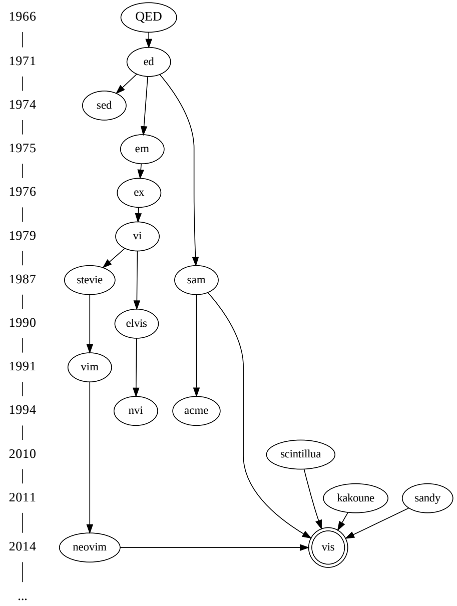
Vis Editor

Combining modal editing with *structural* regular expressions

Marc André Tanner

FOSDEM’18

Editor Lineage[1]



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

TL;DR: What is this all about?

`https://asciinema.org/a/PxFKNGvklhbP5sxjEXafIxY2Y`[2]

- Removing TODO comments
- Variable renaming
- Argument reordering
- Alignment

[2] See also explanations in appendix on the last slides.

Modal Editing

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

Modal Editing: The vi(m) Grammar by Example

Delete two inner blocks:

```
d2i{
```

Modal Editing: The vi(m) Grammar by Example

Delete two inner blocks:

```
d2i{
```

- `d` delete (operator)
- 2 (count)
- `i{` inner block (text-object)

Modal Editing: The vi(m) Grammar

[count] operator [count] (motion | text-object)

Modal Editing: The vi(m) Grammar

[count] operator [count] (motion | text-object)

count = [1-9][0-9]*

Modal Editing: The vi(m) Grammar

[count] operator [count] (motion | text-object)

$$\begin{aligned} \textit{count} &= [1\text{-}9][0\text{-}9]^* \\ \textit{operator} &= \texttt{c}\ (\text{change}) \mid \texttt{d}\ (\text{delete}) \mid \texttt{y}\ (\text{yank}) \mid \ldots \end{aligned}$$

Modal Editing: The vi(m) Grammar

[count] operator [count] (motion | text-object)

count = [1-9][0-9]*

operator = `c` (change) | `d` (delete) | `y` (yank) | ...

motion = `b` (begin) | `e` (end) | ...

Modal Editing: The vi(m) Grammar

[count] operator [count] (motion | text-object)

$$
\begin{aligned}
\textit{count} &= [1\text{-}9][0\text{-}9]^* \\
\textit{operator} &= \texttt{c}\ \text{(change)} \mid \texttt{d}\ \text{(delete)} \mid \texttt{y}\ \text{(yank)} \mid \ldots \\
\textit{motion} &= \texttt{b}\ \text{(begin)} \mid \texttt{e}\ \text{(end)} \mid \ldots \\
\textit{text-object} &= \textit{modifier} \mid \textit{object} \\
\textit{modifier} &= \texttt{a} \mid \texttt{i} \\
\textit{object} &= \texttt{w}\ \text{(word)} \mid \texttt{s}\ \text{(sentence)} \mid \texttt{p}\ \text{(paragraph)} \mid \texttt{\{} \mid \texttt{\}}\ \text{(block)} \mid
\end{aligned}
$$

Structural Regular Expressions[3]

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

[3] `http://doc.cat-v.org/bell_labs/structural_regexps`

Structural Regular Expressions by Example

- Search and replace:

```
x/Emacs/ c/vi/
```

Structural Regular Expressions by Example

- Search and replace:

```
x/Emacs/ c/vi/
```

- Indent

```
x/^/ i/\t/
```

Structural Regular Expressions by Example

- Search and replace:

```
x/Emacs/ c/vi/
```

- Indent

```
x/^/ i/\t/
```

- Deindent

```
x/^\t/ d
```

Structural Regular Expressions by Example

- Nested Loops

```
x/lua/ x/l/ c/L/
```

Structural Regular Expressions by Example

- Nested Loops

```
x/lua/ x/l/ c/L/
```

- Guard

```
x/\w+/ g/^i$/ c/I/
```

Structural Regular Expressions by Example

- Nested Loops

```
x/lua/ x/l/ c/L/
```

- Guard

```
x/\w+/ g/^i$/ c/I/
```

- I/O

```
-/start/+,+/end/- | sort
```

Structural Regular Expressions by Example

- Nested Loops

```
x/lua/ x/l/ c/L/
```

- Guard

```
x/\w+/ g/^i$/ c/I/
```

- I/O

```
-/start/+,+/end/- | sort
```

- Group

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

`sam(1)` Addresses

Simple Addresses

- `.` current selection (dot)
- `0` and `$`
- `n` line *n*
- `/regex/` and `?regex?`

`sam(1)` Addresses

Simple Addresses

- `.` current selection (dot)
- `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`

`sam(1)` Commands

Text Commands

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

`sam(1)` Commands

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

Multiple Selections

Selections as core primitives.

Cursors are singleton selections.

Encourage a more interactive workflow than macros.

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`⟩

Selections: Miscellaneous

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

Selections: Manipulation through Marks

Save (`m`) and Restore (`M`) selections to/from marks (`'a`-`'z`).

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

Lua as a Scripting Language

Portable, powerful, efficient, lightweight, embeddable scripting language with support for higher order functions, closures, coroutines, ...

```lua
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[4]

User definable:

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

[4] https://martanne.github.io/vis/doc/

Design Philosophy[5]

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

[5] Some of those are contradictory

Implementation

- $\approx$ 20K SLOC, standard compliant C99 editor core
- Lua used for run time configuration and in-process scripting
- ISC licensed

Future Plans[6]

- Review accumulated bloat
- Improve editing grammar
- Lua API Improvements
- Alternative Text Management Data structures
- Asynchronous Jobs/Events
- Use CRDTs?
- Client/Server architecture
- RPC interface
- Structural editing?
- Language Server Protocol support
- Efficient Regular Expression Engine
- Oberon & Acme inspired features
- ...

[6] In no particular order, no timeline given.

Your Help Wanted!

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

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!

References

Links for further information:

- These slides (TODO)
- Vis FAQ
- Vis Wiki
- `vis(1)` manual page
- Sam homepage
- Structural Regular Expressions, Rob Pike, 1987
- A Tutorial for the Sam Command Language, Rob Pike
- Sam quick reference card, Steve Simon

Demo: Removing TODO Comments

- Select TODOs `:x/TODO/`
- Apply lexer token text-object `ii`
- Delete comments `d`

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: 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>`

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/#/`