

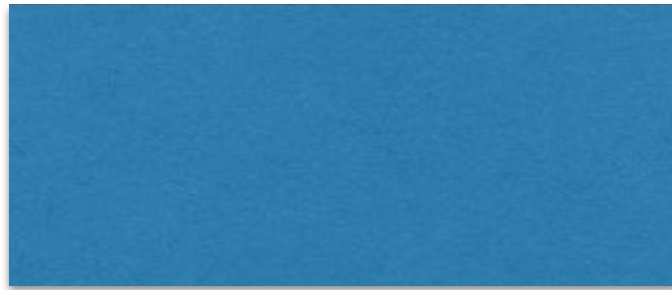
# Why do CS people drone on about Unix?

Dr Ian Batten

`I.G.Batten@bham.ac.uk`

# DPD Pickup Pass For:

Name: BATTEN IAN



Reference: 15505359292961

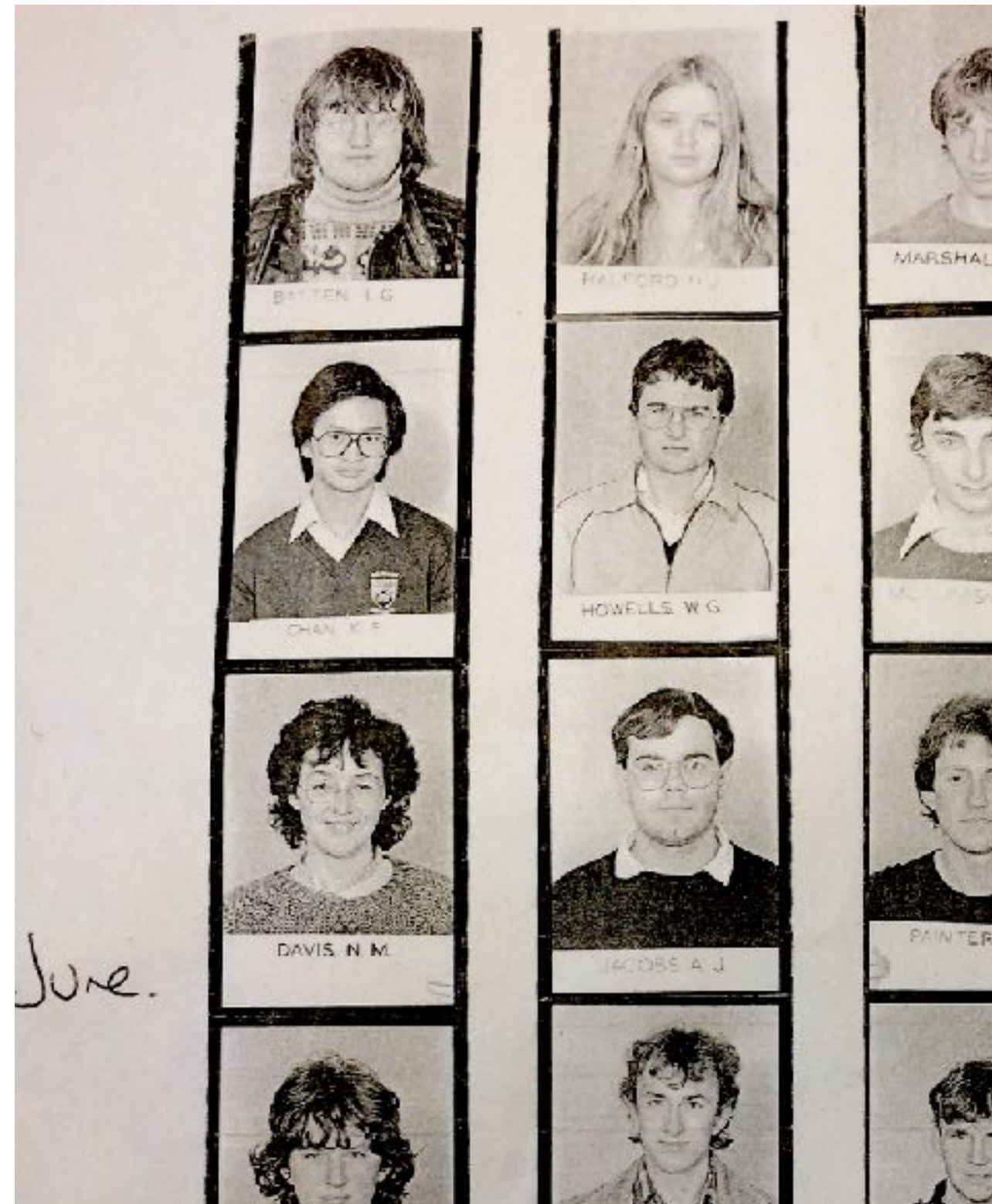
Number of parcels to collect: 1



Your parcel can be collected from today until the **undefined NaN undefined**.

# Welcome to CS

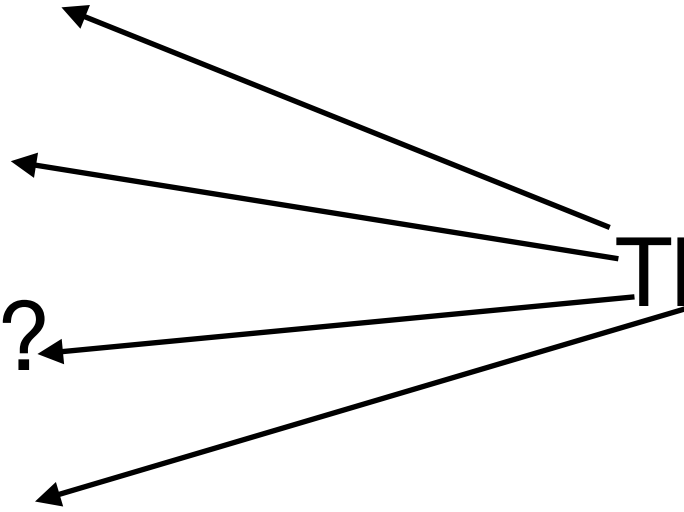
- I'm a new lecturer; after 20-odd years in industry I came back to do a PhD in my late forties and stayed as a lecturer.
- I worked here in the 1980s after doing my degree and some of the computing decisions I made have lingered
  - *The evil that men do lives after them; the good is oft interred with their bones.*
- I hope you will enjoy [cs.bham.ac.uk](http://cs.bham.ac.uk) as much as I have.



# Computing in CS

- Most CS departments are Unix shops: servers, desktops, even laptops are running various incarnations of the Unix operating system
- Windows is notable by its absence
- There is a lot of OSX, sorry, MacOS around, however.

# How many of you have used...?

- Computers at all (don't laugh: until recently there would usually be a few)?
  - Windows machines?
  - Something other than Windows?
    - OSX?
    - iOS?
    - Android?
    - Linux?
- These are all Unix-derived
- 

# So why Unix?

- Long, ancient history, far longer than Windows
- Arose from legacy of **Multics**, started in the mid-1960s and running through until the turn of the century
  - Ran on expensive, complex hardware
  - Tried to do everything
- **Unix** (originally Unics) started in about 1970 as attempt to do one thing better, running on more readily available hardware
- Always intended for programmers and researchers: early release called PWB, for Programmers' Workbench.

# So why Unix?

- Offered a good environment for programming, both in terms of the services and in terms of the tools
- Readily available on a range of hardware, with source code so you could study and change it if you were a university
  - Product of the strange legal position of AT&T/Bell in the 1970s.
- Became *de facto* standard, in various forms, for both CS timeshare machines and (later) “workstations” of various sorts from Sun, SGI, HP, DEC and the rest.

# Unix philosophy

- Small tools that do one thing well
- Easy ways to string those tools together to do larger, more complex jobs (mostly “pipelines”, one command reading the output of another)
- Minimal, general interfaces
- Files as streams of bytes



# Other Approaches

- If you are editing a document “in” (and the preposition is telling) Microsoft Word, you can do nothing with it that Word cannot itself do.
- If you would like to write a program to analyse or change your document, you cannot, because the format of the file in which Word stores the document is private to Word, and very complex.
- Unix offers small, flexible tools you can use on anything; Windows offers large, special purpose tools that do one job but offer little extensibility.
  - Yes there are scripting languages in some tools: the general point remains.

# Structure of Unix

- A *kernel* which looks after the hardware and provides a set of services to access the hardware, protect programs from each other, allow programs to communicate, and do things common to all programs
- A set of *libraries* that make writing programs easier (why write your own square root routine, or your own routine to sort things into order?)
- A set of *utilities* which are small programs to do useful jobs.

# Unix v Linux

- Up until the mid-1980s, either you had a source license (required money and/or being academic) and could tinker to your heart's content, or you didn't, and had to take what a reseller gave you.
- “Commercial” Unix was often not as good as “academic” Unix.
  - And you couldn't change it
  - And if it broke, you couldn't fix it.

# Along comes RMS

- Richard Stallman is not a Unix guy: he comes from another strand of operating system development at MIT.
- He is the original author of Emacs, a favourite text editor amongst developers
- He is a fervent believer in free (as in “free speech”, French “libre”, rather than “gratuit”) software. Everyone should have source code, so they can modify and develop it further.
- Unix wasn't his favourite, but was not too objectionable.
- He set about writing the whole operating system, a Unix clone with enhancements, starting in September 1983. He called it GNU (“Gnu's not Unix”), cf. Eine and Zwei.

# GNU

- The GNU Project produced an implementation of Emacs, a very good C compiler, clones of the main Unix tools and a suite of libraries that replicated most of the Unix functionality. Collectively, this is all called a “userland”. By the end of the 80s, they had almost the complete set.
- They did not, however, produce a working kernel. They became side-tracked into a very complex, very high-tech, very modern project (“HURD: the hurd of Unix replacement daemons”) which used a lot of then-fashionable techniques. It stalled, and is still stalled more than 25 years later.

# GNU

- So you ended up running large parts of the GNU Userland (which was more modern and better suited to modern hardware, as well as having some nice extensions) on top of the commercial kernels (usually Sun Microsystems' "SunOS") or the more academic "BSD".
- And some of the commercial Unixes actually started to use the GNU tools as part of their offering (gcc, particularly).

# Along comes Linus

- Linus Torvalds was at that point a Finnish computer science undergraduate who wanted to learn about operating systems. He saw that GNU had a complete-ish userland, and wanted to complete the project. He set about writing a Unix kernel for the then-new Intel 80386 processor.
- The kernel he wrote, Linux, was not modern, not high-tech, not fashionable: it used very basic techniques and had a wide range of performance and stability issues. It famously provoked the ire of Andy Tannenbaum, a high-profile researcher, who said Linux would get a failing grade on his course if he'd submitted it. Unix greybeards were similarly dismissive.
- It did, however, work, and it was available (September 1991), so people could get on with improving it. And improve it they did, often with the help of those self-same greybeards.

# So...

- Although there were, and are, commercial Unixes of high quality (notably Sun/Oracle's Solaris, IBM's AIX and Apple's OSX/MacOS), Linux dominates academia and much of industry.
  - Free (unless you want support)
  - Portable (unless you want support)
  - Easy to modify (unless you want support)
  - Runs on everything from Raspberry Pis to IBM Z Series mainframes.



# Linux is used...

- For the server side of most big websites (Google, Facebook, Twitter, Amazon...)
- As Android (which is basically Linux)
- As the basis of many routers and other embedded systems (including my TV)
- RMS insists it should be called GNU/Linux, to reflect that a lot of the userland comes from GNU.
  - RMS is fighting a losing battle on this.

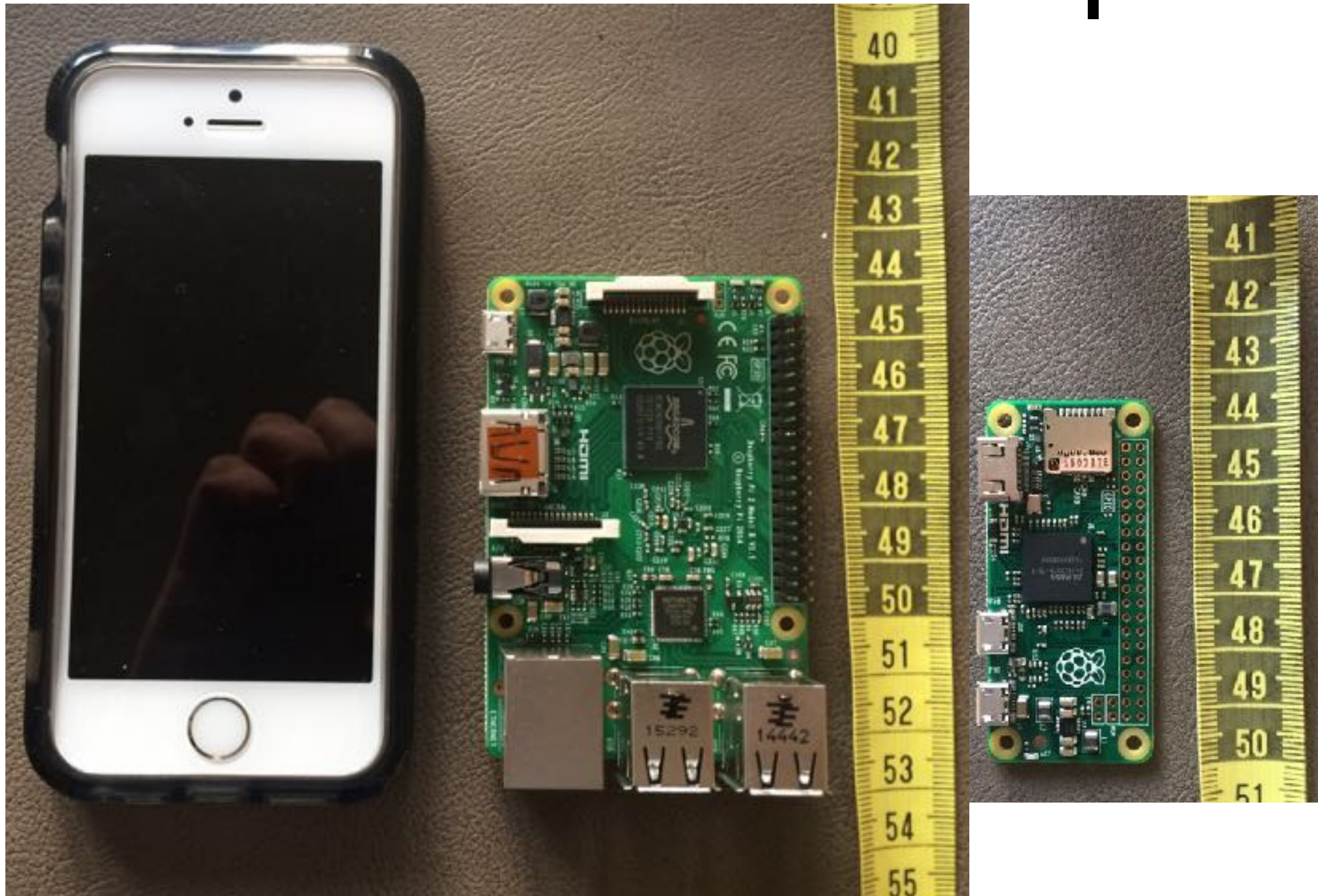
# Linux is used...

- As the obvious “changeable” operating system for research and development in academia
- As the nicest platform for doing software development work
- As the easiest platform to manage at scale
  - Docker, Xen, Containers: big, exciting new technologies come out of either Linux or Solaris.

# Machines are cheap!

- You can install it on your laptop as the main operating system (for the brave), as a dual boot (I don't like this), or using virtualisation (the best solution).
- There are also small, cheap machines available for £25. The ubiquitous Raspberry Pi.

# Small and Cheap



# What CS does badly

- Tell people why Unix is our choice
- Teach much about it as an end in itself
- Explain principles and general ideas, rather than giving you explicit things about individual exercises
- Give you any clue as to why so many of the commands have funny names

# So here's a simple Unix command

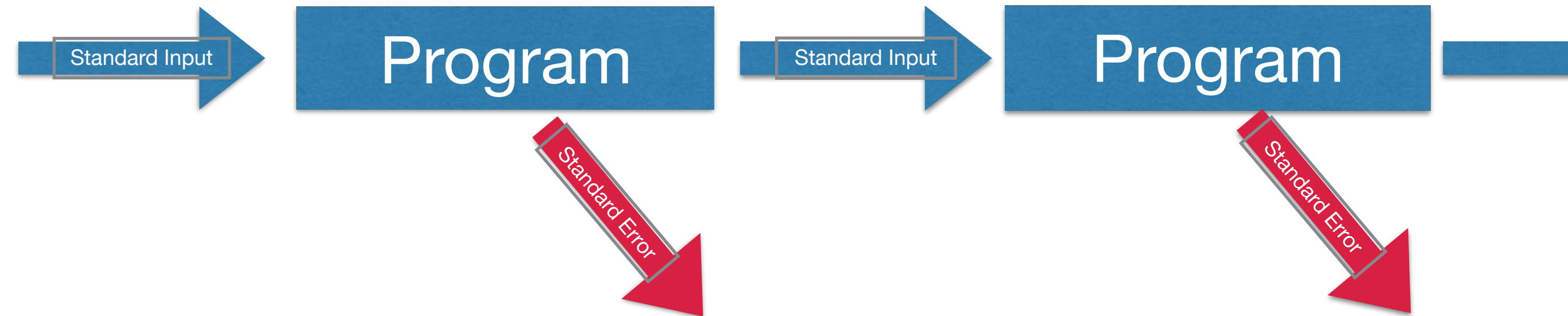
- Create a file, and then count how many lines there are in it

- ```
ians-macbook-air:~ igb$ cat > somefile
this is line one
this is line two
here is another line
ians-macbook-air:~ igb$ wc -l somefile
  3 somefile
ians-macbook-air:~ igb$
```

# stdin/stdout/stderr



# A Pipeline





# Those funny characters

- `>` (greater than) means “send the standard output from this program to this file”
- `<` (less than) means “take standard input from this file”
- `|` (pipe, often shift-backslash but you might need to go hunting) means “send the output of the command to the left to the input of the command to the right”

# Typing commands

- “tab” will try to complete filenames, so far as is possible
- “up-arrow” will bring back the previous command, which you can then edit with the cursor keys.
- If you want to be cool, read up on “C Shell History Mechanism” for another way to do this

# Some useful commands

- **cat**: conCATenate some files and send them to standard output, or failing that copy standard input to standard output
- **grep**: “show me the lines that match this pattern”
  - Stands for “global regular expression print”: regexps are for another day
- **wc**: “word [and lines, and characters] count”
  - Stands for “word count”
- **sort**: show me the input, sorted in various ways
- **ls**: (that’s L, ell): **list** the files in a directory
- **cut**: pick sections of out of lines
- **tr**: transliterate characters (make all “a” into “A”, etc)

# Structure of commands

- Usually [name] [options] [perhaps an operand] [files]
- Options, aka switches, alter the behaviour of the command in some way
- So for example “**grep this file1 file2**” will print all the lines containing “this” in file1 and file2.
- “**grep -v this file1 file2**” will print all the lines that don't contain “this” in file1 and file2. in**V**ert.
- “**grep -i -c -v this file1 file2**” will count the number of lines that don't contain “this”, “This”, “THIS”, etc. Ignore case, **C**ount, in**V**ert.

# Trivial Example

Prompt

What I type in bold

```
ians-macbook-air:testcase igb$ cat > file1
```

```
line containing this
```

```
line not containing the word
```

```
line containing THIS
```

```
ians-macbook-air:testcase igb$ cat > file2
```

```
line with this in it
```

```
some other line
```

Type Control+D to finish input

```
ians-macbook-air:testcase igb$ grep this file1 file2
```

```
file1:line containing this
```

```
file2:line with this in it
```

```
ians-macbook-air:testcase igb$ grep -v this file1 file2
```

```
file1:line not containing the word
```

```
file1:line containing THIS
```

```
file2:some other line
```

```
ians-macbook-air:testcase igb$ grep -i -c -v this file1 file2
```

```
file1:1
```

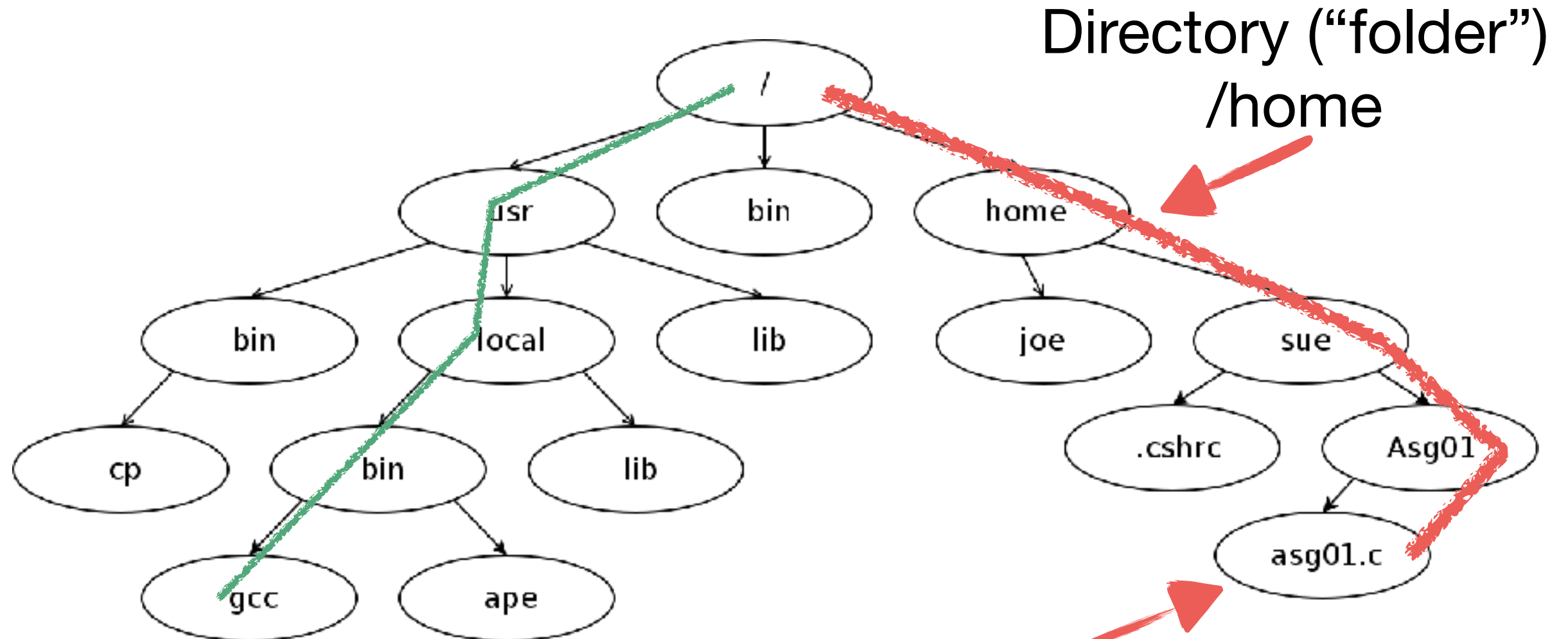
```
file2:1
```

```
ians-macbook-air:testcase igb$
```

# Getting help

- The confusingly named “man” command (which stands for “manual”) will show you the manual pages entry for a particular command.
- The options -h, --help (that’s two dashes) or -? will sometimes, but not always, get you a summary of a command’s options.
- All of this pre-supposes you know which command you want to use, which is where experience and teaching comes in.

# Files



Directory (“folder”)  
/home

File /usr/local/bin/gcc

File /home/sue/Asg01/asg01.c

# Note for Windows users

- On a running Unix machine, there is a single hierarchy of files, always starting at root (/), which is usually, but not always, the disk the machine booted from. Other devices slot into that hierarchy (via “mounting”).
- There is no equivalent of drive letters: any point in the hierarchy can be the root of some device that has been connected to the machine, or a filesystem mounted over the network.
- USB sticks end up in /media, or /Volumes, or /usb, depending on the precise variant you are using.
- There is exactly and precisely one directory called “/”.



# Working with directories

- **cd**: Change to a particular directory and make it the one you are now working in. On login you are in your “home directory” (/home/igb, /Users/igb) and “**cd**” on its own takes you back there.
- **pwd**: print the current directory
- **mkdir**: make a new directory
- **rmdir**: remove a directory
- Vowels are in short supply, you know :-)

# Wild Cards

- \* (star) means “all the files in this directory”
- \*.java (star dot java) means “all the files in this directory whose names end in dot java”
  - By convention, as on most operating systems, the bit after the last dot indicates the type of the file, in this case the source code for Java programs
- 2016\*/\*.mp4 means “all the files whose names end in dot mp4, contained within directories whose names start with 2016, in this directory”.

# Good Filenames

- You can have files and directories with names that contain any character you want other than slash “/” (and, for real completeness one other character, NUL, that you cannot type).
- But if you have files with names containing anything other than letters, numbers, dots, dashes and underscores your Unix life will become more interesting than you need it to be.
  - Most punctuation means something special to something
  - Spaces separate filenames, so `my program.java` will often read as two files, `my` and `program.java`.
- `my_program.java` and `MyProgram.java` are the usual alternatives.

# The Editor Wars

- I suggested yesterday that we needed a laser deterrent system in the building which destroys anyone who starts this discussion.
- Editors are programs which you use, like perhaps Notepad or Textedit, to create files of text (ie, program files).
- They are the subject of intense emotional attachment.

# Emacs, vi and ed

- The two main choices are Emacs (now always GNU Emacs), and vi (now usually in the form of the clone, vim: even MacOS now ships vim in place of vi).
- Both have their fans, who will never agree.
- And for the real hard-core, there is the line editor, ed.
- Pains me to say it, but learning vi probably more useful: always there on any remotely modern machine.
- There are other choices. Ask a young person.



# You can pretend it's 1972

```
ians-macbook-air:~ igb$ ed
i
#include <stdio.h>
int main (void) { printf ("hello world\n"); }
.
2
int main (void) { printf ("hello world\n"); }
s/hello/Hello
int main (void) { printf ("Hello world\n"); }
s/world/World
int main (void) { printf ("Hello World\n"); }
s/void/int argc, char **argv
int main (int argc, char **argv) { printf ("Hello World\n"); }
1,$p
#include <stdio.h>
int main (int argc, char **argv) { printf ("Hello World\n"); }
w
?
w program.c
82
q
ians-macbook-air:~ igb$ cc -o program program.c
ians-macbook-air:~ igb$ ./program
Hello World
ians-macbook-air:~ igb$
```

# A real job

- So I thought I would walk you through a real task I carried out last week using Unix command line tools.
- Bruce Springsteen now sells complete recordings of all his concerts, and as he plays wildly different sets each night, you end up with quite a few of them. You get a directory full of files, one per song.
- He recently played three nights in New Jersey, with very different set lists. I wanted to know a few stats about what was played and how many times. Yes, I could have looked at [setlists.fm](http://setlists.fm).
- Unix lets me do this from the command line.



# Why is this worthwhile?

- The process of breaking a problem down into steps, each simple enough express in one statement or command, and then combining those simple pieces into larger structures, is what programming is.
- Unix blurs the distinction between commands and programming languages, so while typing commands you are actually programming.
- The Unix style is good for programmers, because it mirrors the way programs are written.

# So...

-l: long listing



- Looking at my home music server, we find:

```
root@volumio:~/USB/Bruce Springsteen & The E Street Band# ls -l
total 28
drwxr-xr-x 2  501 staff 4096 Jul 21 23:58 1988-04-23 Los Angeles, CA
drwxr-xr-x 2 root root  4096 Jul 22 00:07 2013-07-11 Rome, IT
drwxr-xr-x 2  501 staff 4096 Jul 22 00:03 2016-04-25 Brooklyn, NY
drwxr-xr-x 2  501 staff 4096 Jul 22 07:49 2016-06-03 Coventry, GB
drwxr-xr-x 2  501 staff 4096 Sep 12 22:02 2016-08-23 East Rutherford, NJ
drwxr-xr-x 2  501 staff 4096 Sep 12 22:01 2016-08-25 East Rutherford, NJ
drwxr-xr-x 2  501 staff 4096 Sep 12 22:03 2016-08-30 East Rutherford, NJ
root@volumio:~/USB/Bruce Springsteen & The E Street Band#
```

For ease of reading, I have made the prompt (“now type something”) just \$, instead of the more common information about username, machine name and current directory. \$ is old school. Note the directory name breaches all the rules!

# What got played in NJ?

\$ ls "2016-08-23 East Rutherford, NJ"

bs160823d1\_01\_New\_York\_City\_Serenade.m4a

bs160823d1\_02\_Wrecking\_Ball.m4a

bs160823d1\_03\_Badlands.m4a

bs160823d1\_04\_Something\_in\_the\_Night.m4a

bs160823d1\_05\_The\_Ties\_That\_Bind.m4a

bs160823d1\_06\_Sherry\_Darling.m4a

bs160823d1\_07\_Spirit\_In\_The\_Night.m4a

bs160823d1\_08\_Santa\_Claus\_Is\_Coming\_To\_Town.m4a

bs160823d1\_09\_Independence\_Day.m4a

bs160823d2\_01\_Hungry\_Heart.m4a

bs160823d2\_02\_Out\_in\_the\_Street.m4a

bs160823d2\_03\_Growin\_Up.m4a

Wrecking Ball: second song on 23rd

\$ ls "2016-08-25 East Rutherford, NJ"

bs160825d1\_01\_New\_York\_City\_Serenade.m4a

bs160825d1\_02\_Prove\_It\_All\_Night.m4a

bs160825d1\_03\_Night.m4a

bs160825d1\_04\_No\_Surrender.m4a

bs160825d1\_05\_Wrecking\_Ball.m4a

bs160825d1\_06\_Sherry\_Darling.m4a

bs160825d1\_07\_Spirit\_In\_The\_Night.m4a

bs160825d1\_08\_My\_City\_of\_Ruins.m4a

bs160825d1\_09\_Waitin\_on\_a\_Sunny\_Day.m4a

bs160825d1\_10\_Darkness\_On\_The\_Edge\_Of\_Town.m4a

bs160825d2\_01\_Lost\_in\_the\_Flood.m4a

bs160825d2\_02\_Hungry\_Heart.m4a

bs160825d2\_03\_Out\_in\_the\_Street.m4a

Wrecking Ball: fifth song  
on 25th

# How many individual songs, including duplicates?

```
$ ls 2016-08*NJ/*.m4a | wc -l  
102  
$
```

“output all the files whose names end in dot m4a contained in directories that start with 2016-08 and end with NJ, one per line, then count the number of lines”

# How many **distinct** songs got played?

```
$ ls 2016-08*NJ/*.m4a | cut -c 47- | head  
New_York_City_Serenade.m4a  
Wrecking_Ball.m4a  
Badlands.m4a  
Something_in_the_Night.m4a  
The_Ties_That_Bind.m4a  
Sherry_Darling.m4a  
Spirit_In_The_Night.m4a  
Santa_Claus_Is_Coming_To_Town.m4a  
Independence_Day.m4a  
Hungry_Heart.m4a  
$
```

Let's get song titles without  
the other information

- List all the files as before, but only show me the 47th and subsequent characters in their names, and then just show me the first ten so I can check “47” was the right number (tail would show me the last ten).

# How did I find 47? Trial and error

```
$ ls 2016-08*NJ/*.m4a | cut -c 40- | head -1
3d1_01_New_York_City_Serenade.m4a      Too much
$ ls 2016-08*NJ/*.m4a | cut -c 50- | head -1
_York_City_Serenade.m4a                Too little
$ ls 2016-08*NJ/*.m4a | cut -c 45- | head -1
1_New_York_City_Serenade.m4a           Try the midpoint
$ ls 2016-08*NJ/*.m4a | cut -c 47- | head -1
New_York_City_Serenade.m4a             And the middle
$   of that: just right!
```

This is actually an algorithm called a “binary chop” that is used for many purposes

# Pro Tip for Experts

- Counting characters is hard work. The alternative is this, but involves understanding **regular expressions**

```
$ ls 2016-08*NJ/*.m4a | sed 's/.*d[1-4]_[0-9][0-9]_//' | head
New_York_City_Serenade.m4a
Wrecking_Ball.m4a
Badlands.m4a
Something_in_the_Night.m4a
The_Ties_That_Bind.m4a
Sherry_Darling.m4a
Spirit_In_The_Night.m4a
Santa_Claus_Is_Coming_To_Town.m4a
Independence_Day.m4a
Hungry_Heart.m4a
$
```

# Now, how many distinct?

```
$ ls 2016-08*NJ/*.m4a | cut -c 47- | sort -u | wc -l  
67  
$
```

Show me the 47th and subsequent character in each of these filenames, sort them into alphabetical order, remove the duplicates (-u = unique) and then count the results: 67

The commands run in parallel if your machine has more than one processor.



# What was played every night?

- We need another command, “`uniq -c`”, which replaces successive identical lines with a count of how many lines are the same:

```
$ ls 2016-08*NJ/*.m4a | cut -c 47- | sort | head
```

```
4th_of_July_Asbury_Park_Sandy.m4a  
American_Skin_41_Shots.m4a  
American_Skin_41_Shots.m4a  
Atlantic_City.m4a  
Backstreets.m4a  
Badlands.m4a  
Badlands.m4a  
Badlands.m4a  
Because_the_Night.m4a  
Because_the_Night.m4a  
$
```

```
$ ls 2016-08*NJ/*.m4a | cut -c 47- | sort | uniq -c | head
```

```
1 4th_of_July_Asbury_Park_Sandy.m4a  
2 American_Skin_41_Shots.m4a  
1 Atlantic_City.m4a  
1 Backstreets.m4a  
3 Badlands.m4a  
3 Because_the_Night.m4a  
1 Blinded_by_the_Light.m4a  
1 Bobby_Jean.m4a  
3 Born_to_Run.m4a  
1 Brilliant_Disguise.m4a
```

```
$
```

# So...

- We now need to pull out all the lines which have “3” in the first column.
- Unix has many “tiny programming languages”, which are rarely used in full, but everyone carries around little snippets.
- The oldest of those is a language called “awk” (yes, we know the birds are spelled “auk”), for its authors, Aho, Weinberg and Kernighan.
- If you find yourself writing multi-line awk programmes, it’s time you learned python or perl, but for this purpose, awk is fine...

# The bit of awk everyone knows

```
$ ls 2016-08*NJ/*.m4a | cut -c 47- | sort | uniq -c | awk '$1==3'  
3 Badlands.m4a  
3 Because_the_Night.m4a  
3 Born_to_Run.m4a  
3 Dancing_in_the_Dark.m4a  
3 Hungry_Heart.m4a  
3 Jersey_Girl.m4a  
3 New_York_City_Serenade.m4a  
3 Out_in_the_Street.m4a  
3 Rosalita_Come_Out_Tonight.m4a  
3 Shout.m4a  
3 Spirit_In_The_Night.m4a  
3 Tenth_Avenue_FreezeOut.m4a  
3 The_Rising.m4a  
$
```

`$1==3`: print all the lines whose first word is 3

# In fact, this is wrong

- It turns out that whoever put these packages together got sloppy over case, and there is a discrepancy.
- We can find this with the “tr” command, which converts all the characters in its first argument into the matching character in its second argument

```
$ tr 'abc' 'def'  
I am being careful  
I dm eeing fdreful  
$
```

# Different answer if we ignore

## case

Downcase all capitals



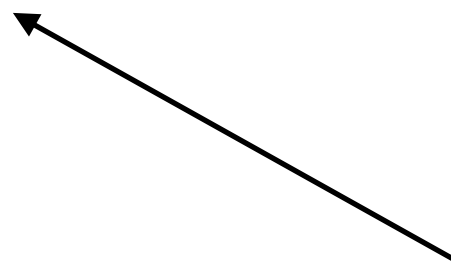
```
$ ls 2016-08*NJ/*.m4a | cut -c 47- | tr 'A-Z' 'a-z' | head -5  
new_york_city_serenade.m4a  
wrecking_ball.m4a  
badlands.m4a  
something_in_the_night.m4a  
the_ties_that_bind.m4a  
$
```

Note, by the way, that there are huge variations in versions of tr. “man tr” is your friend on the system you are using

```
$ ls 2016-08*NJ/*.m4a | cut -c 47- | tr 'A-Z' 'a-z' | sort -u | wc -l  
66  
$ ls 2016-08*NJ/*.m4a | cut -c 47- | sort -u | wc -l  
67  
$
```

Turns out Of and of were used

```
$ ls 2016-08*NJ/*.m4a | cut -c 47- | grep -i Jack  
Jack_of_All_Trades.m4a  
Jack_of_All_Trades.m4a  
$
```



# How did I Find This?

- Honestly, by eye: I just noticed it when looking at the “2” section.
- Sort also has the option “-f” to “fold” case for the purposes of comparison. The command “diff -c” (“context difference”) will show you the difference between two files in a readable format

```
$ ls 2016-08*NJ/*.m4a | cut -c 47- | sort -f -u > /tmp/ignorecase
$ ls 2016-08*NJ/*.m4a | cut -c 47- | sort -u > /tmp/withcase
$ diff -c /tmp/ignorecase /tmp/withcase
*** /tmp/ignorecase 2016-09-22 16:18:49.697449402 +0100
--- /tmp/withcase 2016-09-22 16:18:57.037358535 +0100
*****
*** 25,30 ****
--- 25,31 ----
    Incident_on_57th_Street.m4a
    Independence_Day.m4a
    Its_Hard_To_Be_A_Saint_In_The_City.m4a
+   Jack_of_All_Trades.m4a
    Jack_of_All_Trades.m4a
    Jersey_Girl.m4a
    Jungland.m4a
$
```

Lines  
marked +  
have  
been  
added

# Unix is great

- Note I didn't have to write programs, put commands in files, or anything tedious like that.
- I just typed commands, and the right thing happened. I could build them up in steps, and see the output for each intermediate point.
- You aren't expected to be able to do this immediately
  - I have, rather depressingly, 34 years of daily Unix experience
- But I hope you start to see why Unix is a nice environment for computer scientists to work in.