Vim 7: what's new

Stefano Zacchiroli

zack@debian.org

Bologna, 15/06/2006

Mi Presento

- Stefano Zacchiroli, AKA “Zack”
  - http://www.bononia.it/~zack
- Ricerca
  - mathematical knowledge management
  - collaborazione su web
- Free software:
  - sviluppatore Debian, principalmente maintainer
    - ocaml, vim, math stuff
  - bononia.it

Outline

- :help version7<Tab>
- o meglio: ciò che a Zack è piaciuto di vim7
  - spell checking
  - tab pages
  - undo branches
  - internal grep & location list
    - “ripasso”: QuickFix
  - omni completion
- altre novità

WARNING

- Contiene opinioni FORTI

  (in particolare nei riguardi di Emacs :-))

- Socrate userebbe Vim:

  “So di non sapere (come si usa Vim)”

  cogliete l'occasione di questo talk^W BOF per condividere ciò che (non) sapete

Spell Checking

Spell Checking – Perché?

- osserviamo la comunità:
  - gli utenti emacs vivono nella dicotomia: emacs per programmare (e per un'altra caterva di attività insensate per un editor), vim per l'"editing veloce"
  - gli utenti vim usano vim per tutto ciò per cui abbia senso usare un editor
- quante attività di queste ultime beneficiano dello spell checking?
- esistevano già soluzioni vim-based non soddisfacenti
  - vimspell

Spell Checking – Uso

- Idea: highlighting group per evidenziare parole mispelled
- Opzioni
  - 'spell' toggle
  - 'spelllang' lista
- Motion
  - [s,]s mnemonic ('s' per spell)
- Status
  - zg, zw mnemonic ('g' per good, 'w', per wrong)
  - z=

Spell Checking – Dettagli

- i file per i dizionari (*spellfiles*) sono locati in directory “spell/” contenute in componenti di 'runtimepath':
  - file .spl (i dizionari)
  - file .sug (regole per suggerimenti basati su soundfolding)
- sono encoding-specific
- il formato interno è simile ad un trie, caricato da 'spell'
- si ottengono da dizionari di mydict (OO.org) con :mkspell
  - nei sorgenti di vim trovate le ricette per AAP
  - do not try it at home (up to 1 Gb di RAM, ore di CPU)
- in Debian (soon): vim-spellfiles-XX

Tab Pages

(aka “tab”)

Tab Pages – Perché?

- varie filosofie di editing di file multipli in vim: argument list, buffer hidden, finestre
- la necessità di context switch induce spesso la necessità di disaccoppiare sessioni di editing
- soluzioni parziali
  - :mksession
  - multiple top level windows (non implementata, GUI only)
- soluzione di vim7
  - notebook a-la GTK

Tab Pages – Caratteristiche ed Uso

- caratteristiche:
  - indipendenza dalla UI, istanziati alla UI in uso
  - i layout di finestre sono tab-specific, i buffer condivisi
- creazione e distruzione
  - :tabe, :tabnew, :tabclose, :tabf, ... (mnemonic, prefisso “tab”)
- navigazione
  - :tabn, :tabN
  - comandi specifici per la UI in uso (e.g. CTRL+PageUp/Down)
- meta-editing
  - :tab {cmd}
  - :tabdo {cmd}

Undo Branches

(aka “undo ad albero”)

Undo Branches – Perché?

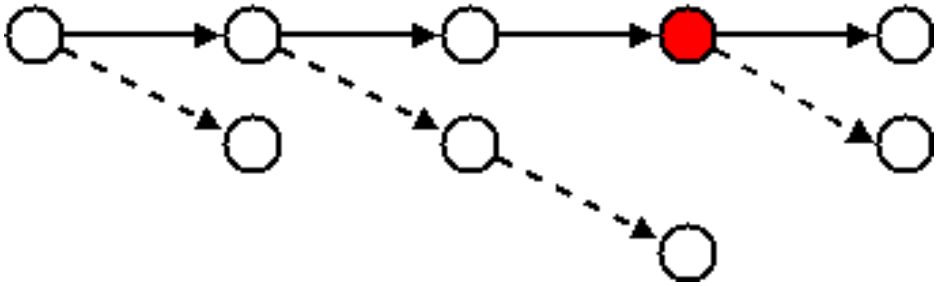
- L'history di undo di vim è da molte versioni pressoché illimitata
- Scenario
  - ricevete una mail chilometrica, rispondete (ETA: 2h)
  - vi accorgete che (all'inizio) avete cancellato un paragrafo che volete quotare
  - vi addormentate su 'u', copiate il paragrafo in un registro e siete pronti ad addormentarvi su 'CTRL-R'
  - **OOPS**: involontariamente fate una modifica (un classico? '~' invece di <Esc>)
- Quiz: che fine ha fatto il vostro testo costato 2h di lavoro?

Undo Branches – Idea

- ogni volta che una modifica porta alla perdita di un “futuro” di redo viene creato un *branch*
- 'undolevels' regola il massimo numero di undo possibili
- che UI vi aspettate per questa feature?

Undo Line in Vim (<< 7.0)

Undo Branches in Vim (>= 7.0)



Undo Branches – Uso

- risposta: vedere l'albero delle modifiche
- la UI di vim7 è purtroppo più sdozza
- è basata sull'ordinamento totale indotto dal tempo
- comandi di ispezione dei branch
  - :undolist
- comandi di controllo del flusso canalizzatore
  - relativi: g- (:earlier), g+ (:later)
  - assoluti: undo {N}
- 'u' e 'CTRL-R' mantengono la vecchia semantica, spostandosi sulla linea del tempo corrente

Internal Grep

&

Location List

Detour: Comandi QuickFix [1/2]

- Vim offre supporto al ciclo edit-compile-edit ispirandosi all'opzione quickfix del compilatore C Manx's Atzec di Amiga (!)
- idea: salvare l'output del compilatore e parsarlo (via 'errorformat') riconscendo le coppie <messaggio, locazione>
- il risultato del parsing viene mantenuto nella *error list*
- gestione della error list
  - :make — compila invocando 'makeprg'
  - :copen, :cclose — (mnemonic: 'c' per compile)
  - :cc — (mnemonic: la seconda 'c' per current)
  - :cnext, :cprevious

Detour: Comandi QuickFix [2/2]

- in Vim l'idea di QuickFix è stata applicata a `grep`
- compilazione = :make + 'makeprg' + 'errorformat'
- grep = :grep + 'grepprg' + 'grepformat'
- i risultati (parsati) di grep vengono memorizzati nella *error list*

Internal Grep

- in vim7 è stato implementato un grep interno
- vantaggi
  - maggiore portabilità
  - grep all'interno di file comprensibili a vim (e.g. gzippati)
  - uso dei pattern di vim
  - feedback
- uso
  - :vimgrep {pattern} {file} ...
  - :vimgrep /{pattern}/ {file} ...
- tip of the day: use **, Luke!

Location List

- qualsiasi uso di :make o :grep salva il proprio output nella *error list* (cancellando contenuto e stato precedente)
- vim7 introduce le *location list*
- error list specifiche della finestra corrente
- uso:
  - :lgrep (vs :grep) mnemonic 'l' per location list
  - :lvimgrep (vs :vimgrep)
- gestione della location list:
  - :ll (vs :cc)
  - :lnext, :lprevious, ...

Omni Completion

(aka “Intellisense ®”)

Omni Completion – Overview

- yet another completion: CTRL-X CTRL-O
  - nello stesso spirito di CTRL-X CTRL-{N,L,F,],...}
- idea: il codominio del completamento dipende da ciò che precede il cursore
  - caso d'uso: completamento dei metodi in un linguaggio OO
- configurabile (tipicamente da ftplugins) via la funzione referenziata da 'omnifunc'
- supporto legacy:
  - alcuni filetype hanno supporto built-in per omni completion
  - meta-support via regioni di syntax highlighting

Omni Completion – Case Studies

- filetype=python
  - completa il contenuto di moduli, preview delle docstring
  - richiede vim compilato con supporto python, utilizza reflection su moduli
  - non completa i metodi delle istanze (perché?)
- filetype=xml (:help ftp-xml-omni)
  - completa elementi, attributi ed entità in base ad un simil-DTD (*XML data file*)
  - data file built-in per (x)html e xsl (/usr/share/vim/vim70/autoload/xml/*.vim)
  - da DTD a data file con l'utility dtd2vim

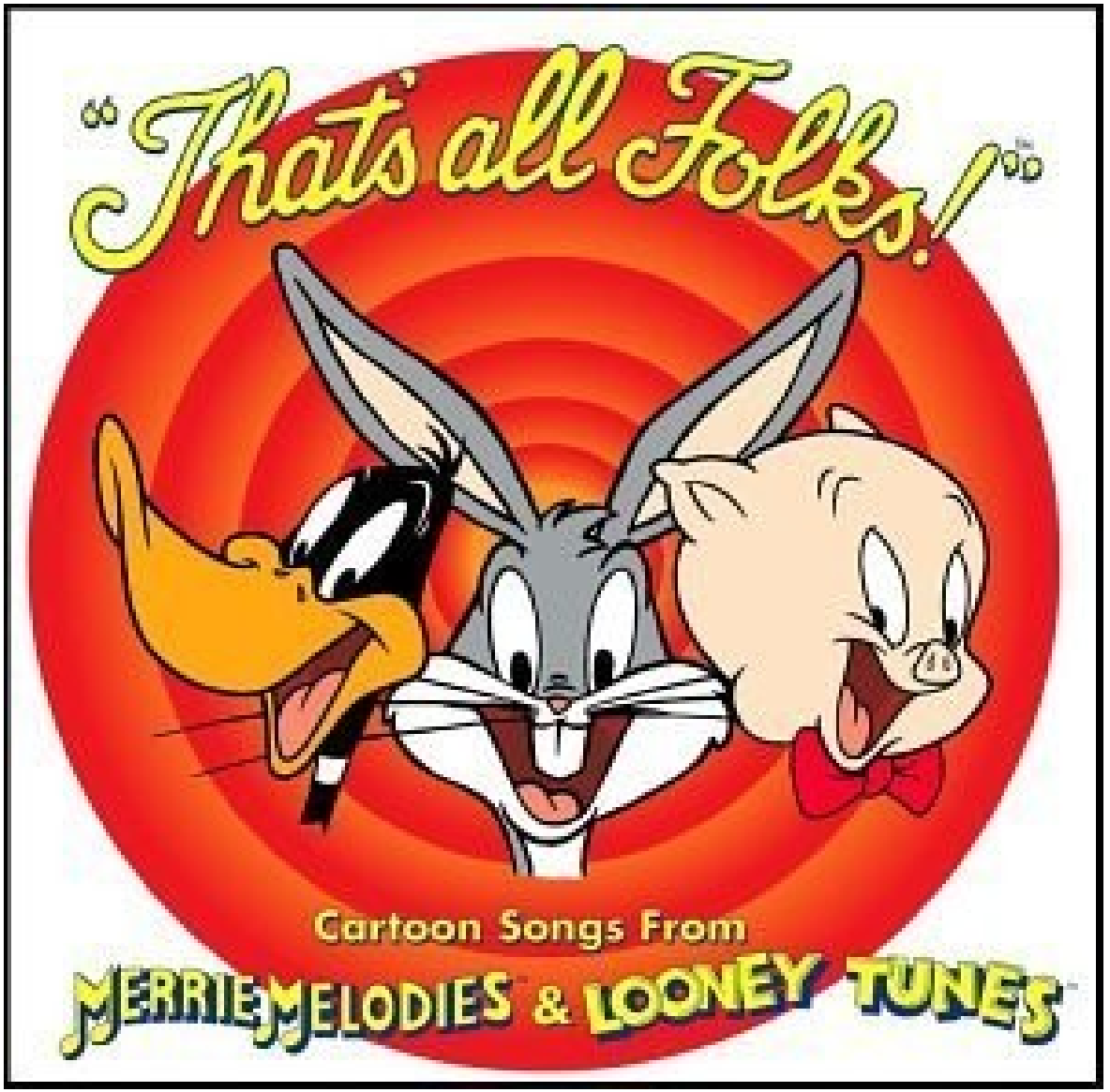
Altre Novità (degne di nota) [1/2]

- utili
  - nuovi tipi di dato in vimscript: list, dictionary, funcref
  - definizione di operatori
  - support unicode up to 6 combining characters
  - scroll all'indietro in messaggi lunghi
- futili
  - baloon expressions
  - matchparen, caratteri invisibili (bug?)
  - posizionamento del cursore a lastcolumn + 1
  - supporto per l'interprete MzScheme

Altre Novità (degne di nota) [2/2]

- plugin
  - netrw + file explorer = remote browsing
  - nuovo plugin standard: vimball
  - nuovi plugin standard: .tar e .zip browsing
- tonnellate di BugFix

Q & A time!


"That's all Folks!"
Cartoon Songs From
MERRIE MELODIES & LOONEY TUNES