1 **DONE** Commenting in org-files

There was an interesting discussion on the org-mode mail list about putting comments in org files. Eric Fraga suggested using inline tasks, and customizing the export of them so they make a footnote, or use the todonotes package (suggested by Marcin Borkowski). Here is Eric's export. A big advantage of this is integration with the Agenda, so you can see what there is todo in your document.

```
  (setq org-inlinetask-export-templates
        '((latex "%s\\footnote{%s\\\\ %s}\\marginpar{\\fbox{\\thefootnote}}"
                 '((unless
                       (eq todo "")
                     (format "\\fbox{\\textsc{%s%s}}" todo priority))
                   heading content))))
```

Eric Abrahamsen suggested an idea to use a link syntax. I like the idea a lot, so here we develop some ideas. A link has two parts, the path, and description. A simple comment would just be a simple link, probably in double square brackets so you can have spaces in your comment. It might be feasible to use the description to "mark text" that the comment refers to. The remaining question is what functionality should our link have when you click on it, and how to export it. For functionality, a click will show the comment in the minibuffer and offer to delete it. For export, for now we will make it export with todonotes in LaTeX, and as a red COMMENT with a tooltip in html. To use this, you need to have the LaTeX package todonotes included in your org file. Here is our comment link.

Why do you think there are only two parts

Why do you quote mark?

```
(org-add-link-type
 "comment"
 (lambda (linkstring)
   (let ((elm (org-element-context))
         (use-dialog-box nil))
     (when (y-or-n-p "Delete comment? ")
       (setf (buffer-substring
              (org-element-property :begin elm)
              (org-element-property :end elm))
             (cond
              ((org-element-property :contents-begin elm)
               (buffer-substring
                (org-element-property :contents-begin elm)
                (org-element-property :contents-end elm)))
              (t
               ""))))))
 (lambda (keyword desc format)
   (cond
    ((eq format 'html)
     (format "<font color=\"red\"><abbr title=\"%s\" color=\"red\">COMMENT</abbr></font> %s" keyword (or desc "")))
    ((eq format 'latex)
     (format "\\todo{%s}{%s}" keyword (or desc ""))))))
```

It would be convenient to have a quick function for adding a comment to some highlighted text.

```
(defun add-comment (begin end)
  (interactive "r")
  (if (region-active-p)
      (let ((selected-text (buffer-substring begin end)))
        (setf (buffer-substring begin end)
              (format "[[comment:%s][%s]]"
                      (read-input "Comment: ") selected-text)))
  (insert (format  "[[comment:%s]]" (read-input "Comment: ")))))
```

Test 1:

Test 2

test comment

You seem to have forgotten Test 2

That is it. I could see a few other enhancements that might be very useful, e.g. a command to list all the comments, remove all the comments, etc... I am pretty satisfied with this for now though.

1.1 An updated approach to comments.

Rainer asked about making some comments inline. It would be nice if a single link syntax could accommodate both styles of comments. I previously developed an approach to extend links with attributes (http://kitchingroup.cheme.cmu.edu/blog/2015/02/05/Extending-the-org-mode-link-syntax-with-a which I will reuse here for that purpose. The idea is to add an ":inline" attribute to change the export behavior. We only modify the LaTeX export here.

```
(org-add-link-type
 "comment"
 ;;  follow function
(lambda (linkstring)
   (let ((elm (org-element-context))
         (use-dialog-box nil))
     (when (y-or-n-p "Delete comment? ")
       (setf (buffer-substring
              (org-element-property :begin elm)
              (org-element-property :end elm))
             (cond
              ((org-element-property :contents-begin elm)
               (buffer-substring
                (org-element-property :contents-begin elm)
                (org-element-property :contents-end elm)))
              (t
               ""))))))
 ;; format function
 (lambda (path description format)
   (let* ((data (read (concat "(" path ")")))
          (head (car data))
          (plist (cdr data)))
     (cond
      ((eq format 'html)
       (format "<font color=\"red\"><abbr title=\"%s\" color=\"red\">COMMENT</abbr></font> %s" path (or description
      ((eq format 'latex)
       (format "\\todo%s{%s}%s"
               (if (-contains? data :inline) "[inline]" "")
               (mapconcat (lambda (s)
                            (format "%s" s))
                          (-remove-item :inline data) " ")
               (if description (format "{%s}" description) "")))))))
```

Here are some examples of the syntax:

`[[comment: :inline the rest of your text]]`

`[[comment:Some text you want to highlight]]`

`[[comment:Some text you want to highlight :inline]]`

It doesn't matter where the :inline attribute is added. This seems to work pretty well.

We can modify our convenience function to allow us to use a prefix arg to make the comment inline. Here is one way to do it.

```
(defun add-comment (begin end &optional arg)
  "Comment the region. With a prefix ARG, make the comment inline."
  (interactive (list (region-beginning)
                     (region-end)
                     current-prefix-arg))
  (let ((inline (if arg ":inline " "")))
        (if (region-active-p)
            (let ((selected-text (buffer-substring begin end)))
              (setf (buffer-substring begin end)
                    (format
                     "[[comment:%s%s][%s]]"
                     inline
                     (read-input "Comment: ") selected-text)))
          (insert (format
                   "[[comment:%s%s]]"
                   inline
                   (read-input "Comment: "))))))
```

`add-comment`

Test text to

an inline comment

a new regular comment

comment on.