org-change: Track changes in org-mode

Stefano Ghirlanda

April 21, 2023

1 Introduction

org-change is an Emacs minor mode that (ab)uses the org-mode link syntax for a simple "track changes" feature similar to some word processors. The main use case is for authors to highlight the changes between two versions of a document, such as before and after review by a third party. For this, org-change provides:

- A `change` link type to mark additions, deletions, and replacements.
- Functions and key bindings to manipulate `change` links.
- Export filters for `change` links (currently Latex and HTML).

2 Installation

Install from MELPA, or manually from here.

3 `change` link syntax

To indicate that "old text" is being replaced by "new text," org-change defines the following `change` link syntax:

```
[[change:old text][new text]]
```

The idea is that you end up seeing only "new text," because org-mode (typically) hides the part of the link within the first pair of brackets. To indicate an addition, org-change just omits `old text`:

```
[[change:][new text]]
```

To indicate a deletion, org-change uses `((DELETED))` as `new text`:

```
[[change:old text][((DELETED))]]
```

You can embed comments in change links by surrounding them with double stars at the end of `new text`:

```
[[change:old text][new text**A comment**]]
```

All this by itself is not very useful, but read on.

4 `change` link manipulation

org-change provides key sequences to easily manipulate `change` links. All key sequences start with `C-‘` (control + left quote). Not the prettiest, but few control prefixes are free. It's the curse of keymensionality. To change it, see section 6.

The key sequences are:

`C-‘ a` for additions.

`C-‘ d` for deletions.

`C-‘ r` for replacements.

All these act on the active region. For example, in the case of replacement, the region is marked for deletion, and you are prompted for new text. You can also use `C-‘ a` without marking a region, in which case you are prompted for new text.

Key sequences are also provided to accept or reject the change under the cursor:

`C-‘ k` to accept.

`C-‘ x` to reject.

"Accept" means to delete the change link (including any comments) and insert the new text (or nothing, if the change is a deletion). "Reject" means to delete the change link and insert the old text (or nothing, if the change is an addition).

If there is no change under the cursor, accept and reject work on all change links in the active region. If there is no active region, nothing happens. You can accept or reject all changes in a document by selecting the

whole buffer, but note that this deletes all changes. If you just want to export a clean manuscript, see section 5.3.

The following functionality is provided by org-mode, and is useful for `change` links:

`C-c C-l` lets you edit the link in the minibuffer. Because this is an org-mode function for all links, it will display the "old text" as `Link: change:old text` and the "new text" as `Description:  new text`.

`M-x org-toggle-link-display` toggles between showing and hiding the hidden part of every link in the buffer. This can be useful to work on longer edits.

5 Exporting

5.1 LaTeX export

When exporting to LaTeX, org-change uses the `changes` package, which it includes automatically in the exported document. org-change will then use the commands `\added`, `\deleted`, and `\replaced` provided by this package.

org-change supports some additional features of the `changes` package. It supports comments, so that

```
[[change:old text][new text**A comment**]]
```

is exported to

```
\replaced[comment=A comment]{new text}{old text}
```

You can also sneak in other fields supported by `changes` at the end of the comment. For example, you can indicate the author of the comment:

```
[[change:old text][new text**My comment,author=SG**]]
```

which is exported to:

```
\replaced[comment=My comment,author=SG]{new text}{old text}
```

Lastly, you can set options for the `changes` package by setting the variable `org-change-latex-options`. For example, you can place this code somewhere in your document and evaluate it:

```
#+begin_src elisp
  (setq org-change-latex-options "[markup=underline]")
#+end_src
```

Note that you need to include the brackets. The `changes` package also has configurations that are not set through package options, which you can set through `#+latex_header:` lines.

The `changes` package causes errors with some LaTeX commands. This can happen, for example, when `\cite` and similar commands appear in a change. To fix these problems, you can try to add `\protect` or `\noexpand` before the offending command, or to wrap the command in an `\mbox`.

5.2 HTML export

When exporting to HTML, org-change produces `<span>` elements with classes `org-change-added`, `org-change-deleted`, and `org-change-comment`. A replace link has both an added and a deleted span, while add and delete links only have one span. The comment span is embedded in the add span when present, otherwise in the delete span. So this:

```
[[change:old text][new-text**comment**]]
```

becomes this:

```
<span class="org-change-added">
  new text
  <span class="org-change-comment">
    comment
  </span>
</span>
<span class="org-change-deleted">
  old text
</span>
```

You can then use CSS to display these classes as desired.

5.3 Producing a clean document

When exporting, org-change looks first at the variable `org-change-final`. This is initially `nil`, meaning that the export proceeds according to the selected backend as detailed above. If `org-change-final` is not `nil`, then only the new text is exported, resulting in a "clean" document without change markup. To achieve this, you can evaluate this code block before exporting:

```
#+begin_src elisp :exports none :results silent
  (setq org-change-final t)
#+end_src
```

This code can be anywhere in your file, even a `:noexport:` section.

6 Customizing and extending

6.1 Customization

The key sequences and the face used to display change links can be changed through the customize interface:

```
M-x customize-group RET org-change
```

6.2 Adding exporters

To add an export format, add something like this to your org file:

```
#+begin_src elisp
  (org-change-add-export-backend 'backend 'backend-function)
#+end_src
```

where `backend` is a backend known to org-mode and `backend-function` is a function that produces the desired string from three string arguments: `old-text`, `new-text`, and `comment`. The function can figure out whether the change is an addition, deletion, or replacement by looking at these variables: for additions, `old-text` is empty; for deletions, `new-text` is `((DELETED))`; other cases are replacements.

7 Bugs and limitations

- The content of the change link can contain org-mode notation like **bold** and *emphasis*, as well as Latex code. However, some other features do not currently work. Notably, org-ref links must be translated manually to Latex. So this will **not** work:

  ```
  [[change:][Let's cite something cite:&something1972]]
  ```

  But this will:

  ```
  [[change:][Let's cite something \cite{something1972}]]
  ```

- Link hiding is sometimes inaccurate in org-mode. You may see stray brackets especially with link that span multiple lines. Often `M-q` takes care of this, or you can enable `visual-line-mode` and keep paragraphs as single unbroken lines.

8 Planned features

- Simple HTML CSS for change markup.
- More export filters?

Send suggestions to drghirlanda@gmail.com.

9 Notes

To get started on org-change, I described some features to ChatGPT (April 2023 version) and asked for the corresponding code. It was wrong in many ways, like using non-existing functions with plausible names (`org-escape-latex`) and other non-existing features. It also insisted that some things would work even when told that they did not. It did have a good grasp of many things, like defining a minor mode and customize variables, and it was always syntactically correct.