Edicts on editors…editor wars!

- Emacs is a fine operating system, all it's missing is a good editor
  - developers can 'live' in emacs : code, OS cmds :compile, run, email…
  - Some say … (it is a great editor!)
    - it's big and slow, to load and run with all these features.
    - You need big hands and slow, with all these 'chords' (key combos!)
      - And that it gives you little finger/pinky RSI from Ctrl button use.
- Vim is just an editor…
  - And in the right hands can edit at the speed of thought!
  - But some say
    - it's got a steep learning curve (as if emacs or any flexible thing isn't) with modal options which confuse
    - it's as big and slow as emacs, by the time you add all the plugins
- But either way…
  - There is less mouse/trackpad/touchscreen and RSI related issues with GUI
  - trackballs, (dead mice) are better : support your arm rather than neck strain

Editor wars – Vim vs Emacs

- Emacs
  - Not always installed : can't install without admin rights
  - Feature rich, does lots of things
  - Can do most things within emacs … features already in, web, email etc.
  - Big & slower : memory requirements and features (vim plugins big also)
  - Easier to edit (no modes); harder to control – key combinations (save etc)
  - Twisted fingers & memory: lots of chords.. Ctrl, Alt etc.
    - Pain – the "emacs-pinky",
- Vim – philosophy and part of Unix
  - Universally available ; bundled (incl. vimtutor tutorial) with all nix,
    - Local branch of multinational telco management company uses 'vi' not even vim, simply because it is everywhere:- routers, exchanges, royalty free, robust, rapid!
  - Unix philosophy : do one thing really well
  - Just a text editor : but lots of plugins available
  - Lightweight & fast : even on phones (before plugins !)
  - Fast mover, more modes; but easier to control (save etc)
  - Easy on the fingers : normal, command & insert modes
  - Easy on others / conscience : help kids in Africa

Some history may explain peculiarities & strengths of editors

Historically machines were

- Powerfully Small requiring powerful tools & administration
  - computing power and storage by today's standard : needed powerful tools to minimise load on machine
  - Moore's empirical law…
    - Integrated Circuit (IC) transistor count doubling every 2 yrs since 1958
    - So in 2008, after 50 yrs = 25 doubling periods

    $=>2^{25}$ times as many transistors ~ increase by factor of 32 million!
- Powerfully large requiring controlled environment
  - Physically large, heavy and fragile :
    - punched card, paper & magnetic tape, humidity control
  - Substantial power requirements

Editors – did a lot, with a little!

- Storage
  - had no user accessible storage or I/O – punched cards – shuffle in a toss!?
  - Then tape : needed fairly uniform temperature & humidity
    - punched paper - tear,
    - magnetic – kept only in the computer room
- Display
  - Line printer…at end of run
  - Teletype … like a typewriter .. With a roll of paper
  - Visual Display Terminals… electronic / valve : like old TV's

Bottom line :

- all machine related items were at a premium, time, access, storage, CPU power, print & display : so short cryptic powerful commands
- Skilled competent people were relatively cheap compared to computers
- So offload work from the machine to people…

…little commands… with big results!

- Peculiarity (like line dancing)
  - Line (at a time) editing..
  - 2-character commands
  - Minimal display output
- Strength
  - Robust – tried & tested, built from the ground up
  - Speed – when everything was at a premium, super-efficient design
  - Powerful – both in operation and expressiveness, but excessively cryptic

Brief history of…computing time

- Hardwired – fixed circuit for a fixed purpose
  - Modern variations…with some more flexibility
    - FPGA – Field Programmable Gate Arrays
    - ASIC – application specific integrated circuit
- Programmable…initially only by programmers
  - By changing the wiring – the hardwired!
    - Like operators in manual telephone exchanges
  - By changing the instruction sequence… program
    - Machine code
    - Assembly language
    - Compiled languages

Program loading – to get it into the machine to run!

- Manually – in binary
  - Select a switch to choose to enter either
    - the address,
    - or data for the address
  - Set the switches representing data or address
  - Hit the Enter/Load switch to load the info
    - either address
    - or associated data
- Editing !? …at the flick of a switch … or (a power of) 2!?
  - Step through the addresses to check the data
  - Change any that needed it … all still in binary
- Next major advance… hexadecimal code - less bit errors!
  - 4- bits ⇔ 16 states ⇔ 0-9 A – F
  - Entered using a 16-key keypad
  - Displayed using a 7-filament bulb, forerunner of 7-segment LEDs

Next move : remove human error
Machine readable – sure & fast

- Punched card
- Paper tape
- Magnetic tape
- Needed controlled environment…with fairly constant temperature and humidity to avoid dimensional distortion of paper media, machines and even tape (which may absorb water slightly) causing loosening of the magnetic film.

Initial public service offering…offline editing & batch jobs…

- Well initially only private
  - no user storage,
    - Punched card resubmission decks, no online storage of code at all.
    - A bit like lottery cards where people mark numbers with pens, except holes were punched so light could shine (or pins could poke) through
    - Each card represented an 80-character / column line of text
  - no interactive use
    - Punched card 'typewriters' were expensive desk sized machines
    - Editing was done
      - By hand on grid-lined paper sheets,
      - given to the punch typists who repunched new cards,
      - which the programmer then reinserted (like line inserts) into his deck of cards, corresponding to a file.
    - And you thought line editing was bad using 'ed'
- Editing turnaround ranged from a few hours to a few days
- And getting the program run, about the same…
- You might get a few runs of a program in a week..or a month.
- You desk checked your program for logic, syntax and typographic errors!

Running a program

- Programs were submitted as a batch of jobs, each having it's own card sequence
  - Basic Job Control Language to accompany card deck…
    - Specifying upper requirements of each of
      - Time – seconds of CPU time … minutes at most.
      - Memory – Kb of RAM – in low single digits
      - File – Kb of disk – yes Kb ..in the 10's
    - For each of the following basic phases of running a program
      - compile requirements – space & time to translate to machine code
      - Load – with libraries to prepare to run
      - Run -
        - » Input Data file
          - Punched cards
          - Or on tape .. Perhaps on disk
        - » Output
          - To tape
          - Printed later..
    - Specification was like bidding for a tender..
      - Too high .. And it would be last in line, when most else was run
      - Too low .. And it would exceed costs and go bust & be dumped

Basic operating systems…

- Fully interactive
  - online editing
    - Could edit the files on the machine disk
  - job control
    - Could create 'job control files' to run jobs
      - These are the forerunners of OS scripts

Online editors

- So online editors were a big step forward…
  - You could change a file in minutes instead of…
  - turnaround with punched-card operators could be hours/days due to job queue
- But editors still had to be short and snappy to save storage or time
  "Do one thing and do it well"
  - Short : Commands & program (little RAM)
  - Snappy : Simple operations
  - Write to buffer space on disk or tape
  - Limited display on paper…
- And since
  - they were designed to be mighty efficient, fast, and need few resources,
  - Tradition rules

  then 'if it isn't broke, don't fix it!"

But you get the idea why editors work the way they do.
And are ***still*** great when working on megafiles too big for RAM!

File and process control

Clearly there also needed to be a way to manage

- File organisation
  - Since these held the data
  - Proliferated so needed to be organised
    - Moved, copied, archived to tape etc.
- Processes
  - Since these got the results
  - Need to be able to run them with appropriate files for
    - Input
    - output
  - Monitor them while they ran
  - Stop them if something went wrong

Editors -

- CLI – will generally work over simple remote connecstions ..e.g. ssh
  - Line editors: - a pain to use on one file, but easier to apply to many files.
    - ed, ex, (Scriptable stream editor:-sed)
  - Visual editors :- (may not seem very visual but practice makes perfect!)
    - Vi, vim, Pico, Nano etc.
- GUI – editors
  - Any amount
- IDE – Integrated Development Environments
  - Combined editor and execution environment
  - Editor can be customised
    - For user
    - For language (sometimes with plugins) – highlighting keywords & errors
  - Execution environment .. Facilitates
    - Recompilation of changed parts, relinking and running the code
    - Providing debugging options
      - Single step mode while viewing varibles and program flow
      - Breakpoints to run quickly through to single-step monitoring.

Editors

- GUI
  - easy to use on one file, generally a repetitive pain for many files
  - Generally won't work over simple remote connections…
  - Kwrite general purpose
  - Editors in Program Development environments
    - Automatic styles:
      - indentation,
      - colour coded keyword display,
      - word completion,
    - e.g. simplest KATE - shows
      - editor panel,
      - + CLI,
      - + filesystem tree

Kate - (KDE Advanced Text Editor)

- multi document editor, based on a rewritten version of the kwrite editing widget of KDE.
- multi-view editor which allows user to view
  - several instances of the same document with all instances being synced,
  - more files at the same time for easy reference or simultaneous editing.
- The terminal emulation and sidebar are docked windows
  - that can be plugged out of the main window,
  - or put back again at any time according to your preference.
- Some random features:
  - `* Editing of big files`
  - `* Extensible syntax highlighting`
  - `* Folding`
  - `* Dynamic word wrap`
  - `* Selectable encoding`
  - `* Filter command`
  - `* Global grep dialog`

Why ed? Roots of sed, ex, vi(m), uses regex

- ***ed*** is
  - Quite outdated & cumbersome
  - A line editor : generally works on a line at a time.
  - Designed for minimal output to noisy costly teletype long before VDU (Visual Display Units) (instead of a screen, there was a kind of typewriter for a kitchen roll – with costly paper, ribbons & brainstorming rackets!)
  - Therefore powerful compact **search** and **substitute** commands were developed **... that's useful!**
  - Based on **regular expressions ... they're useful**
  - Basic commands filter through to more advanced editors & ***man***... eg ***vi*** inherit ***w*** (write (save)) & ***q*** : quit
  - But the benefit is for search and substitute in sed...

Just ed4sed!?

- ed is
  - A line editor, works only on a line specified by
    - Line number
    - Regular expression match
  - With archaic cryptic commands
    - s/this/that/g (but that's about all you'll need to know!)
  - 'show and tell' – forget it : 'not saying nothing'
  - And still can make a total mess on the quiet!
- What better preparation is there for sed!?
- But don't waste any more than 5-10 mins on ed, for it will mess with your head!

Sed – a stream editor

- ***ed*** can do global replacement within a file
- ***sed*** can do global replacement in
  - All files
  - Or a subset
    - within a directory
    - Chosen by a filter .. Such as grep or whatever..
- Just pipe the files to sed, and it will change all.
- Basic similarity of search and substitute commands and regular expressions... that's all you need to remember from ed for sed.
- Regular expression test apps & sites exist,
- but if desperate, then test it on ed, then sed on one file, before trying it on all files.
- General script design reality (quick fix – rushed scrabble) – mix and match, stitch and scratch, fix and patch,

ed - line editor

ed filename

- File is copied into buffer, /tmp/ed.* where edits are made and can be
  - Either ***w*** - written out with changes made being written and saved using w
  - or ***q*** – quit simply abandoned without saving by using q - quit.
  - NB quitting without first writing loses changes since last writte.
- Unless a range is specified, it edits a line at a time, the current line.
- On opening the file, the current line is set to last line, for appending.
- If filename does not exist, may give warning and create it, automatically entering insert mode, for first line.
- Two modes
  - Command mode - works on the lines e.g. ***d*** for delete etc.
    - ***^C*** generally interrupts current command
  - Insert mode: just by entering i, a, or c followed by enter on a line.
    - ***i, a c*** – ***i***nserts before, ***a***ppends after or ***c***hanges entirely the current line with text entered terminated by a '.' on a new line.

ed - define the line(s)1

- In general, commands have the structure:
  [address [,address]]command[parameters]

Addressing lines;-

- single lines e.g. current line or specified by number
- Ranges of lines: 1,n etc. provided n does not exceed end of file
- Special symbols for ranges

.	- current line
$	- last line
1,$ or , or %	- first through last line => i.e. entire file
. , $ or ;	- current through last line, remainder of file
/re/	- next line with regular expression
?re?	- previous line with regular expression
	- note similarity with CLI history search

- Mark current line with a lower case letter - klc
  - e.g. kp - Then refer to line as 'p - single quote p

ed - define the line(s)1

- Use multipliers e.g. dd or d2 - delete 2 lines

+	- next line;
n or +n	- n^{th} next line
- or ^	- previous line
-n or ^n	- n^{th} previous line

- Many commands accept a suffix
  - p, l, n (print, list, enumerate)
    last line affected by command

ed - basic commands

- **a** - appends text after current line, terminated b . on a line
- (**l1,l2**) **c** - changes lines from **l1** to **l2** inclusive in the buffer, deleting current lines and entering append mode after **l1**, basically it replaces **l1**,**l2** with text entered until **.** on a line.
- **d** - delete, dd, d2, ddd, d3 - double / triple delete etc., current line is set to line after range deleted if one exists, else it is set to line before current range.
- **e** newfile - deletes current file buffer, and copies in newfile for edit ..
- **i** - inserts text before current line, terminated b . on a line
- **(l1,l2)m(l3)** - moves range (**l1, l2**) to after **l3**, which may be 0, and current line is set to after new relocated **l2**.
- **p, l, n** (print, list, enumerate) line changed by last command

ed - basic commands

- s/old/new - substitute new for old
- s/old/new/g - substitute new for old, globally within the line
- ,s/old/new/g - substitute new for old, globally within the file
- s/re/newtext/g - switch reg. expr. for newtext globally
- s/re/newtext/n - n^{th} occurrence reg. expr. for newtext
- ,G/re/ - interactively edit, by printing each line with an occurrence of re, allowing user to edit line when displayed (but not visually)
- u - undoes the last command
- (1,10)p – displays (prints on screen) lines 1 to n
- (1,$)p – displays (prints on screen) entire file - lines 1 to end of file.
  (Remember in regular expressions $ referred to end of line!)
- (1,$)w filename
  - - writes the lines from 1 to last ($) to file, filename
- (1,$)W filename
  - - appends the lines from 1 to last ($) to file, filename
- q - quits ed - should give warning if you have not saved changes
- Q - quits ed unconditionally

ed - options

- -s Suppress diagnostics e.g. if ed is run from a script
- -x Prompts for an encryption key to be used
  in subsequent reads and writes (see the `x' command).
- -p string
  - Specifies a command prompt which may be toggled on and off with the `P' command.
- file

  Specifies the name of a file to read.
  If file is prefixed with a bang (!), then it is interpreted as a shell command.
  In this case, what is read is the standard output of file executed via sh(1).
  To read a file whose name begins with a bang, prefix the name with a backslash (\).
  The default filename is set to file only if it is not prefixed with a bang.

ex - Yet Another Line Editor

- *ex*, just like *ed*, is a line editor
- ex is the line editor which *vi*, the visual editor, is based on
  - This means that **all** of the *ex* commands are available from *vi*
- Using the *ex* commands within *vi* can often simplify tasks

ex versus ed

- ed and ex have many similarities
  - They both use line addresses
  - They both use the same basic command set
    - p, m, t, /expr/, ?expr?, r, w, q, d, u, ! command
- They also have some differences
  - co as well as t for copy
  - abbreviations
  - sh to invoke a Unix shell
  - set and setno
  - : is the default command mode prompt