Being Productive With Emacs

Part 2

Phil Sung
sipb-iap-emacs@mit.edu
http://stuff.mit.edu/iap/emacs

Special thanks to Piaw Na and Arthur Gleckler
Previously...

- Emacs as an editor
  - Useful features
  - Motifs in emacs
  - Learning more
Previously...

• Acquiring Emacs
  - Already installed on Athena (v.21)
  - Ubuntu: emacs-snapshot-gtk package (v.22)
  - Gentoo: emacs-cvs package (v.22)
  - Windows: EmacsW32 installer
Previously...

• Learning more about emacs
  - Look up an existing function or key
    • C-h f, C-h k
  - Apropos (search for commands)
    • C-h a
  - Help about help facilities
    • C-h C-h
Previously...

- Learning more about Emacs
  - Emacs tutorial
    - `C-h t`
  - Emacs manual
    - `M-x info, select emacs`
Previously...

• If you're stuck...
  – Cancel: c−g
  – Undo: c−/ or c−_
Today

• Customizing Emacs
• Elisp basics
• Defining a new command
Customizing Emacs

- Almost every aspect of the editor can be customized
  - More fine-grained control than major/minor modes
  - In general, use `M-x eval-expression`
  - Show trailing whitespace on lines
    `(setq show-trailing-whitespace t)`
  - Show column numbers in mode line
    `(column-number-mode t)`
Customizing Emacs

- Customization buffer: `M-x customize`
  - Browse, point and click for customization options
  - No elisp necessary, but capabilities are limited
Review: running elisp code

- Evaluate elisp with `M-x eval-expression`
  - Example function call: `(+ 2 4 6)`

- Get or set variables:
  - `sentence-end-double-space`
  - `(setq inhibit-startup-message t)`

- Sometimes these correspond to commands:
  - `(forward-char)` is same as `C-f` or `M-x forward-char`
Writing elisp code

- Use the *scratch* buffer for playing with elisp
  - C-x C-e to evaluate
  - Move to ~/.emacs for permanent changes
Your .emacs file

- `C-x C-f ~/.emacs`
- Elisp code here is loaded when Emacs starts
  - Run any valid lisp code here
  - Set variables, keybindings to your liking
  - Define your own commands
- `M-x customize` works by adding code to your .emacs
Why bother with elisp?

• Macros record and play back key sequences
  - Start recording macro: `C-x (`
  - Stop recording macro: `C-x )`
  - Execute last macro: `C-x e`

• Great for automating tedious tasks
  - `C-x e e e e ...`
  - `C-u 100 C-x e`
Macro example

<table>
<thead>
<tr>
<th>Course Code</th>
<th>Semester</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.00</td>
<td>12</td>
<td>programming</td>
</tr>
<tr>
<td>6.001</td>
<td>15</td>
<td>sicp</td>
</tr>
<tr>
<td>6.002</td>
<td>15</td>
<td>circuits</td>
</tr>
<tr>
<td>6.003</td>
<td>15</td>
<td>linear-systems</td>
</tr>
<tr>
<td>6.004</td>
<td>15</td>
<td>digital</td>
</tr>
<tr>
<td>6.011</td>
<td>12</td>
<td>signal-proc</td>
</tr>
</tbody>
</table>

Let's remove this column repeatedly.

M-f M-f M-d C-n C-a repeatedly
Why elisp?

- Macros only repeat canned key sequences
- Sometimes you need:
  - Calculations
  - Control flow
  - User interaction
  - Additional features
  - Maintainability
Elisp is...

- an implementation language
- a customization language
- an extension language
Elisp for implementation

• Example: \texttt{M-x calc}
  - \texttt{C-h f} to see where \texttt{calc} is defined
  - \texttt{RET} on filename in help buffer to view source code
Elisp for customization

- Set variables and options
- Persistent customizations can go in .emacs
- Compare to \texttt{M-x customize}
Elisp for extensions

- Alter behavior of existing commands
- Define your own commands, functions
- Define new modes
Why elisp?

• It's the implementation language
• Dynamic environment
  – No need to recompile/restart emacs
  – Easily override or modify existing behaviors
• Simple one-liners are sufficient to do a lot!
Getting started

• Similar to lisp and scheme

• Use *scratch* buffer as a temporary work space
  
  - or activate `lisp-interaction-mode` anywhere
  
  - `C-x C-e` after an expression to evaluate it
  
  - or use `M-x eval-expression (M-:)`

• Example: setting a variable
  
  - `(setq undo-limit 100000)`
Getting started

• Evaluating an expression can mean
  – Performing some computation/action
  – Displaying the value of a variable
  – Defining a function for later use
Basic elisp

• These are expressions (“atoms”)
  − 15
  − “Error message”
  − best-value

• These are also (“compound”) expressions
  − (+ 1 2)
  − (setq include-all-files t)
Setting variables

- Set variable by evaluating
  `(setq undo-limit 100000)`
  - i.e. do `M-:` (setq ...) [RET]

- Read variable by evaluating `undo-limit`
  - i.e. do `M-:` undo-limit [RET]

- Find out more about any variable with `C-h v`
Common customizations

- Configuration options
- Set your own keybindings
Configuration options

- Some customizations are done by setting variables
  - `(setq undo-limit 100000)`
  - `(setq enable-recursive-minibuffers t)`
  - `(setq fill-column 80)`
Configuration options

- Other options are exposed as their own functions

  - `(menu-bar-mode nil)` (Hide menu bar)
  - `(icomplete-mode)` (Show completions continuously)
  - `(server-start)` (Start emacs server)
More about variables

• Many variables are boolean
  - Usually a distinction is only made between \texttt{nil} and non-\texttt{nil} values (e.g. \texttt{t})

• Look in function documentation to see which variables can alter the function's behavior

• \texttt{C-h v} to get documentation for a variable
Key bindings

- We've seen two ways to invoke commands
  - C-x n w (key invocation)
  - M-x widen (M-x invocation, or invoking by name)
- Emacs binds each key to a command in a keymap
  - A keymap can be specific to a mode or feature
  - Bindings may be changed at any time
Customizing key bindings

- `(global-set-key [f2] 'split-window-horizontally)
- `(global-set-key "\C-o" 'find-file)
- `(global-set-key "\C-x\C-\\" 'next-line)

binds to \texttt{C-x C-\\}
Customizing key bindings

- A binding can be set to apply only in a particular mode
  
  ```lisp
  (define-key text-mode-map        
   "\C-cp"                        
   'backward-paragraph)
  ```

  binds to **C-c p**
Keybindings

• What keys can you assign?
  – Reserved for users:
    • C–c [letter]
  – Reserved for major and minor modes:
    • C–c C–[anything]
    • C–c [punctuation]
    • C–c [digit]
Calling commands

• Any command you use can be invoked programmatically by elisp
  – Often, `M-x my-function` is accessible as `(my-function)`
  – For key commands, look up the full name first

• Use commands as building blocks for more complex behaviors
Hooks

• Specify a custom command to run whenever a particular event occurs, e.g.
  - when a particular mode is entered
  - when any file is loaded or saved
  - when a file is committed to CVS
Hooks

- (add-hook
  'vc-checkin-hook
  '(lambda ()
     (send-email-to-group)))
Hooks

- (add-hook 'java-mode-hook
  '(lambda () (setq indent-tabs-mode t)
     (setq tab-width 4)
     (set-fill-column 80)))
Hooks

• General template

- `(add-hook 'name-of-hook
  '(lambda () (do-this)
    (do-that)
    (do-the-other-thing)))`
Hooks

• To find available hooks:
  - Every major mode has a hook
  - `M-x apropos-variable` and search for "hook"
Defining your own functions

• `(defun function-name (arg1 arg2 ...)
   "Description of function"
   (do-this)
   (do-that)
   (do-the-other-thing))

• Invoke with:
  `(function-name one two ...)`
Strategy for making functions

• Find key commands that would have desired result
• Replace key commands with elisp function calls
A simple function

• (defun capitalize-backwards ()
    "Capitalize last letter of a word."
    (backward-word)
    (forward-word)
    (backward-char)
    (capitalize-word 1))
Not every function is a command

- Functions need arguments:
  - `(defun square (x) (* x x))`
  - `(square 5) ==> 25`

- Commands don't say what arguments to substitute
  - `M-x square ==> ??`

- *Interactive* specification needed to say what arguments to fill in
A simple command

• (defun capitalize-backwards ()
  "Capitalize last letter of a word."
  (interactive)
  (backward-word)
  (forward-word)
  (backward-char)
  (capitalize-word 1))
Problem

• This command moves the cursor
  - This can be distracting if the user isn't expecting it
Restoring the cursor

• (defun capitalize-backwards ()
   "Capitalize last letter of a word."
   (interactive)
   (save-excursion
      (backward-word)
      (forward-word)
      (backward-char)
      (capitalize-word 1)))
Useful functions

- (point)
- (point-max)
- (current-buffer)
- (message "This is the answer: %s" answer)
Local variables

• (let ((a new-value)
        (b another-value)
        ...) (do-something)
        (do-something-else))
Example: counting word length

• (defun word-length ()
   "Prints the length of a word."
   (interactive)
   (save-excursion
     (backward-word)
     (let ((a (point)))
       (forward-word)
       (let ((b (point)))
         (message "Word is %d letters"
                  (- b a))))))
Getting help with elisp

- Manuals
  - M-x info, then select elisp or eintr

- Learning by example
  - Function documentation (C-h f or C-h k) always gives a link to the function's source code
Next week...

- Control flow
- User interaction
- Commands for manipulating text
- Other extension methods