**TO DO: Create a table for all the UNIX shell commands?**

**PART 1 (1.5 hours)**

Most UNIX shell commands are LINE-BASED. Occasionally, you may want to do character by character processing (such as TR)

**cat:** the simplest filter

* The cat command copies its input to output unchanged
* When supplied a list of file names, it concatenates them into std out
* Options:
  + **–n** = number output lines (starting from 1)
  + **–s** = squeeze consecutive blank lines into single blank-line
  + **–v** = display control characters in visible form (E.g. ^C)
* **tac** copies files, but reverses the order of lines.
* Learn more about cat / tac via. Google and StackOverflow etc.

**wc:** word counter

* WC command is a summarising filter.
* Useful with other filters to count things.
* Options:
  + **–c** = counts no. of characters
  + **–w** = counts no. of words
  + **–l** = counts no. of lines
* Some filters find counting so useful that they define their own options for it.
  + E.g. grep –c

**tr**: transliterate characters

* TR command converts text char-by-char according to a mapping of characters



* Each input character from *sourceChars* is mapped to the corresponding character in *destChars*
* Example:



* sourceChars = ‘abc’ and destChars = ‘123’
* Therefore: a🡪1 b🡪2 c🡪3
* Note: tr doesn’t accept file name on command line (you can’t specify a file)
  + You have to do it from standard input.
  + E.g. 
  + The command wouldn’t work without the “ < “ less than character.
* ‘a-z’ is equivalent to ‘abcdefghijklmnopqrstuvwxyz’
* Characters that aren’t specified in sourceChars are copied unchanged to output.
* Options:
  + **–c** = map all chars NOT occurring in sourceChars (complement)
  + **–s** = squeeze adjacent repeated chars out (only copy the first char)
  + **–d** = delete all chars in sourceChars (no destChars)

**head / tail**: select lines

* Head prints the first *n* (default 10) lines of input.
  + E.g.  prints the first 10 lines of fileName
* Options
  + **–n** = changes the number of lines head/tail prints
* Tail prints the last *n* lines of input.
  + E.g.  prints the last 30 lines of filename
* Combine head and tail to select a range of lines.
  + E.g.  copies lines 81 … 100 to output.

**Egrep**: select lines matching a pattern

* The egrep command only copies to output, the lines in the input that match a specified pattern.
* The pattern is supplied as a regular expression on the command line (and should be quoted using single-quotes)
* Options:
  + **–i** = ignore upper / lower-case difference in matching
  + **–v** = only display lines that DO NOT match the pattern (complement)
  + **–w** = only match pattern if it makes a completed word

The **grep** family

* Egrep is one of a group of related filters using different kinds of pattern matching:
* **GREP** uses a limited form of POSIX regular exp (no **+** ? | or parenthesis)
* **EGREP** (extended grep) implements the full regex syntax
* **FGREP** finds any of several (maybe even thousands of) fixed strings using an optimised algorithm (does a literal match, doesn’t use regexp)
* **GREP = G**lobally search with **R**egular **E**xpressions and **P**rint.

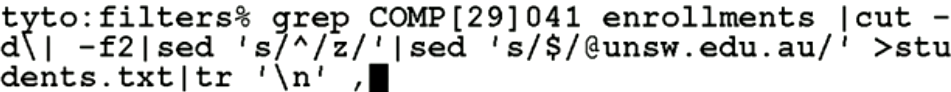
**Regular Expressions**

* A regular expression (regex) defines a set of strings.
* It is usually thought of as a pattern.
* It specifies a possibly infinite set of strings.
* They can be succinct and powerful
* Regular expression libraries are available for most languages.

**PART 2 (1 hour)**

**sed** program

* “replace this regulation expression with this string”
* Example:



* **grep COMP[29]041 enrollments**
  + search for pattern matching “COMP[29]041 enrollments”
* **cut – d\ | -f2**
  + cut out ‘\ |’ of field position #2
* **Sed ‘s/^/z/’**
  + Replace this reg expression ‘s/^/z/’ (anchor to start of the line)
* **Sed ‘s/$/@unsw.edu.au/’**
  + Replace this reg expression ‘s/$/@unsw.edu.au/’ (anchor to end of the line)
* **> students.txt**
  + Input data into a file called students.txt
* **tr ‘\n’ ,**
  + Change all the newline inputs into commas
  + **tr** command copies standard input to standard output with substitution or deletion of selected characters (in this case, substituting newline with commas)

NOTE: The lecture example above is incorrect, as the “**tr ‘\n’ ,”** should be placed before using “**>students.txt**”

Example: Web Server log data

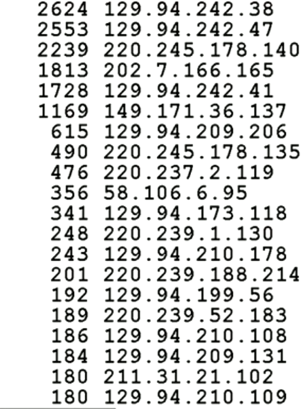
* How would I find out what computer accesses my web server most frequently?



* **cut –d ‘ ‘ -f1**
  + cut out space of field position #1
* **access\_log**
  + name of file
* **head**
  + display only the first count lines or bytes of each of the specified files.
  + If count is omitted, it defaults to 10

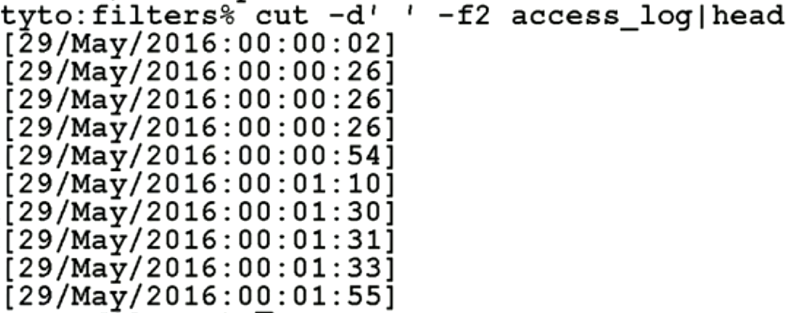
Example 2: Web Server log data



1. **Sort**
   * Sort lines of the text file
   * It only prints the sorted data as standard output, but doesn’t change whats in the file itself
2. **Uniq –c**
   * Uniq reads the specified input file comparing adjacent lines and writes a copy of each unique input line to the output file.
   * Uniq –c = precede each output line with the count of the number of times the line occurred in the input, followed by a single space.
3. **Sort –r**
   * Reverse sort
4. RESULT:
   * Sorted order of IP numbers in the order of frequency that computers  
     are accessing the web server.
   * E.g. IP 129.94.242.38 accessing the server 2624 times (most times)  
     VS IP 129.94.210.109 accessing the server 180 times (least times)

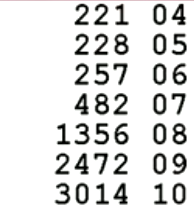
Example 3: Which hour of the day has the highest load on the web server?

* Same as above, except using **–f2**



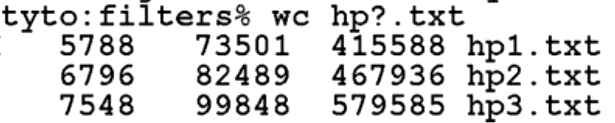
* However, I’m only interested in the HOUR:



* RESULT:
  + 4AM is the hour with the lowest load on the server
  + 10am is the hour with the highest load on the server

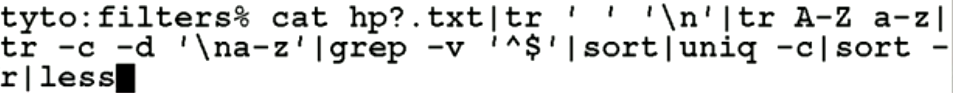
Example 4: Harry Potter novels.txt

* How many **lines, words, characters** in each HP novel?



1. **wc**
   * Displays number of lines, words and bytes (characters) contained in each input file.
2. **hp?.txt**
   * File name = hp(wildcard).txt

* What words was JK. Rowling using most frequently throughout all the HP novels? (CHALLENGE)



1. **cat hp?.txt**

* Concatenate all files with name “hp(wildcard).txt”
* This is necessary to be able to use the program **tr**

1. **tr ‘ ‘ ‘ \n’**

* Substitute all spaces with newline

1. **tr A-Z a-z**

* Substitute all uppercase characters (A-Z) with lower case characters (a-z)

1. **tr –c –d ‘\na-z’**

* ???

1. **grep –v ‘^$’**

* Invert search (-v condition) that don’t match an empty line
* **^$** would be an empty line as there is nothing in between the **beginning of the line** **“^”** and **end of the line “$”**

1. **sort**

* Sort the data

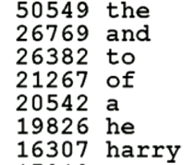
1. **Uniq –c**

* Show the number of times the line occurred in the input

1. **Sort –r**

* Reverse sort

1. **Less**

* Use LESS program (allows backwards movement in the file as well as forward movement)
* RESULT:
* “the” is the most used word (50,549 times)
* “harry” is the 7th most used word (16,307 times)
* NOTE: this disregards upper/lowercase chars.

Is there an Andrew in the HP novels?



1. **grep Andrew hp?.txt**
   * Search for the string “Andrew” in the input file “hp(wildcard).txt”

* RESULT:
  + “Andrew” appears in the 5th HP novel.

Replace “Hermoine” with “Andrew” in one of the novels:



1. **Sed ‘s/Hermione/Andrew/g’**
   * If you want to replace the regexp every occurrence in a line (not just the first occurrence), the **“g”** flag will do that.
2. **Grep Andrew | more**
   * Search for string “Andrew” and display using MORE program.

COMMAND INTERPRETERS

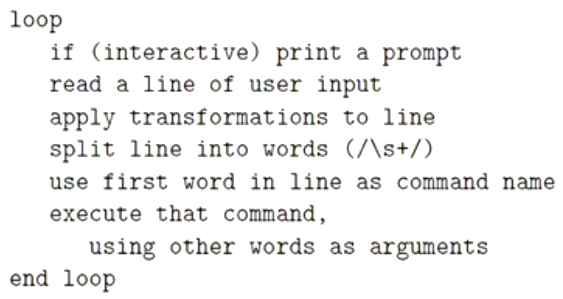
* Every computer has a way for the user to launch commands
* E.g. Android Phone 🡪 clicking on a graphical interface to run programs

We will use UNIX Shells to run programs.

* A **command interpreter** is a program that executes other programs.
* Command interpreters come in two flavours:
  + **Graphical** (E.g. Windows or Mac)  
    Advantage: easy to naïve users to start using system
  + **Command Line** (E.g. Unix Shell)

Advantage: programmable, powerful tool for expert users

* On UNIX/Linux, **bash** has become the default standard shell

All UNIX shells have the same basic mode of operations:

* The shell gets a line of input:
  + E.g. cat hp1.txt h2.txt
  + A “line of user input” could be a line from a file.  
    In that case, the shell is reading a “script” of commands  
    and acting as a kind of programming language interpreter.
* The shell breaks up the line of input into a words
* Takes the first word as the name of a program.
* Searches through a series of directories to find the program  
  to access and run it.
* Why can’t you just search your laptop yourself to find the program and run it?
  + There are probably a couple of million files and it could take a while to find it.
  + It’s better to have a small set of locations configured, ready for you to run.