**FreeGeek Chicago 02/2013**

**SCOPE:** The purpose of this seminar is to lead you to faster better editing of basic text files. These skills translate to HTML web documents, program coding, config files, and host of other types of computer files.

**Basic File Editing**

What is a regular expression?

A regular expression is a set of text characters and/or meta-characters used to match a pattern(s) in a body of text, or stream of characters.

I won’t be covering all the meta-characters and regular expressions you can use, but I will be giving you the basics, so you can edit files easily, safely, and quickly.

Basic Regular Expressions for matching patterns.

I Syntax and Metacharacters ( .\* vs .\*? )

II Character Classes

III Operators

IV Anchors

V Variables

Locating and changing text in vi.

Perl One Liners (How to do mass editing quickly and easily?)

Bash Shell Command Line Editing.

Resources

Regular Expressions (Regex)

Google the search string “Regex Primer”

<http://marvin.cs.uidaho.edu/Teaching/CS445/regex.html>

<http://content.hccfl.edu/pollock/ShScript/RegExp.htm>

Regular Expression Generator (RegEx)

<http://txt2re.com/>

Perl Resources

man perl

perldoc perreref

perldoc perlcheat

perldoc perlfaq6 (Regexes)

Perl One Liners

<http://www.softpanorama.info/Scripting/Perlorama/perl_one_liners.shtml>

<http://www.math.harvard.edu/computing/perl/oneliners.txt>

If you want to check it out of the library or buy this book,

*Learning Perl, 5th Ed*   
 Covers perl version 5.10.1  
July 2008  
by O’Reilly Media   
Authors: Randal L. Schwartz, Tom Phoenix, & brian d foy.

WARNING: Don’t download the free PDF version, many of these web sites have malware.

Free Online Books

<http://www.perl.org/books/library.html>

Differences between perl 5.8.8 and perl 5.10.1

<http://search.cpan.org/~rgarcia/perl/pod/perl5100delta.pod>

Bash Shell Command Line Editing

<http://www.math.utah.edu/docs/info/features_7.html>

<http://www.talug.org/events/20030709/cmdline_history.html>

**C stands for the Control Key M stands for the Alt key**

C-d Delete the character underneath the cursor.

C-\_ Undo the last thing that you did. You can undo all the way back to an empty line.

C-a Move to the start of the line.

C-e Move to the end of the line.

M-f Move forward a word.

M-b Move backward a word.

C-l Clear the screen, reprinting the current line at the top.

Here is the list of commands for killing text.

C-k Kill the text from the current cursor position to the end of the line.

M-d Kill from the cursor to the end of the current word, or if between words, to the end of the next word.

M-DEL Kill from the cursor the start of the previous word, or if between words, to the start of the previous word.

C-w Kill from the cursor to the previous whitespace. This is different than M-DEL because the word boundaries differ.

And, here is how to **yank** the text back into the line. Yanking means to copy the most-recently-killed text from the kill buffer.

C-y Yank the most recently killed text back into the buffer at the cursor.

M-y Rotate the kill-ring, and yank the new top. You can only do this if the prior command is C-y or M-y.

USING HISTORY

Okay, well say you didn't *just* run the sought-after command. You know you've used it within the past few days, but you don't want to scroll through what could be hundreds of commands to find it. Well, there are a couple of ways to do this, depending on how much you can remember about the history yourself...

* **special knowledge**
  + If you know you haven't executed any commands with the same starting letter sequence since then, you can just use the built-in Bash history expansion command **!** followed by the first few letters of the command.
  + The unique string doesn't have to be at the start of the command. You can use the more flexible built-in Bash history expansion command **!?** followed by a unique string appearing anywhere in the command.
  + Both of these commands will immediately recall *and execute* the most recent matching command. Thus, it is usually not a good idea to use these methods with destructive commands like **rm**!
* **some knowledge**
  + If you aren't positively sure of what would happen if you were to use the **!** or **!?** method, or if you need to search for something more unique in the command than the first few letters can provide, then you could use the history search feature.
  + Before you begin typing your command, type **ctrl-r**. This will put you into history search mode (actually, reverse incremental history search mode).
  + Now when you begin typing, the most recent command matching what you've typed so far will appear on the line with a cursor at the start of the match. (Try playing around with this feature; there are a few interesting behaviors in there.)
  + When you've found what you're looking for, you have a couple of options. Just pressing Enter will immediately recall and execute the command. **ctrl-j** or Escape will retrieve the command, but allow you to continue editing. If you can't find what you're looking for or if you just change your mind, hit **ctrl-g** or **ctrl-c** to cancel.
* **vague memory**
  + If you really are uncertain of the history or if you know you could be searching back through many similar commands for one of particular interest, then you can use this more brute-force method.
  + Type the following command to get a list of all related commands with their history numbers:

**history | grep -i "*<search string>*"**

* + Once you've found the command you want, you can execute it specifically by its number using the following built-in history expansion command:

**!*<history number>***

**Remember**:  Once you've used one of these methods to recall and execute a command, that command is now the most recent command in your history. You can now just press the up arrow once to retrieve it again.