Homework 1: Text Editors

Due September 18, 11:59 pm

This homework will give you some exposure to the text editors vim and emacs. When doing these exercises, please use the commands on the reference sheets at

`www.stat.wisc.edu/~jgillett/605/emacs/emacs.html`

and

`https://vim.rtorr.com/`

The goal is not simply to get the specified tasks done, but rather to do so using the functionality available in emacs and vim, so try do the tasks elegantly, using the tools provided by the editors.

You may ask your classmates, TA or instructors for help with these exercises. If you collaborate with classmates, please include their names and their primary `@wisc.edu` email addresses on the line after your name in your submission files.

1 Editing text in emacs

This problem will give you some practice using emacs to edit text.

1. Start emacs on your Linux virtual machine.
2. Use emacs to make a new directory, `1` (for "homework 1"), in your `~/Desktop` directory. That is, make the new directory `~/Desktop/1`.
3. Download `baby_T_Test.R` and move it to your `1` directory.
   (a) Read `baby_T_Test.R` into emacs.
   (b) The code has four problems with parentheses. Indent the buffer. The first indenting irregularity indicates a parentheses problem on the previous line. Fix it by adding a parenthesis. Repeat until the whole file is indented correctly.
   (c) Save the file as `baby_t_test.R` (note "`T_Test`" changed to "`t_test`").
   (d) Make these replacements in `baby_t_test.R`.

- Inside the `baby.t.test()` function, replace each `r` (where `r` indicates the list being set up as a return value) with `return.list`.
- Replace each occurrence of three newlines with two newlines. Hint: We can't use `Enter` to type a newline in the minibuffer, as it ends the minibuffer's input. Use `C-q C-j` to type a literal newline into the minibuffer. `C-q` runs `quoted-insert`, which allows inserting a literal newline in linux (newline is `C-j`, the decimal ASCII code 10), a control character, etc. There are still sequences of three newlines remaining, so jump to the top of the buffer and do it again. (We'll see a way to do these two steps in one step after we study regular expressions soon).

(e) Use the emacs R buffer to:

i. Run the chunk of code consisting of the `baby.t.test()` function definition.

ii. Run the "test case" code one line at a time. Note that the `p.value` test fails. The bug is that I mistakenly used `df=n` instead of `df=n-1` in my call to `pt()` in `baby.t.test()`. Fix this bug, run the `baby.t.test()` chunk again, and run the test code (one line at a time) again.

iii. Run the entire buffer.

(f) Kill the R buffer. Start it again.

i. Find `3 + 4` (just type it at the R command prompt).

ii. Run `source("baby_t_test.R")` (just type it at the R command prompt).

(g) Kill the R buffer again and exit emacs.

4. Download `gettysburg1.txt` and `gettysburg2.txt` to your `1` directory.

(a) Use Multiple Windows (in a single emacs session) to split emacs vertically (into halves) and vertically again (now you have one half and two quarters).

(b) Resize the windows so they all have the same size. (Oops, I forgot to demonstrate resizing windows, but it's on the reference sheet under Multiple Windows: `C-x ^` will make the current subwindow taller. Do it a few times in each of the quarter-sized windows until you have three one-third-sized windows.)

(c) Open `gettysburg1.txt` in the first window and `gettysburg2.txt` in the second. Open (the new empty file) `gettysburg_emacs.txt` in the third window.

(d) Use cut, copy, and paste commands to reassemble the Gettysburg Address from its segments in the first two windows into a whole in the third window. Handle each "paragraph" in the downloaded files separately (e.g. lines 1-2 are a paragraph, as is line 3, as are lines 4-6). Save `gettysburg_emacs.txt` when you are done. Close its buffer.

(e) Return to a single window. Open the Directory Editor on `.` ("dot"), the current directory.

i. Rename `gettysburg_emacs.txt` to `GettysburgAddress_emacs.txt`.

ii. Find (open) `GettysburgAddress_emacs.txt` again.

(f) Kill the rectangle consisting of the line numbers and spaces preceding each line of the address. Save `GettysburgAddress_emacs.txt` again.

2 Editing text in vim

This problem will give you more experience using vim, and will show you a few useful tricks for getting things done quickly.

1. Make sure `gettysburg1.txt` and `gettysburg2.txt` are still in your `1` directory (download them again, if not).

   (a) Use vim's windowing system to split the screen horizontally once, and then vertically once (we didn't discuss this latter command in the videos– `:split` splits horizontally; `:vsplit` splits vertically). Your screen should be split into one half and two quarters.

   (b) Open `gettysburg1.txt` in the first window and `gettysburg2.txt` in the second. Open (the new empty file) `gettysburg_vim.txt` in the third window.

   (c) Use cut (`dd`), copy (`yy`), and paste (`p` or `P`) commands to reassemble the Gettysburg Address from its segments in the first two windows into a whole in the third window. Handle each "paragraph" in the downloaded files separately (e.g. lines 1-2 are a paragraph, as is line 3, as are lines 4-6). Save `gettysburg_vim.txt` when you are done and close the windows. Return to the command line and do the following (or try using vim's `:!shell_cmd` to run the command `shell_cmd` without leaving vim):

      i. Rename `gettysburg_vim.txt` to `GettysburgAddress_vim.txt`.

      ii. Remove `gettysburg1.txt` and `gettysburg2.txt`.

      iii. Open `GettysburgAddress_vim.txt` in vim again.

   (d) Use visual block mode to delete the line numbers and spaces preceding each line of the address. Save `GettysburgAddress_vim.txt` again.

   (e) Exit vim.

2. Download the file `semicolons.c` from the course webpage and save it to your `1` directory. Open it in vim.

   (a) `semicolons.c` is a snippet of C code. Don't worry if you've never coded in C before! All you need to know for this excise is that every line of C needs to end in a semicolon (`;`). So, for the code in `semicolons.c` to run, we need to put a semicolon at the end of each line. Go to the first line, type `A` to enter insert mode at the end of the line, type a semicolon (`;`), and press escape (`<Esc>`) to return to normal mode.

   (b) Now, we want to perform this same operation on every line of `semicolons.c`. The period (`.`) repeats the last operation we performed. Move the cursor to the second line and press (`.`). The result *should* be that the second line now has a semicolon at the end, too.

   (c) We *could* put a semicolon at the end of every line by moving the cursor to each line, one at a time, and pressing `.`, but there's a better way. We can run a

command on multiple lines by selecting them in visual line mode, and then telling vim to execute the `.` command.

1) Put your cursor on the third line, then type `V` to enter visual line mode.
2) Highlight from line 3 to the end of the file (a shortcut for this is to type `G`).
3) Type `:'<,'>normal .` and hit enter. This tells vim to take each highlighted line, and call the normal-mode command `.`.

(d) The most typical formula for editing in vim is to combine an operator (e.g., `d`, `x` or `y`) with a motion (i.e., specifying a location, such as the next beginning of a word or the end of the current line). We saw the `dd` command in lecture, but the `d` command has a lot more options available. For example, we can write `dw` to delete from the cursor to the next start of a word, or `d^` to delete from the cursor to the beginning of the line. An especially useful command is the `daw` command. It deletes the current word under the cursor (it's easy to remember– `daw` stands for "delete a word"). Use the `daw` command to delete the word `approximately` from the last line of `semicolons.c`.

(e) Type the command `:wq` to save your changes and quit vim.

3. Download the file `hw_snippet.tex` to your `1` directory and open it in vim.

(a) `.tex` is the standard extension for LaTeX, a markdown language for creating pdfs. `hw_snippet.tex` contains the LaTeX code for a snippet of this homework assignment. I make a habit of putting lots of new lines (i.e., the "return" or "enter" key) in my `.tex` files, but some people find this annoying. You can get rid of linebreaks with the `J` command. Press `J` once and you'll notice that the newline at the end of the current line is deleted, and the text on the next line is "joined" to the current line.

(b) We could hit `J` repeatedly to join all the lines into one, but instead let's use visual line mode again to apply the `J` command to every line. Highlight every line of the file using visual line mode, and then type `J`. The result should be that all the text is joined onto one line. Note that there is some possibility for confusion, here. When we talk about a *line* in programming, we mean a length of text, ended by a new line (i.e., the "return" or "enter" key). This is in slight contrast with lines in your terminal. By default, your terminal will only display 80 characters per line, and a line that is longer than 80 characters will be displayed on multiple lines in the terminal (you may see this described as *wrapped* text).

(c) Now, let's put together a few of the things we saw above. Recall from lecture the search command `/XYZ`, which searches for the next occurrence of the string `XYZ`. Search for the string `the` in the text: type `/the` and hit enter. This will move the cursor to the next occurrence of the string `the`. Type the command `daw` to delete it. Type `n` to move to the next occurrence of `the` and type `.` to repeat the previous action (i.e., delete the current word). Proceed in this manner to delete all instances of the word `the` from `hw_snippet.tex`. Later in

the course, we'll see some more graceful tools for deleting text and performing find-and-replace operations.

(d) Use the command `:wq` to save your file and quit.

3 Prepare your files for submission

Okay, let's get your files ready to submit HW1. Start emacs or vim and open the (new empty) file `hw1.txt`.

1. Type your full name, followed by your primary `@wisc.edu` email address.
2. If you discussed this homework with any of your classmates, please list their names and primary `@wisc.edu` email addresses on the next line.
3. Press the "return" key three times to insert two blank lines.
4. Insert your `baby_t_test.R` file.
5. Type two blank lines.
6. Insert your `GettysburgAddress_emacs.txt` file.
7. Type two blank lines.
8. Insert your `GettysburgAddress_vim.txt` file.
9. Type two blank lines.
10. Insert your `semicolons.txt` file.
11. Type two blank lines.
12. Insert your `hw_swnippet.tex` file.
13. Search online to find something interesting emacs can do. Summarize it briefly (no more than 100 words) for consideration by the class for a demonstration.
14. Write a short emacs exercise (no more than 100 words) for consideration by the class for a quiz or homework.
15. Search online to find something interesting vim can do. Summarize it briefly (no more than 100 words) for consideration by the class for a demonstration.
16. Write a short vim exercise (no more than 100 words) for consideration by the class for a quiz or homework.
17. Save `hw1.txt` and exit emacs.

Okay, that's it– great work! Upload your `hw1.txt` file via the Canvas "HW1" grades item.