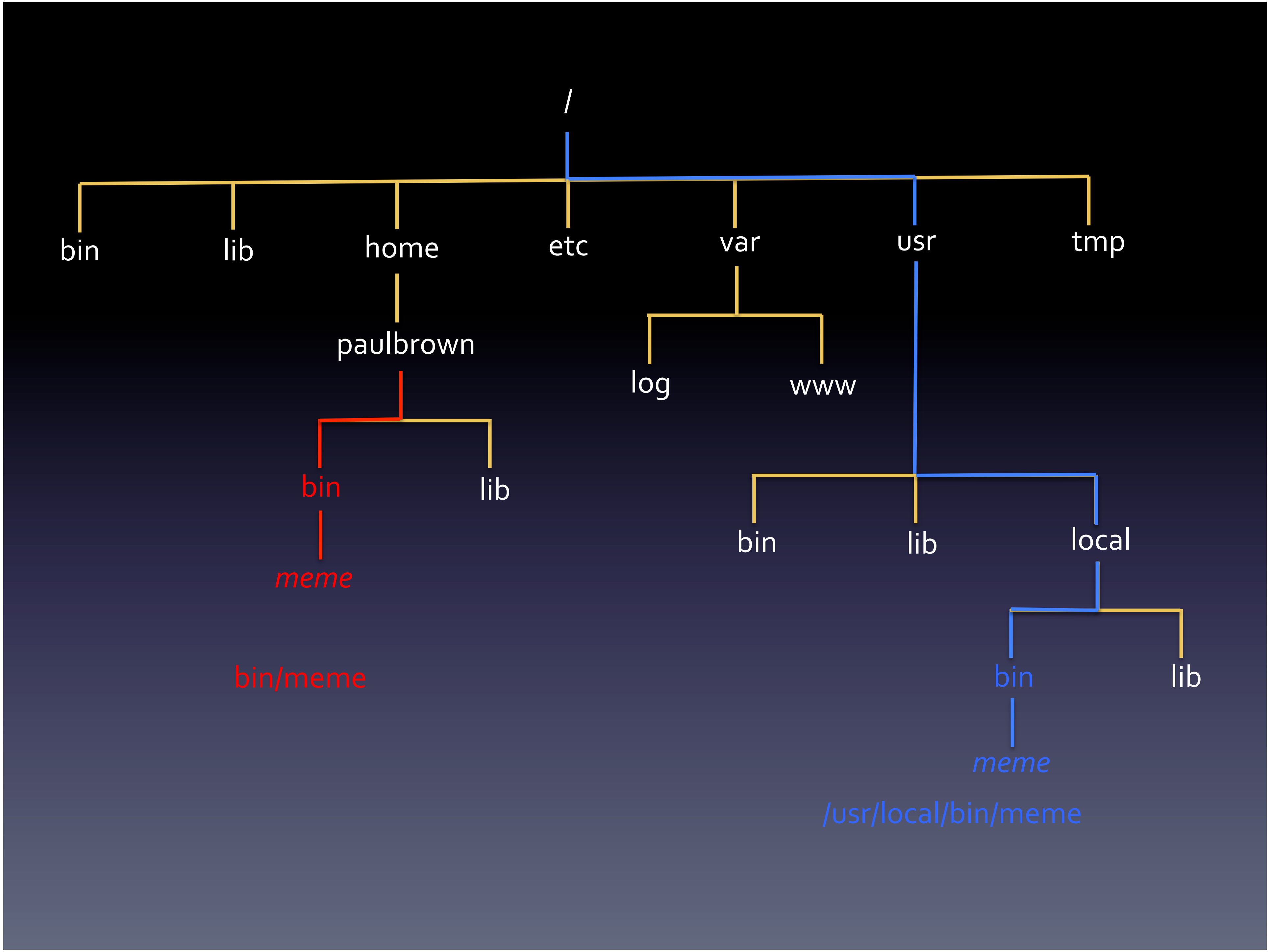
Some useful commands

```
nero:var paulbrown$ cd $HOME
nero:~ paulbrown$ pwd
/home/paulbrown
nero:~ paulbrown$ ls -l
total 104240
drwxr-xr-x    2 paulbrown  staff  64 19 Jun  2017 Anaconda
drwxr-xr-x   29 paulbrown  staff  928 30 Nov  2017 Android
drwx------+  93 paulbrown  staff  2976  1 Nov 12:34 Documents
-rw-r--r--    1 paulbrown  staff  8 22 Feb  2018 README.md
-rw-r--r--    1 paulbrown  staff  0 14 Jul  2015 mcmc.csv
lrwxr-xr-x    1 paulbrown  staff  25 20 Sep  2016 meme ->
/home/paulbrown/bin/meme
nero:~ paulbrown$ mv mcmc.csv mcmc_101121.csv
nero:~ paulbrown$ cp -r Android Android.backup
nero:~ paulbrown$ rm –r Android
```


/
bin
lib
home
etc
var
usr
tmp
paulbrown
log
www
bin
lib
meme
bin
lib
local
bin
lib
meme

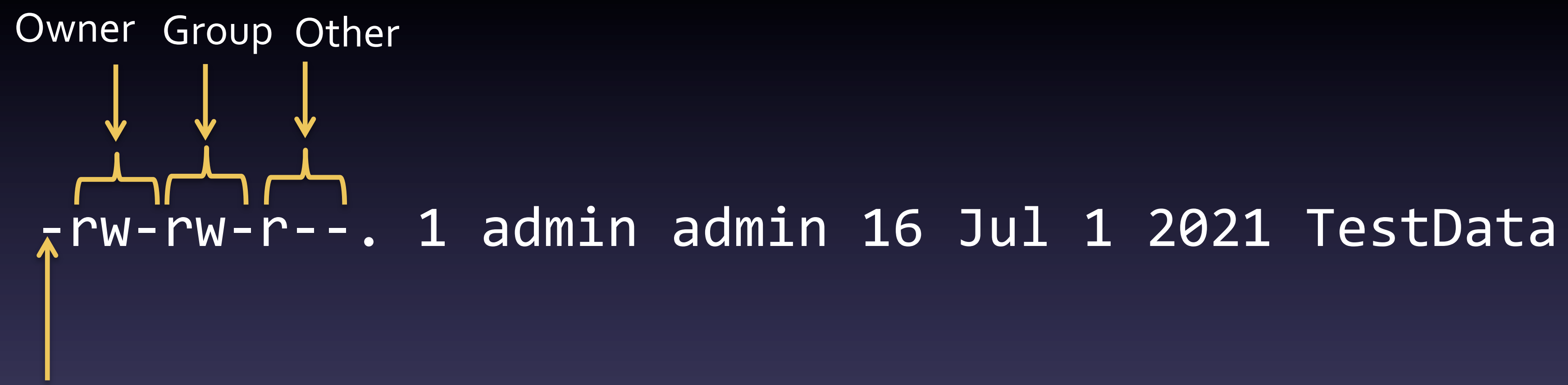
bin/meme


/
bin
lib
home
etc
var
usr
tmp
paulbrown
log
www
meme
bin
lib
meme
./meme
bin/meme
bin
lib
local
bin
lib
meme


/
bin
lib
home
etc
var
usr
tmp
paulbrown
log
www
bin
lib
meme
bin
lib
local
bin
lib
meme
bin/meme
/usr/local/bin/meme

Changing file and directory permissions

r – read, w – write, x - execute



```
-rw-rw-r--. 1 admin admin 16 Jul 1 2021 TestData
```

File type
_ regular file
d directory
l symbolic link

	file	**directory**	**value**
read	Read contents	Display content with ls	4
write	Alter contents	Add/delete file/sub-folder	2
execute	Run as a program	cd into directory/access content with ls	1

Changing file and directory permissions

- Change these with chmod
- This command takes the –R parameter when operating on directory <u>content</u>

Changing file and directory permissions

- Permissions represented by a 3 digit octal number
- Common values include

777	rwxrwxrwx
755	rwxr-xr-x
700	rwx------
644	rw-r--r--
600	rw-------

Changing file and directory permissions

- Permissions can be set individually as well

```
chmod [ugo][+-][rwx] filename
```

- Examples

```
chmod u+x script.sh

chmod go+rx bindir

chmod –R go+r bindir
```

Startup files

- .bash_profile, .bashrc
- Need to use ‘ls -a’ to see these files
- Put customisations in .bashrc as this is run by .bash_profile
- Put them after the comment line to avoid being overwritten by system-wide files

Aliases

- Shortcuts to common commands
- A way of enforcing a specific way of running a command

```
alias matlab=“matlab –nodesktop -nodisplay”
alias trimmomatic="java -jar /usr/local/bin/trimmomatic-0.39.jar"
```

Environment variables

- Variable can be set or appended to
- You can also create new variables
- Conventionally, always use block capitals
- Remember $ when referring to existing value

Environment variables

- Use the export command

```
export PATH=$PATH:$HOME/bin

export PATH=$HOME/bin:$PATH
```

Environment Variables

```
Nero:~paulbrown$ echo $PATH
/bin:/usr/bin:/usr/sbin:/sbin:/usr/local/bin:~/
bin
Nero:~paulbrown$ meme
-bash: meme: command not found
Nero:~paulbrown$ export PATH=$PATH:/usr/local/
meme/bin
Nero:~paulbrown$ echo $PATH
/bin:/usr/bin:/usr/sbin:/sbin:/usr/local/bin:~/
bin:/usr/local/meme/bin
Nero:~paulbrown$ meme
 USAGE:
      meme  <dataset> [optional arguments]
```

More on variables

- There are no variable types
- VARNAME is a reference
- $VARNAME is the value held there

```
Nero:~paulbrown$ echo PATH
PATH
```

Using quotation marks

- Important to know the difference between single and double quotes
- Expressions are evaluated inside "…", but not inside '…'

```
paul-browns-macbook:~ paulbrown$ NAME="Paul"
paul-browns-macbook:~ paulbrown$ echo "Hello $NAME"
Hello Paul
paul-browns-macbook:~ paulbrown$ echo 'Hello $NAME'
Hello $NAME
```

Working with files and directories

- Many useful commands
- Use `cat` to quickly view the contents of small files
- Use `less` to scroll though a larger file

  `-N` option to display line numbers
- Use `head/tail` to display the start/end of a file

  `-n` option specifies the number of lines

Searching files and directories using `grep`

`grep` stands for Global Regular Expression Print

Can be used to

- Test for the presence/absence of a word or pattern
- Names of files containing a word or pattern

Searching files and directories using `grep`

Useful options are

- -i ignore case
- -v invert (print non-matching lines)
- -c display only count of lines that match
- -A *number* and –B *number* Print number of lines before and after matching line
- -r recursively search all files in a directory
- -l display names of files with a match

Searching files and directories using `grep`

- The simplest use is to print matching lines within a file

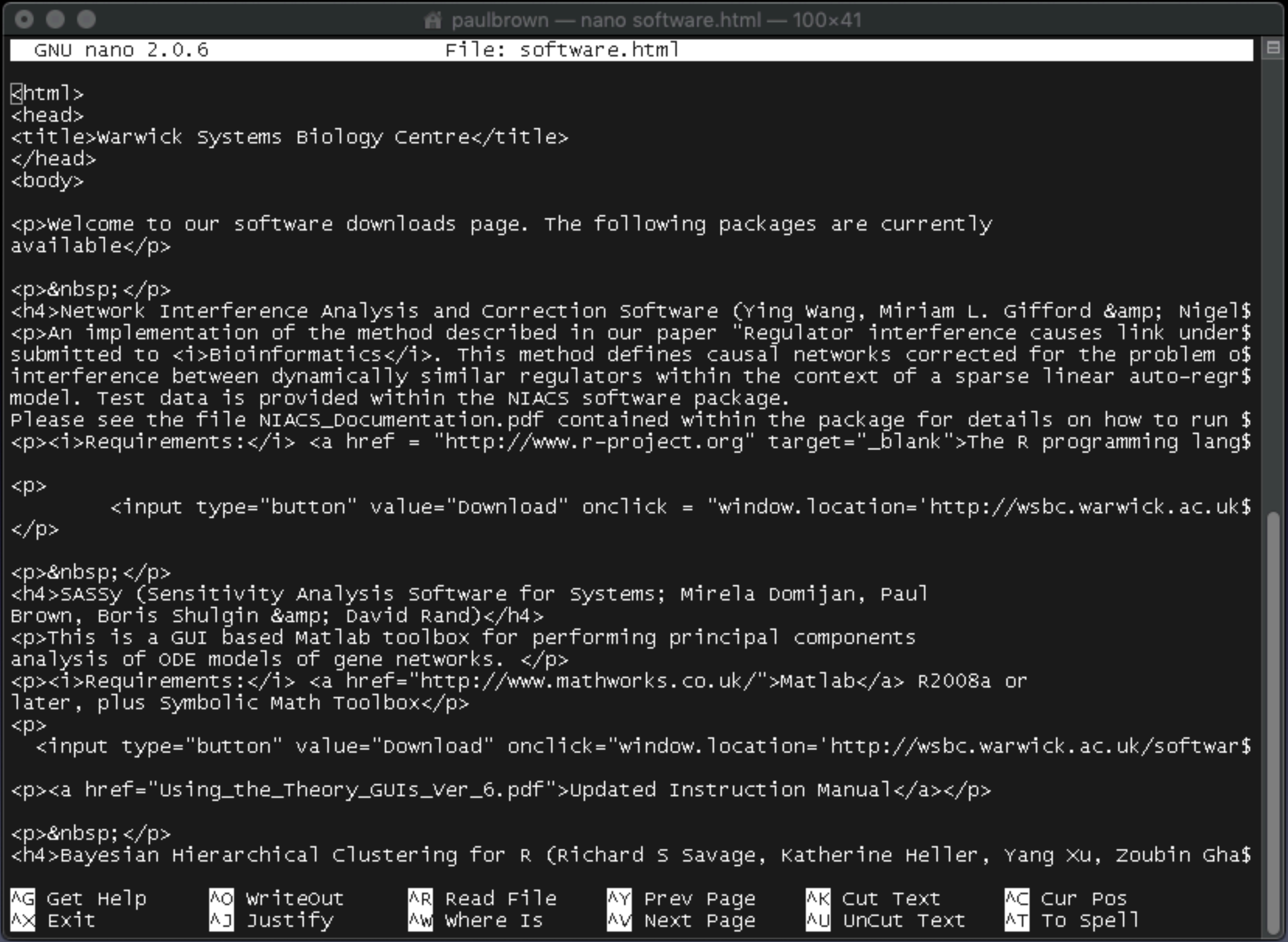
```
grep –B 2 –A 2 ‘paulbrown’ /var/log/secure
```

- Or find files that contain a match

```
grep –r –l “paulbrown” /var/log
```

Writing files

- A number of interactive text editors, eg vi, nano
- Also use re-direction >, >>
- echo “some content” >> script.sh



```
  GNU nano 2.0.6                    File: software.html

<html>
<head>
<title>Warwick Systems Biology Centre</title>
</head>
<body>

<p>Welcome to our software downloads page. The following packages are currently
available</p>

<p>&nbsp;</p>
<h4>Network Interference Analysis and Correction Software (Ying Wang, Miriam L. Gifford &amp; Nigel$
<p>An implementation of the method described in our paper "Regulator interference causes link under$
submitted to <i>Bioinformatics</i>. This method defines causal networks corrected for the problem o$
interference between dynamically similar regulators within the context of a sparse linear auto-regr$
model. Test data is provided within the NIACS software package.
Please see the file NIACS_Documentation.pdf contained within the package for details on how to run $
<p><i>Requirements:</i> <a href = "http://www.r-project.org" target="_blank">The R programming lang$

<p>
        <input type="button" value="Download" onclick = "window.location='http://wsbc.warwick.ac.uk$
</p>

<p>&nbsp;</p>
<h4>SASSy (Sensitivity Analysis Software for Systems; Mirela Domijan, Paul
Brown, Boris Shulgin &amp; David Rand)</h4>
<p>This is a GUI based Matlab toolbox for performing principal components
analysis of ODE models of gene networks. </p>
<p><i>Requirements:</i> <a href="http://www.mathworks.co.uk/">Matlab</a> R2008a or
later, plus Symbolic Math Toolbox</p>
<p>
  <input type="button" value="Download" onclick="window.location='http://wsbc.warwick.ac.uk/softwar$

<p><a href="Using_the_Theory_GUIs_Ver_6.pdf">Updated Instruction Manual</a></p>

<p>&nbsp;</p>
<h4>Bayesian Hierarchical Clustering for R (Richard S Savage, Katherine Heller, Yang Xu, Zoubin Gha$

^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

Redirection

- Input to and output from command can be re-directed away from stdin and stdout
- Re-direct output to file

```
ls -l > dircontent.txt
```

Re-direct input from file

```
sort -k5 -n < dircontent.txt
```

Redirection

Pipes are used to chain commands together so the output of one becomes the input of the next

```
ls -l | sort -k5 -n

tail -n 1000 logfile.log | sort | more
```

Command substitution

- This allows the output of a command to be captured and used piped back to be used as an argument for something else, or to be captured in a variable
- Preferred way is to use `$(…)`

```
rm –f $(find . –name “*.txt”)
```

Arithmetic expansion

Use command substitution

```
paul-browns-macbook:~ paulbrown$ echo 2+3
2+3
paul-browns-macbook:~ paulbrown$ echo $(2+3)
-bash: 2+3: command not found
paul-browns-macbook:~ paulbrown$ echo $((2+3))
5
paul-browns-macbook:~ paulbrown$ a=$((2+3))
paul-browns-macbook:~ paulbrown$ echo $a
5
```

Arithmetic expansion

Bash handles only integer types

```
paul-browns-macbook:~ paulbrown$ echo $((4/3))
1
```

Use ‘bc’ to perform calculations with floating point types

```
paul-browns-macbook:~ paulbrown$ echo 'scale=3;4/3’ | bc
1.333
```

Remote Shells

- rsh (remote shell). Do not use, insecure
- ssh (secure shell, port 22)

```
paul-browns-macbook:~ paulbrown$ ssh nero.wsbc.warwick.ac.uk
paulbrown@nero.wsbc.warwick.ac.uk's password:
Last login: Mon Nov  4 23:10:34 2019 from 95.149.133.253
-sh-4.1$ hostname
nero.wsbc.warwick.ac.uk
```

- Also sftp and scp

```
scp -r /local/stuff paulbrown@nero.wsbc.warwick.ac.uk:$HOME
```

Shell scripting

- Conventionally, files have .sh extension
- Remember to set execute permission
- Script begins with

```
#!/bin/bash
```

Input arguments

- Referred to as `$1`, `$2` etc..
- `$#` is the number of inputs
- Same applies to functions
- Use `read` to request user input

Conditionals

- Surround an expression with [[…]]
- String operators : -z, -n, ==, !=, <, >, =~
- Numerical operators: -eq, -ne, -lt, -le, -gt, -ge
- File operators: -e, -f, -d, -r, -w, -x

Conditionals

```
#!/bin/bash

if [[ $# -lt 3 ]] ; then
      echo “Not enough input arguments”
      exit 0
elif [[ $# -gt 5 ]] ; then
      echo “Too many input arguments”
      exit 0
else
      echo “OK”
fi
```

Conditionals

- Can be chained together using logical operators &&, ||

```
#!/bin/bash

if [[ $# -lt 3 ]] || [[ $# -gt 5 ]]; then
        echo “Wrong number of input arguments”
        exit 0
else
        echo “OK”
fi
```

- These operators allow conditional execution

```
mkdir newdir || echo “Cannot create directory”
mkdir newdir && touch newdir/newfile
```

While Loops

```
while read  line; do
    fields=(${line}) #expand to array
    …
done < infile
```

`break` and `continue` can be used within the loop body

For loops

A for loop iterates a series of words in a string

```
for i in $(ls); do
        echo $i
done
```

A C-style for loop can be created using arithmetic expressions

```
for ((i = 0; i < 100; i++)); do
        echo $i
done
```

Range expression

```
for i in {1..10}; do
        echo $i
done
```

Functions

```
myFunc() {
      local localVar=“Hello ”$1;
      echo localVar;
}
myFunc “Paul”
```

Return values can be captured by command substitution

Text Processing

- Bash as a comprehensive set of time-saving text processing functions.
- Be aware of these before attempting to write your own

sort

- Sorts lines of a file
- Alphabetic or numeric sort
- Can sort on columns eg csv files
- Can randomise rows as well

```
sort -n -k2 data.txt -o data.txt
```

uniq

- Remove duplicates in a list
- Use after sort as it compares only adjacent values

```
sort list.txt | uniq > unique_list.txt
```

comm

- Compare 2 files and print lines unique to each and common to both, in 3 columns
- Can suppress any of these outputs with -1, -2, -3

```
comm -12 file1 file2
```

tr

- Translates or deletes characters

```
tr a-z A-Z < lowercase.txt > upper_case.txt

tr –d _ <input.txt
```

cut

- Removes section from each line of a file with well defined columns
- Default delimiter is TAB, but can be changed

```
 cat data.txt | cut  –f1,5
```

paste

- Merges lines of files into columns with the specified delimiter

```
paste -d, names.txt values.txt > scores.csv
```

Getting help

- man pages for most commands
- Huge amount on online resources, eg a good cheat sheet at https://devhints.io/bash