SVN and Elisp. ITWS-2 Lab, Jan Week 3

itws2 instructors 2012-Spring, IIIT HYDERABAD

January 17, 2012

Contents

1 Introduction

In this lab you will get a basic introduction to SVN. It is **very important** that you are comfortable with svn because you will use it heavily in this course (for future assignments etc.). You will also learn a bit of elisp which will help you configure basic options in your `.emacs` file.

2 Subversion

Subversion (or SVN for short), is a tool which helps you maintain all your documents, codes etc. What do I mean by **maintain** ?

- SVN keeps all your data in one central location (called a repository)
- You can decide who can get access to this repository.
- You can "put" multiple documents (organized in a directory structure) into this repository.
- You can have multiple "versions" of a single document.
  - You write a code named `sort.c` and put it in your repository. Your friend has access to this repository. He claims that he has improved your `sort.c` program and has "put" the updated version

in the repository. Now you can check how his version of `sort.c` differs from your original code. You can further improve on it, and "put" your updated version in. In this way, both of you can collaborate on developing a great `sort.c` code!

2.1 Setting up a repository

We will now try to make a SVN repository.

2.1.1 Create an empty repository

Just `cd` to the location you want to keep your repository in. We now use the `svnadmin` command.

```
svnadmin create <repository-name>
```

Replace `<repository-name>` with the name of your repository. In this writeup, let's assume the name is `myrepo` and its path is `/home/user/work/myrepo` Running the above command creates a folder with the name of your repository.

2.2 Using a local repository

2.2.1 Checking out a file

The SVN repository stores all your files. In order for you to "retrieve" these files, you can use the following command.

```
svn checkout file://<path-to-your-repo>
```

In our case we will use

```
svn checkout file:///home/user/work/myrepo
```

This will create a folder named `myrepo` in your current directory. This directory is called a "working copy" of your repository.

2.2.2 Committing a new file or directory

Create a new file in your `myrepo` directory. You will want to “commit” it into your repository, so that further “check-outs” of the repository contain this file. Whenever you create a new file you cannot directly commit it. You need to do it in two steps.

```
svn add <path-to-new-file>
svn commit -m "<your-message>"
```

This adds directories too (recursively).

2.2.3 Committing an updated version

You can modify any file in your `myrepo` directory. To add the latest version just use the following command from your working copy.

```
svn commit -m "<your-message>"
```

2.2.4 Status of your working copy

To check the status of files in your working copy use the following

```
svn status
```

To check status of a particular file in your working copy, use

```
svn status <filename>
```

The output is a list of files with the following characters for each file. Use this link to understand the output http://ldc.usb.ve/docs/svn/svn.ref.svn.c.status.html

2.2.5 Updating a working copy

To update your working copy to the latest “revision” of your SVN repository, use the following in your working copy

```
svn update
```

To update to revision number 2

```
svn update -r 2
```

2.2.6 Checking logs

You can see a log of all the revisions that have been made on your working copy by using the command `svn log` in your working directory.

2.3 Using a repository on a remote machine

The commands for `status, commit, update` remain the same.

2.3.1 Checking out

If your repository is on a remote machine (like a server) then you need to use slightly modified commands.

```
svn checkout svn+ssh://username@hostname/<path-to-remote-repository>
```

As an example, say I created a repository on my `mirage` account. The repository is located at `/home/user/work/genetics`

```
svn checkout svn+ssh://user@mirage.iiit.ac.in/home/user/work/genetics
```

2.4 A little note on "revisions"

SVN revisions indicate the "version number" of your repository as a whole. It may be that a certain file inside your working copy has been changed, but others have not. If you `commit` this working copy, the `revision` of your repository will increase by 1. This is in spite of the fact that only one file has changed. Hence its always recommended to add a useful message using the `svn commit -m` command, so that you know what changes were made in a particular revision.

3 Elisp (a.k.a. Understanding/Editing the `.emacs`)

> "GNU Emacs is an extensible, customizable text editor—and more. At its core is an interpreter for Emacs Lisp, a dialect of the Lisp programming language with extensions to support text editing." [1]

[1]The very first line on the GNU Emacs Homepage

Customizability is one of the cornerstones of Emacs. Infact, one of the things you will realize with some experience is that customizing Emacs to behave the way you want it to is a very important part of boosting your productivity with Emacs.

Our goal here (in this handout) is *not* to teach you how to program in Emacs Lisp (Elisp) but to give you enough background to be able to customize your copy of Emacs effectively.

3.1 The `.emacs`

Wass dis dawt emacs thing ya been yappin' on about? [2]

The `.emacs` is your personal Emacs initialization file. It contains lines of Elisp code that tell Emacs to do certain things on startup. Some of the common things you'll be asking Emacs to do are :

Set the values of Variables The values of these variables, in turn, affect some or the other aspect of Emacs' behaviour.

Define your own Variables Occasionally you may need to define your own variables for use in your Elisp statements.

Enable/Disable certain Modes Emacs automagically loads certain *modes* depending on what sort of editing it sees you doing. Sometimes you want to change its default behaviour by adding/removing certain modes in a given situation.

Trigger some action based on an Event This may sound cryptic, but it isn't. An *event* could be something very simple like the opening of a file of a certain kind (mostly identified by the file's extension), or some keystroke/key-combination you press. An *action* is what you want Emacs to do in response.

All these things, and a lot more, can be done by adding appropriate lines to the `.emacs`.

Some noteworthy points:

- Your `.emacs` should be in your home directory (path: `~/.emacs`).

[2] probably what **you** are thinking right now.

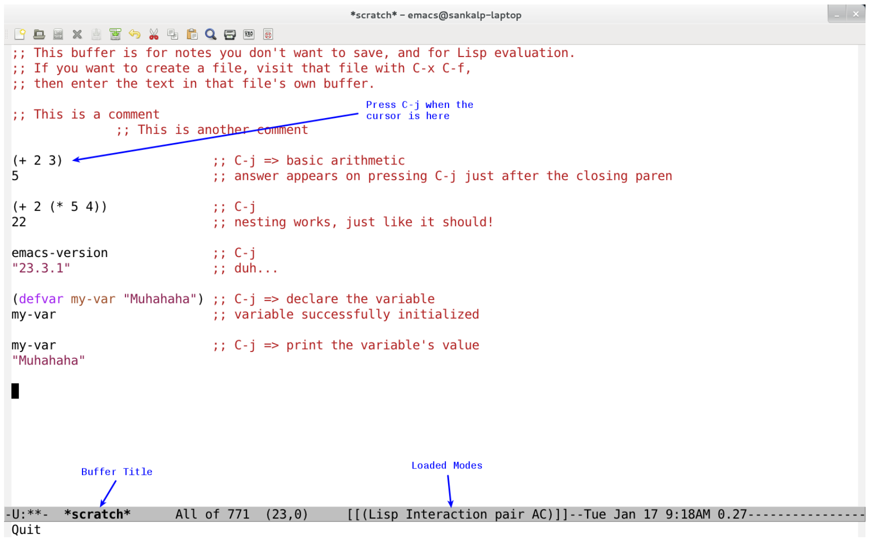
- For changes in the `.emacs` file to take effect, you need to restart Emacs or type `M-x eval-buffer <RET>` (in the `.emacs` buffer)

3.2 The `*scratch*` buffer

The `*scratch*` buffer is the place where you can test out Elisp code. How?

1. Put (type/paste) the code you want to evaluate in the `*scratch*` buffer
2. Press `C-j` to evaluate the Elisp expression immediately prior to the cursor.

Try out the stuff shown in the Figure.



scratch – emacs@sankalp-laptop
Press C-j when the cursor is here
Buffer Title
Loaded Modes

```
;; This buffer is for notes you don't want to save, and for Lisp evaluation.
;; If you want to create a file, visit that file with C-x C-f,
;; then enter the text in that file's own buffer.

;; This is a comment
              ;; This is another comment

(+ 2 3)                    ;; C-j => basic arithmetic
5                          ;; answer appears on pressing C-j just after the closing paren

(+ 2 (* 5 4))              ;; C-j
22                         ;; nesting works, just like it should!

emacs-version              ;; C-j
"23.3.1"                   ;; duh...

(defvar my-var "Muhahaha") ;; C-j => declare the variable
my-var                     ;; variable successfully initialized

my-var                     ;; C-j => print the variable's value
"Muhahaha"

-U:**-  *scratch*      All of 771  (23,0)     [[(Lisp Interaction pair AC)]]--Tue Jan 17 9:18AM 0.27----------------
Quit
```

Figure 1: The Scratch Buffer

Any code that you want to put into your `.emacs` can be run/evaluated in the `*scratch*` buffer first, to see if it behaves the way you expect (the meaning of this will become clearer as you read on).

3.3 I've read enough. Show me some of the real stuff!

The title says it all. We'll now go over some examples of how Elisp code (in your `.emacs` OR `*scratch* buffer`) can be used to do cool things.

(Please try to go through all the examples, as Elisp concepts have been explained as and when needed)

3.3.1 Display Time and Date in the mode line

By default Emacs does not show the current time/date. You can make it do so using the following snippet of Elisp code

```
;; Display time with day and date
(setq display-time-day-and-date t)
```

As you might have figured, `display-time-day-and-date` is an Emacs internal variable, whose value you are setting to `t` (true) using the function `setq`.

Note that the syntax for function calls in Lisp (not just Elisp) is different from what you knew in `C`.

```
(function args ...) ;; Lisp function call syntax
```

As mentioned earlier, you can either put these lines in your `.emacs` for them to be persistent across Emacs restarts, or simply try them out by evaluating them in the `*scratch*` buffer.

Question : What kind of a Variable do you think `display-time-day-and-date` is? Is it a Boolean? or is it something slightly different?

Hint : Without making the above customization, go to the `*scratch*` buffer and find out the default value of `display-time-day-and-date`.

3.3.2 Turn on the Visible Bell

Remember that irritating sound you hear when a certian Operating System (that which must not be named!) pops up an error dialog box? (those of you who have used computers before coming here should have some unpleasant memories associated with that).

Well, Emacs doesn't make sounds. At least on my machine it doesn't. So, to be fair to Emacs, lets give it permission to make the screen blink in a peculiar way, to indicate that we're doing something that doesn't make sense.

```
;; turn on visual bell
(setq visible-bell t)
```

After you've set this, try scrolling up/down beyond the boundaries of your buffer, or `M-x crucio <RET>` (see, it's a crime to do that in Emacs too!).

3.3.3 Customizing the Colours of your Emacs Display

Lots of us like a dark terminal, with white/gray text. Now before you go about dismissing Emacs because of it's default light theme, try out the following customizations:

```
;; Colour customizations
(set-background-color "black")
(set-foreground-color "white")
(set-cursor-color "white")
(set-mouse-color "white")
```

Pretty neat, huh? (try other colours!)

Notice that here we're not setting the value of a variable, instead we're making function calls with certain arguments (these functions are part of the Emacs source code).

3.3.4 Adjusting the default font size

We all know that the font size in an Emacs buffer can be increased or decreased using `C-x C-+` (the plus button next to `Backspace`) and `C-x C--` (the minus button next to the plus button mentioned).

But how about setting the font size globally at startup?

```
;; font size
(set-face-attribute 'default nil :height 180)
```

Keeping the other function parameters unchanged, you can adjust the value of the last parameter as per your needs to get the desired font size.

3.3.5 Display the line and column number in the mode line

```
;; display line and column number in "status bar" (mode line)
(line-number-mode 1)
(column-number-mode 1)
```

This is an example of enabling a mode (two modes, infact). To find out what possible values the arguments here can take, and what those values will result in, you can do a `M-x describe-function <RET> line-number-mode <RET>`.

This opens up a frame with the necessary information.

The `describe-function` function is quite handy, and can be used to pull up information about any function. Try it! (It is also mapped to `C-h f`).

3.3.6 Displaying the line number next to each line

Missing the "set nu" feature from that other Editor?

Fret not. Customizations to the rescue!

```
;; set nu
(global-linum-mode)
```

This was an example of a function that takes no arguments. Try `C-j` ing it a couple of times in the `*scratch*` buffer.

(*Never forget the parentheses!* – ancient Lisp wisdom)

3.3.7 Syntax Highlighting

You can control whether Emacs performs syntax highlighting or not.

```
;; Enable syntax highlighting for older Emacsen that have it off
(global-font-lock-mode t)
```

Naturally there must be a way to turn it back off. Investigate!

3.3.8 Make the Menu Bar go away

When you feel you're an *Emacs Ninja*, you might want to reduce the clutter by doing away with the Menu Bar.

When that day comes, you should remember *this* day as when you learnt how to do it :)

```
;; No Menu Bar
(menu-bar-mode -1)
```

3.3.9 Highlight matching parentheses

Enable `show-paren-mode` (Hint: Look at how we enabled other modes)

3.3.10 Custom Keybindings

You can set keybindings for commonly used functions (those that you invoke otherwise with `M-x`)

```
;;set the keybinding so that f3 will start the shell
(global-set-key [f3] 'shell)
```

Once this is set, pressing `F3` is the same as `M-x shell`.

3.3.11 Allow `y` or `n` for most choice prompts

I hate having to type `yes` or `no` in places where a simple `y` or `n` should suffice.

```
;; allow typing y/n instead of yes/no in most cases
(fset 'yes-or-no-p 'y-or-n-p)
```

This example illustrates that you need not understand each and every line of what goes into your `.emacs`. Most of the times you just copy-paste what you find on the internet – And it works! :D

3.3.12 Word Wrap

You might want to enable automatic line-wrapping (often called Word Wrap). To do that, use the following code :

```
;; auto-fill mode
(add-hook 'text-mode-hook 'turn-on-auto-fill)
(setq-default fill-column 90) ;; widescreen ftw
```

`fill-column` decides the number of characters after which the line gets wrapped. You can alter it's value as per your screen width.

This code snippet introduces the concept of a `hook`.

Hooks are seen in many software systems, notably Emacs and svn which are part of our course. A hook is a construct which allows you to trigger actions (Imagine a hook as a literal hook, into which you attach things). Picture a hook attached to `text-mode` in Emacs. All things attached to this hook get executed when `text-mode` is loaded for any buffer.

3.3.13 Auto-Maximizing Emacs on startup

This is the example which demonstrates how even functions can be defined in your `.emacs`

```
;; For full screen on starting Emacs -- Very Handy
;; taken from Prof.Choppella's dotemacs
(defun toggle-fullscreen ()
  (interactive)
  (x-send-client-message nil 0 nil "_NET_WM_STATE" 32
                         '(2 "_NET_WM_STATE_MAXIMIZED_VERT" 0))
  (x-send-client-message nil 0 nil "_NET_WM_STATE" 32
                         '(2 "_NET_WM_STATE_MAXIMIZED_HORZ" 0)))
(toggle-fullscreen)
```

You need not fully understand what is going on, except that the function `toggle-fullscreen` was defined, and a call was made to it immediately after the definition.

3.4 Concluding remarks

We have barely scratched the surface of what *can* be done, but this should whet your appetite and get you started with tweaking your own `.emacs` as per your whims and fantasies.

You can consult the references at the end of this document to read about these things in more detail.

4 Lab Evaluation

Lab evaluation is for a total of 20 marks. 10 marks for the SVN part, and 10 for the `.emacs` part.

4.1 SVN

TAs should grade the students if they successfully complete the following tasks.

1. Creating a local repository (1 mark)
2. Using `svn checkout`. (2 marks)
3. Using `svn add` and `svn commit -m`. (2 marks)
4. Understanding output of `svn status`. (3 marks)
5. Using `svn update -r` and `svn update`. (2 marks)

4.2 .emacs

Write Elisp lines to do the following:

1. Get rid of the Tool Bar (the one which contains the buttons for cut/copy/paste) (**1 Mark**)
2. Get rid of the Scroll Bars (*Emacs Ninja Extreme!*) (**1 Mark**)
3. Get rid of the Emacs Welcome Screen – Emacs should start with only the scratch buffer visible. (**2 Marks**)
4. Stop the Cursor from Blinking – *now that's customization!* (**2 Marks**)
5. Set things up so that pressing `F4` saves the file being edited (i.e. works the same as `C-x C-s`). (**2 Marks**)
6. Enable `flyspell` based spell-checking for `org-mode` (**2 Marks**)

Remember, Google is your friend!

5 References

Subversion

1. http://aymanh.com/subversion-a-quick-tutorial

Emacs

1. The Official GNU Emacs Homepage

2. Initialization File – The GNU Emacs Manual
3. Bool-Vectors in Emacs Lisp
4. Emacs @ Sethi.org Certain things, like “The Emacs Display” are quite well explained.
5. Customizing Emacs
6. Emacs Beginner’s HOWTO: Customizing Emacs
7. Hooks – The GNU Emacs Manual
8. Hooks, Emacs-Fu
9. Harry Potter Spell List
10. My .emacs setup [Link will be provided after the Lab :)]