

Tools for effective data/file manipulation
and greater research productivity
(and happiness)

Wes Huang
Algorithmic Robotics Lab
Department of Computer Science
Rensselaer Polytechnic Institute

November 17, 2004

Learn to use your tools well!

- Many data/file manipulation tasks:
 - reformat a data file
 - scan files for information
 - rename batches of files
 - etc.
- Don't do it manually!
- Don't write a C++ program!
- Some tools at your disposal:
 - GNU Emacs
 - BASH shell (& scripts)
 - Ruby

GNU Emacs

- *Extremely* powerful editor
- Learn the basic key bindings! A few examples:
 - C-v, M-v \equiv scroll up/down page
I see too many students who press the up arrow and wait!
 - C-s, C-r \equiv isearch forward/backward
- Learn key bindings beyond the basics! For example:
 - M-a, M-e \equiv backward/forward sentence
You need to have 2 spaces after sentence ending punctuation for this to work.
 - M-{, M-} \equiv backwards/forwards paragraph
 - C-M-b, C-M-f \equiv backwards/forwards s-expression (balanced expression)
 - C-u \equiv $\times 4$ multiplier. Examples:
 - * C-u M-f forward 4 words
 - * C-u C-u C-p up 16 lines
 - * C-u C-u C-u ; 64 semicolons
 - M-% \equiv query replace. Type original text, replacement text, and then:
 - * space to replace
 - * 'n' to not replace
 - * '!' to replace the rest (no more questions)
 - * there are other commands...

GNU Emacs keyboard macros

- record a sequence of keystrokes/commands and then repeatedly execute them.
- `C-x (` \equiv starts recording macro
`C-x)` \equiv stops recording macro
- When you define the macro, you will do the operation on the first instance.
- Try out the macro a few times to make sure it's doing the right thing. (Then run it lots of times.)
Know the undo command: `C-/` or `C-_`
- If the “bell rings” (i.e., you get an error beep), then you have to start over. (Will also terminate the macro during execution.) Most commonly because you search for a string that isn't found.
- To execute macro: `C-x C-e`
Use prefix commands to execute multiple times:
`C-u C-x C-e` \equiv 4 times
`C-u C-2 C-7 C-x C-e` \equiv 27 times

GNU Emacs keyboard macro tips

- You need to be strategic about the sequence of commands!
- Create a line/record/instance oriented macro:
 - Start cursor at beginning of line/record/instance
 - Do the operation
 - Move the cursor to the beginning of the next line/record/instance
- Helpful commands:
 - C-a to put cursor at beginning of line
 - C-} to move to the beginning of the next paragraph (useful when records are separated by blank lines)
 - C-s to search forward to the next instance
- Be careful about variations in your data, for example:
 - tabs versus spaces
 - multi-word last names
 - variations in spacing

GNU Emacs keyboard macro examples

- Insert a “> ” at beginning of each line. (I know there’s probably a better way to do this, but I actually use this macro a lot.)

```
C-x (  start macro (assume cursor at beginning of line)
>      type characters
C-a    beginning of line
C-n    next line
C-x )  end macro
```

- Change: `\epsfig{file=pic.eps}`
to: `\includegraphics{pic}`

```
C-x (      start macro
C-s \epsfig  search for “\epsfig”
RET        stop search
M-BKSP    backward delete word (“epsfig”)
includegraphics type text
C-s file=   search for “file=”
RET        stop search
M-BKSP    backward delete word (“file=”)
C-s .eps    search for “.eps”
RET        stop search
M_BKSP    backward delete word (“eps”)
BKSP      delete character (“.”)
C-x )      end macro
```

GNU Emacs Major Modes

- Many different modes for different types of files and operations:
 - Fundamental
 - Text
 - LaTeX
 - HTML
 - C, C++
 - DirEd
 - and many more...
- Major modes have specialized key bindings
- For example, in \LaTeX mode:
 - $\text{C-c C-f} \equiv$ run \LaTeX on buffer
 - $\text{C-c C-o} \equiv$ open \LaTeX block
 - * asks what kind of block you want (with autocompletion when you hit tab/space)
 - * inserts $\text{\begin\{...}}$ and $\text{\end\{...}}$
 - * doesn't work for "document" for some reason
 - $\text{C-c C-e} \equiv$ end \LaTeX block
closes the closest open $\text{\begin\{...}}$ block
 - $\text{M-RET} \equiv$ insert \item

BASH shell

- Useful for file manipulation and applying UNIX commands
- You can write BASH shell scripts, but...
- You can do a lot from the command line!
- One of the most useful commands:
for var in ... do ... done

- For example:

```
for d in */*.scm; do
  mv $d scheme_progs/
done
```

You can enter this on the command line!

- The “word list” given (after “in”) need not be file names. For example:

```
for d in 3 2 1; do let x=$d+1;
  mv file$d.txt file$x.txt; done
```

Renames file3.txt to file4.txt, file2.txt to file3.txt, etc.

Need to use a let statement to make an integer variable.

BASH shell variables

- Often you only want part of a string variable:

`${d#*/}` remove everything from start to the first /

`${d##*/}` remove everything from start to the last /

`${d%.*}` remove everything from last . to the end

`${d%%.*}` remove everything from first . to the end

- For example:

```
for d in *[1-9]*.scm; do
  mv $d ${d%%.*}-${d#*.}
done
```

Renames files of the form `whuang.1.scn` to `whuang-1.scn`

Ruby

- My new favorite scripting language!
- BASH is good for file manipulation and using UNIX commands to process files. It is not well suited to directly process information in the file! Ruby is well suited for this.
- Ruby (so I am told) is like a combination of:
 - Smalltalk (a very object oriented language)
 - Perl (powerful “write-only” scripting language)
- If you already know perl, maybe you don't want to learn ruby.
If you don't know perl, learn Ruby instead!
- Easy file processing
- Provides extensive libraries for:
 - Arrays
 - Strings
 - Hash tables
 - http
 - date/time
 - much, much more...

Ruby example

Select and rearrange fields in a comma/tab separated spreadsheet file

```
# These are the fields I want in the proper order
outfields = ["Name", "Quiz1", "Quiz2", "Assign1", "Assign2"]

# read first line of file (which has column headings)
headingstr = gets

# split the string to create an array of the column headings
headings = headingstr.split(/,|\t/)

# read each line of the file
while instr = gets
  # split it on commas/tabs to get each field
  data = instr.split(/,|\t/)

  # create an array of the selected fields in output order
  # this uses a "closure" --- like a lambda fn
  # for each field, it looks up its index in the headings array
  # and selects that fields in the current data
  rarr = outfields.each { |f| data[headings.index(f)] }

  # turn this array of strings into a single string
  # with a comma separating each field
  outstr = rarr.join(",")

  puts outstr
end
```

References

- GNU Emacs

- The O'Reilly book is good; it goes beyond the basics but doesn't get too advanced.
- Emacs info mode contains documentation of most everything. Type: `C-h i`
- There is also plenty of stuff online.

- BASH

- Again, I like the O'Reilly book — I keep it right at my desk. Nice presentation. Covers a lot of stuff.
- The UNIX man page has pretty much everything you want to know, albeit in a somewhat less accessible format.
- Plenty of stuff online too...

- Ruby

- The book "Programming Ruby" by Thomas and Hunt is, I think, the standard tutorial/reference book.
- The first edition of this book is mostly online:
<http://www.rubycentral.com/book/index.htm>
Note: the online chapters are not complete, but it's still quite useful.
- There is now (as of Oct 2004) a 2nd edition available.

Other resources

- You can look at some of my initialization files (in `/cs/whuang`):
 - `.emacs`
Emacs initialization file loaded every time emacs starts.
 - `.bashrc`
BASH initialization file loaded every time a new shell starts.
My convention is to keep system-specific things in my `.bashrc` such as environment variable definitions.
 - `.alias`
I source this file from my `.bashrc`. It contains stuff such as aliases and functions that I want on all my systems (e.g., SunOS at RPI, linux at home, and for cygwin (and linux) on my laptop)