Today's Class

- Recap from last time
  - Emacs
- Awk
  - Practice exercises
- Sed

Why use a text editor?

- Plain text (rather than formatted text)
  - No hidden characters - readable by another program
  - Files are portable (won't become obsolete)

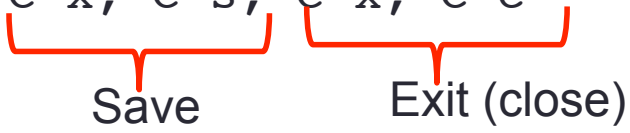
- Many features to make writing code more efficient
  - Syntax highlighting
  - Indentation
  - … and more… prevents bugs

A continuum:

Text editors  IDEs

Getting Started in Emacs

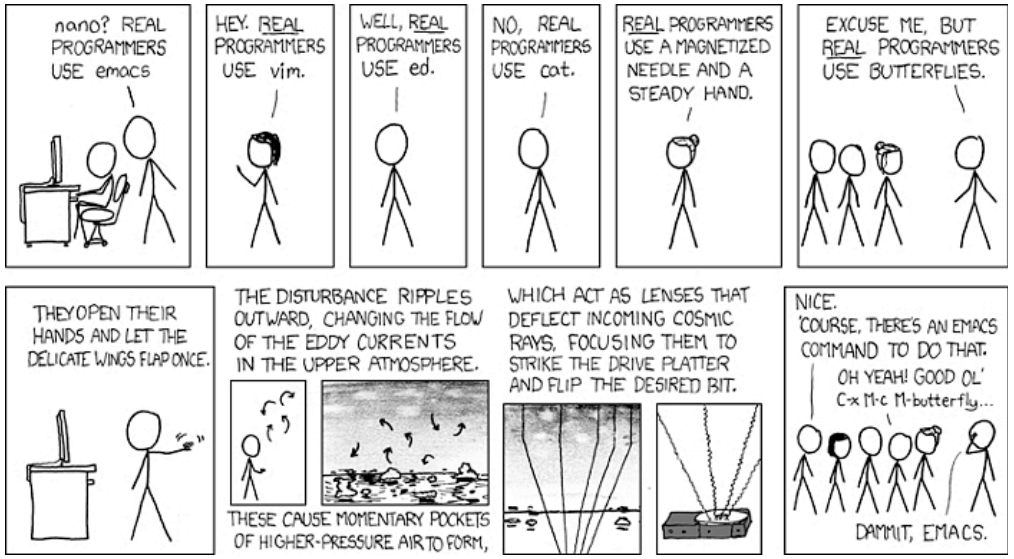
- Start editing a file by typing
  - `$ emacs filename`
- Save your work and exit
  - `C-x, C-s, C-x, C-c`



- Many of the keystrokes we learned for the command-line also work in Emacs (C-a, C-e, etc.)

2,000+ commands!

Nearly all commands involve holding down the **Ctrl key (“C”)** or the **Meta key (Alt key, “M”)** while typing one other letter or symbol



Real Programmers, http://xkcd.com/378/

Add your own key-bindings in your .emacs file

But you really only need a few to get by…

Useful Emacs Commands

Key Combination	Result
C-a	**Beginning of line**
C-e	**End of line**
C-k	**Kill line from cursor**
C-d	**Delete**
C-v	Scroll down
M-<	Go to beginning of document
M->	Go to end of document

Key Combination	Result
C-g	Quit
C-x, C-s	Save
C-x, C-c	Exit (‘Close’)
C-_	Undo
C-s	Search forward
C-r	Search reverse
M-x goto-line (M-g g in Emacs v. 23)	Go to line
M-x replace-string	Find and replace

Boldface indicate keystrokes that also work on the UNIX command line

Emacs = editing with macros

- One of most powerful utilities in emacs are its macros
  - Quickly do repetitive tasks that might otherwise require writing a program (and might be more complicated to do without making any keystrokes)

Steps to recording a macro:

C-x (→ starting recording

C-x) → end recording

C-x e → execute the macro

C-u 20, C-x e → repeat 20 times, execute the macro

Example 1: Replace DDB_G with ddb_g for every line in dd_dev_transcriptome.txt)

Example 2: Split “locus” in compute files into two columns

Emacs, find and replace

- M-x replace-string
- M-x replace-regexp

  “Find” must follow this format: \(My_Pattern\)

  My pattern can also be spaces, digits, etc.

  [[:space:]]

  [[:digit]]

  Tab is Ctrl-q TAB

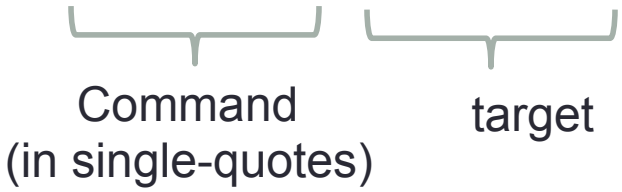
Awk

- Interpreted programming language for extracting and transforming text -- i.e., processing text files
- Standard on most UNIX systems
- Created in Bell Lab's in the 1970's
  - Named for its authors (Aho,Weinberger, and Kernigan)
  - Influenced the development of Perl
- Most useful for "one-liners"
  - But it is capable of doing everything other languages do

Awk basics

- From the command line:

```
$ awk '$1>20' filename
```



- Awk commands have the following syntax:

```
condition { action }
```

  - If no action is specified, prints the entire line
  - $1,$2, $3… refer to column1, column2, column3, etc.
  - $0 refers to the entire line

```
$ awk '$1>20 {print $5} ' filename
```

(for all lines where column 1 > 20, print column 5)

Awk basics

- The ‘condition’ can be a pattern-match *(aka regular expression)*:

```
awk '$2~/act/ && $4 > 30' dd_dev_transcriptome.txt
```

(prints all lines where column 2 matches “act” and column 4 is greater than 30)

- You can perform substitutions:

```
awk '{gsub(/DDB_G0/, "ddb0");print}' dd_dev_transcriptome.txt
```

(replaces DDB_G0 with ddb0)

- You can do calculations:

```
awk '{sum+=$3} END {print sum/NR}' dd_dev_transcriptome.txt
```

(calculates mean of column 3)

NF = Number of Fields (=columns)
NR = Number of Records (=rows)

BEGIN = Do prior to reading file
END = After reading file

Awk basics

- Combine with UNIX commands

```
awk '$3>30'filename | sort -n -k4 | tail -10 > outfile
```

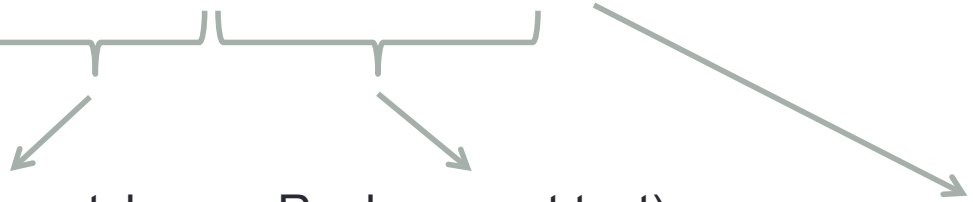
(Extract all rows where column 3 is greater than 30, sort them by column 4, take the 10 highest values, and save the results to outfile)

Sed (stream editor)

- UNIX utility that parses and transforms text
- Developed at Bell Labs
  - One of the first tools for regular expressions (aka pattern matching)
  - grep, sed and AWK developed from “ed” (early editor) and were the inspiration for Perl
    - All of these are notable for their editing of strings
      - Influenced the syntax

Example sed command - substitute

```
$ sed 's/regexp/replacement/g' inFileName > outFileName
```



Text to match (exact or regular expression)

Replacement text)

If nothing: defaults to first Instance

g: global (all matches)
1: first match
2: second match
Etc

Try this example:

```
sed 's/DDB_G/ddb_g/g' dd_dev_transcriptome.txt > dd_dev_transcriptome.txt2
```

Homework

- Awk practice exercises

- Try a sed tutorial:

  http://www.panix.com/~elflord/unix/sed.html

  http://www.catonmat.net/blog/worlds-best-introduction-to-sed/

  https://www.digitalocean.com/community/tutorials/the-basics-of-using-the-sed-stream-editor-to-manipulate-text-in-linux