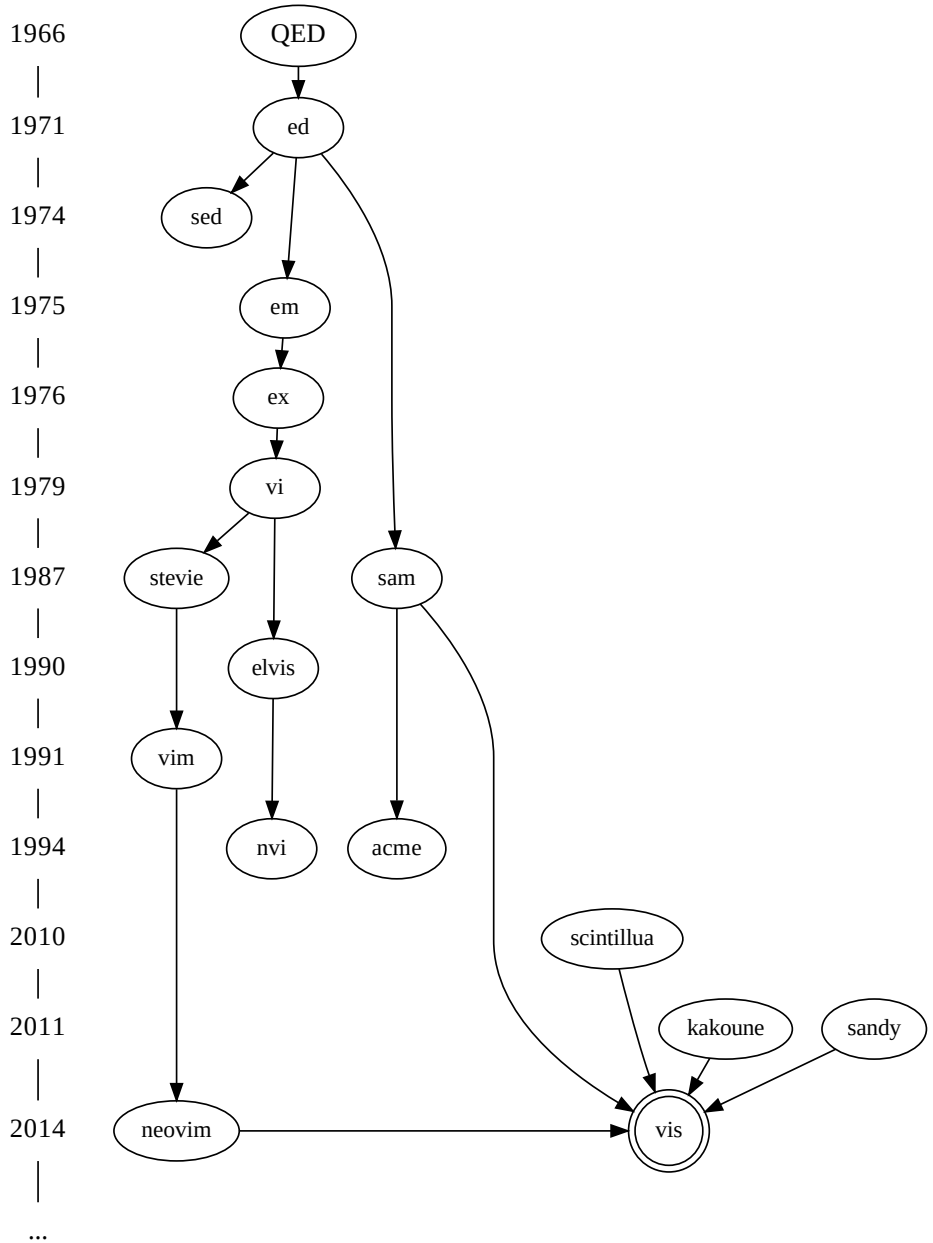
Vis Editor

Combining modal editing with *structural* regular expressions

Marc André Tanner

CoSin 2017, 17. May

Editor Lineage[1]



[1]Wikipedia, vi family tree, editor history project

TL;DR: What is this all about?

Demo

- `https://asciinema.org/a/41361`
- `asciinema play ...`

Modal Editing

- Most editors are optimized for *insertion*
- What about: navigation, repeated changes, reformatting etc.
- Different *modes*, reuse same keys for different functions

The vi(m) Grammar

Editing language based on:

- Operators (delete, change, yank, put, shift, ...)
- Counts
- Motions (start/end of next/previous word, ...)
- Text Objects (word, sentence, paragraph, block, ...)
- Registers

Demo: Motions and Text Objects

Motions

- Search regex `/pattern`
- Find character `f{char}`
- Match brace `%`

Text Objects

- Block `a{`
- Indentation `i<Tab>`
- Lexer Token `ii`

Structural Regular Expressions

The current UNIX® text processing tools are weakened by the built-in concept of a line. There is a simple notation that can describe the 'shape' of files when the typical array-of-lines picture is inadequate. That notation is regular expressions. Using regular expressions to describe the structure in addition to the contents of files has interesting applications, and yields elegant methods for dealing with some problems the current tools handle clumsily. When operations using these expressions are composed, the result is reminiscent of shell pipelines.

Structural Regular Expressions, Rob Pike, 1987

`sam(1)` Grammar

Simple Addresses

- `0` and `$`
- `n` line *n*
- `/regex/` and `?regex?`

sam(1) Grammar

Simple Addresses

- `0` and `$`
- `n` line *n*
- `/regex/` and `?regex?`

Compound Addresses

- `a1+a2` evaluate `a2` at end of `a1`
- `a1-a2` evaluate `a2` in reverse direction at beginning of `a1`
- `a1,a2` from beginning of `a1` to end of `a2`

Demo: Sam Addresses

- Select lines `13,15`
- Extend selection by 2 lines `.,+2`
- Grow selection in both directions by 2 lines `-2,+2`
- Select first occurence in file `0+/Emacs/`
- Select last occurence in file `$-/Emacs/`
- Select from first to last occurence in file `0+/Emacs/,$-/Emacs/`
- Select second occurence from current position `/Emacs/+//`

sam(1) Grammar

Text Commands

- a/text/ append text after range
- i/text/ insert text before range
- c/text/ change text
- d delete range

sam(1) Grammar

Text Commands

- a/text/ append text after range
- i/text/ insert text before range
- c/text/ change text
- d delete range

I/O Commands

- | filter
- > pipe out
- < pipe in
- ! run

sam(1) Grammar

Loops

- `x/regex/ command`
- `y/regex/ command`

  Extract every match / non-match and run `command`

sam(1) Grammar

Loops

- `x/regex/ command`
- `y/regex/ command`

  Extract every match / non-match and run `command`

Conditionals

- `g/regex/ command`
- `v/regex/ command`

  Run `command` on every match / non-match

sam(1) Grammar

Loops

- `x/regex/ command`
- `y/regex/ command`

  Extract every match / non-match and run `command`

Conditionals

- `g/regex/ command`
- `v/regex/ command`

  Run `command` on every match / non-match

Groups `{ ... }`

- All commands operate on initial state
- Changes must be non-overlapping

sam(1) by Example

- Select all occurences `x/pattern/`
- Search and replace `x/pattern/ c/replacement/`
- Indent `x/^/ i/\t/`
- Deindent `x/^\t/ d`
- Lowercase `x/Emacs/ x/E/ c/e/`
- `< seq 9, | sort -r, | fmt, | column -t, | tr a-z A-Z`
- `x/Emacs/ a/{TM}/`
- `x/Emacs/ /\{TM\}/ d`
- `x/Emacs/ { i/v/ d a/i/ }`
- Swap Emacs and vi

```
x/\<(Emacs|vi)\>/ {
        g/Emacs/ c/vi/
        g/vi/ c/Emacs/
}
```

Multiple Selections

Selections as core primitives.

Cursors are singleton selections.

Encourage a more interactive workflow than macros.

Demo: Variable Renaming

- Select current word ⟨C-n⟩
- Find next match ⟨C-n⟩
- Skip unrelated match ⟨C-x⟩
- Find next match ⟨C-n⟩
- Remove last match ⟨C-p⟩
- Rename c

Demo: Argument Reordering

- Select argument list `:x/\(.*\)/`
- Ineractively adjust selections `h`, `o`, `l`
- Split arguments `:y/,/`
- Trim selections `_`
- Rotate selections `+` or `-`

Or in one go: `:x/\(.*\)/ x/[a-z]+/`

Demo: Commenting out Code

- Select lines using visual line mode `V}h`
- Create selection at start of every line `I`
- Enter insert mode and type text `i#<Escape>`

Alternatively: `:x i/#/`

Demo: Alignment

- Select block `vi<Tab>`
- Split words `:x/\w+/`
- Align columns `<Tab>`
- Drop first column `<C-c>` or `:g%2`
- Reduce selections `<Escape>`
- Move to brace `f}`
- Align `<S-Tab>`

Selections: Creation

- `:x/regex/` and `:y/regex/`
- ⟨`C-k`⟩ and ⟨`C-j`⟩ line above/below
- ⟨`C-n`⟩ next match
- `I` and `A` in visual mode

Selections: Removal

- `:g/regex/` and `:v/regex/`
- ⟨`C-p`⟩ remove
- ⟨`C-x`⟩ skip
- ⟨`Escape`⟩

Multiple Selections

- Navigation with ⟨C-d⟩ and ⟨C-u⟩
- Rotation + and -
- Alignment ⟨Tab⟩ and ⟨S-Tab⟩
- Trim white space _

Selections: Manipulation through Registers

Save (`s`) and Restore (`S`) selections to/from registers.

- Union `|`
- Intersection `&`
- Complement `!`
- Minus `\`
- Pairwise Combine (`z|`, `z&`, `z<`, `z>`, `z+`, `z-`)

Demo: Selection Combination

DEMO

Lua as a Scripting Language

- Portable, powerful, efficient, lightweight, embeddable scripting language with support for higher order functions, closures, coroutines, ...
- No special purpose configuration file format/parser necessary
- Execute `~/.config/vis/visrc.lua` during start up

```
function win_open(win)
    -- Your per window configuration options e.g.
    vis:command("set number")
end

vis.events.subscribe(vis.events.WIN_OPEN, win_open)
```

Lua Plugin API

User definable:

- Key mappings
- Operators
- Motions
- Text Objects
- :-commands
- ...

Lua Plugin API Example

```
vis:operator_new("gq", function(file, range, pos)
  local ok, out, err = vis:pipe(file, range, "fmt")
  if not ok then
    vis:info(err)
  else
    file:delete(range)
    file:insert(range.start, out)
  end
  return range.start -- new cursor location
end, "Formatting operator, using fmt(1)")
```

LPeg based Syntax Highlighting

Parsing Expression Grammars (PEGs)

- More expressive than *pure* regular expressions
- Share similarities with CFGs
- Unifies scanning and parsing
- Closed under union, intersection, complement

LPeg

- Lua implementation using a virtual parsing machine
- Backtracking may be exponential for some patterns
- Reuse ≈ 130 existing lexers from Scintillua project

Design Philosophy[2]

- Leverage external tools (UNIX as IDE)
- Keep things simple, robust and fast
- Portable, lightweight, easily deployable

[2] Some of those are contradictory

Implementation

- $\approx$ 20K SLOC, standard compliant C99 editor core
- Lua used for run time configuration and in-process scripting
- man pages in mdoc(7) format
- ISC licensed

Future Plans

- Lua API Improvements
- Asynchronous Jobs/Events
- Quickfix functionality
- Alternative Text Management Data structures
- Client/Server architecture
- RPC interface
- Language Server Protocol support
- ...

Your Help Wanted!

- C hackers
- Lua developers (plugin API, syntax lexers, ...)
- Power users (testing/fuzzing infrastructure)
- Artists (color themes, logo, homepage, ...)
- Technical writers (documentation etc)
- Distribution packagers (OpenBSD, Fedora, openSUSE, macOS, ...)

Conclusion

Not just a vi(m) clone!

Powerful combination of:

- vi(m)'s modal editing
- sam's structural regular expressions
- selection manipulation primitives

Solid base suitable for experimentation with new ideas.

Questions?

https://github.com/martanne/vis

git://repo.or.cz/vis.git

mat@brain-dump.org

#vis-editor on freenode

Happy Hacking!

Changes compared to `sam(1)`

- *Interactive* refinements
- Multiple dots (a.k.a selections)
- More compact syntax

  ```
  x/Emacs/ { i/>/ a/</ }
  ```

- Relaxed ordering requirements of changes

  ```
  x/Emacs/ { d i/V/ a/i/ }
  ```

- Different regex anchor behavior

  ```
  x/[A-Za-z]+/ g/^i$/ c/I/
  ```

Changes compared to `sam(1)`

- Substitute command removed in favor of:

  ```
  x/regex/ c/replacement/
  ```

- Sub expression match references `\1` - `\9` and `&`

  ```
  x/Emacs/ c/>&</
  ```

- Count specifier for `g` and `v` commands

  ```
  g1/regex/      v1/regex/
  g%2/regex/     v%2/regex/
  g-1/regex/     v-1/regex/
  ```

- Tab character in text specifiers `\t`
- No mouse support!