The PracTEX Journal, 2010, No. 1
Article revision 2010/02/12

Enhancing Command Completion for TEXShop

Herbert Schulz
herbs2@mac.com

Abstract LATEX environments and commands are rather wordy markup. These make the intentions of the author easy to determine but more difficult to write. Using Command Completion, authors can write a few letters and trigger an expansion into complete environments and commands along with ways of going between arguments of those commands. In this paper I present an enhancement to Command Completion in TEXShop that allows more consistent completions and inclusion of short comments to help authors remember the order and contents of the arguments to those environments and commands.

1 Introduction & History

TEXShop is a popular Editor, Viewer and TEX Front End on the Macintosh. As of v1.34 TEXShop has offered a Command Completion facility that is reasonably powerful, if under-utilized. Command Completion in TEXShop allows continuations (Completions) and substitutions (Abbreviations) for a set of characters bounded on the left by a Word Boundary Character[1] and triggered by the Escape (`Esc`) key.

With the help of the good folks on the Mac OS X TeX e-mail list[2], I put together a `CommandCompletion.txt` file along with associated Applescript macros to take advantage of that facility. The completions and abbreviations supplied often contain bullet characters, '•', called Marks[3], as placeholders for command arguments or to easily get to the end of an environment. Skipping forward/backward and

1. The Word Boundary Characters are space, tab, linefeed(newline), period, comma, semicolon, colon, {, }, (,) or \ (actually the `TeX Command Character` which can vary in different implementations). The { and \ also become part of the expansion.
2. Subscribe by sending an e-mail to <mailto:MacOSX-TeX-on@email.esm.psu.edu>.
3. Previously called Tabs.

```
\rule[#INS#]{•}{•}
```

```
\rule[|]{•}{•}
```

Figure 1: Original `CommandCompletion.txt` contents and result in the document source. Here | is the insertion point.

selecting/deleting these Marks were accomplished using macros[4]. Most of the abbreviations were inspired by those used in the FasTEX[5] set used with TypeIt4Me[6]. The completion/abbreviation list created for TEXShop is the basis for the similar feature used in TEXworks[7].

Early in 2006, Hugh Neary sent me some Objective C code to implement the macros as an integral part of TEXShop. I modified that code and used it for quite a while in a personal build of TEXShop.

In April of 2006, Will Robertson added an extension to the Applescript macros that allowed the addition of an explanatory Comment within the arguments; the macros selected the Mark and Comment so typing replaced both with the entered information. This was reported in the `mactextoolbox-talk` list and some discussion followed it about the best delimiters for Comments but there was no final conclusion; the original suggestion of "•‹" and "›[8]" has been retained. Unfortunately, the original completion code could only position the insertion point (i.e., the cursor) so the initial selection could only have zero length (see Figure (1)); inconsistent with the behaviour with other arguments since you could not move back to the first argument using the macros.

At that point I decided to do a complete re-write of the code to implement a reasonably general version of Will Robertson's ideas and, at the same time, extend

4. The original `CommandCompletion.txt` files, macros and documentation are still available as `CommandCompletion.zip` from <http://homepage.mac.com/herbs2/>.
5. FasTEX was developed by Filip G. Machi, Jerrold E. Marsden and Wendy G. McKay. For more information see the FasTEX web page, <http://www.cds.caltech.edu/~fastex/>.
6. TypeIt4Me, by Riccardo Ettore, version 3 and later is a preference pane that allows abbreviation replacement in most OS X programs. See the TypeIt4Me web page, <http://www.typeit4me.com/>, for more information.
7. TEXworks is a multi-platform Editor, Viewer and TEX Front End using the same design philosophy as TEXShop. It was written by Jonathan Kew. More information is available at <http://www.tug.org/texworks/>.
8. Note that '‹' and '›' are single "guillemot" glyphs, not '<' and '>'.

the Command Completion code so that the implementation was consistent starting with the first argument of the completion. The result is backward compatible with the original behaviour of that code but with additional capabilities.

2 Changes to TeXShop.

Four interconnected changes were made in TeXShop: an addition to the way TeXShop handles completions from `CommandCompletion.txt`; a new menu with commands for searching and selecting Marks within completions; the ability to have comments attached to Marks; a new `CommandCompletion.txt` file that takes some advantages of the previous three changes. The rest of this section discusses each of these changes in more detail.

2.1 Changes to Completion Handling

Completions (in the `CommandCompletion.txt` file) in previous versions of TeXShop could contain a single `#INS#` command for the positioning of the insertion point within the completion.

This version of TeXShop allows completions to have *two* copies of `#INS#` and the text between them is selected. A single `#INS#` behaves the same as before; there is complete backward compatibility with previous versions of TeXShop.

2.2 A New `Source`→`Completion`→`Marks` Menu

The new `Source`→`Completion`→`Marks` menu contains commands to search for, move to and select Marks and Comments. The commands are shown in Table (1) and the default command menu appears in Figure (2). The `(Del)` versions of the search commands only show in the menu when the Option (`Opt`) key is pressed and the `Insert Comment` command only appears when you hold down the Control (`Ctl`) key. The `Insert Mark` command is added since TeXShop's autocompletion (keybinding) facility will insert `\bullet` in the document when the keystroke that normally inserts a '•' (`Opt`-8 with a US keyboard mapping) is pressed.

Menu Item	Shortcut	Internal Connection
`Next Mark`	`Ctl-Cmd-F`	Jump to and select the next Mark and/or Comment.
`Next Mark (Del)`	`Ctl-Opt-Cmd-F`	Jump to and select the next Mark and/or Comment and delete the Mark. This is most useful when you have nested environments to automatically delete a Mark at the end of an inner environment.
`Previous Mark`	`Ctl-Cmd-G`	Like `Next Mark` but search backwards.
`Previous Mark (Del)`	`Ctl-Opt-Cmd-G`	Like `Next Mark (Del)` but search backwards.
`Insert Mark`	`Cmd-8`	Places a Mark at the insertion point. Handy for creating completions in `CommandCompletion.txt`.
`Insert Comment`	`Ctl-Cmd-8`	Places a Comment Skeleton, "•‹›" with the insertion point before the "›", at the insertion point. Handy for creating comments in `CommandCompletion.txt`.

Table 1: Commands in the `Source`→`Completion`→`Marks` Menu.

2.3 Comments

The change mentioned in the previous sub-sections allow completions to contain Comments—short "memory joggers" that have some information about the contents of a given argument. The comments are contained within arguments and are surrounded by "•‹" and "›" within the arguments; if the first argument contains a comment it should be surrounded by two `#INS#` so it is the initial selection.

2.4 The New `CommandCompletion.txt` File

The `CommandCompletion.txt` file that comes with this version of TEXShop replaces all single `#INS#` commands by `#INS#•#INS#` so that the initial selection is a se-

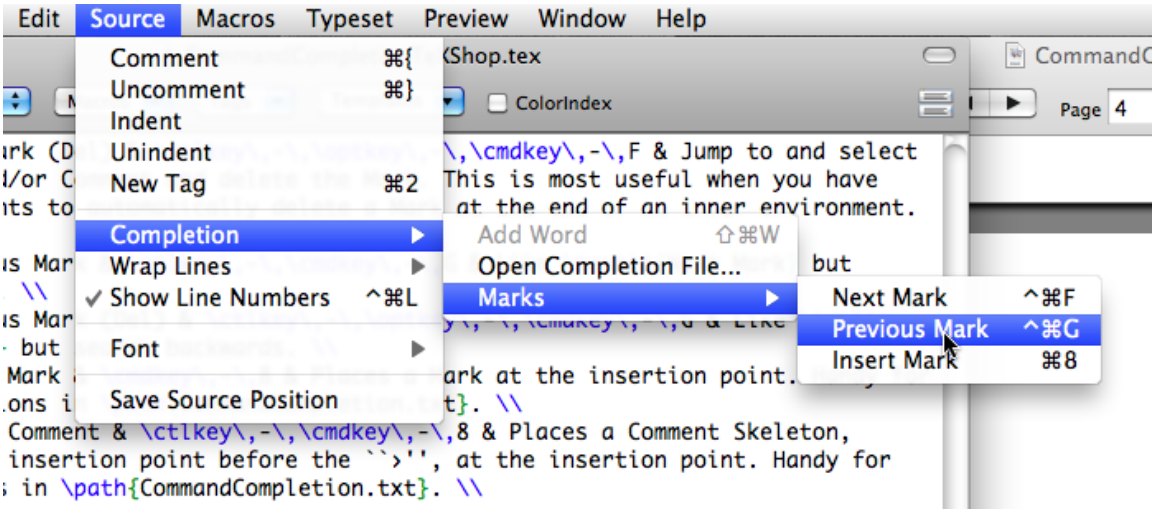

Figure 2: The Default `Source` → `Completion` → `Marks` Menu.

```
\rule[#INS#•#INS#]{•}{•}
```

```
\rule[•]{•}{•}
```

Figure 3: New `CommandCompletion.txt` contents and result in document.

lected Mark, •, for consistency in appearance and behavior when using the commands in the `Source`→`Completion`→`Marks` Menu. Figure (3) shows what this looks like.

The file, in addition, contains a few (too few?) examples of using comments.

3 Usage

3.1 Command Completion

A Command Completion is typically used to set up environments. To do this type `\b` and `Esc`; this should return `\begin{`. Then start to type the environment name; e.g., `eq` and `Esc` will give

```
\begin{equation}
•
\end{equation}•
```

while the next Esc gives eqnarray followed by it's *-variant. After entering your equation text at the cursor run the Source→Completion→Marks→Next Mark command and the cursor will select (and delete if Next Mark (Del) is used) the next '•' so you can start to type following text.

The macros are also handy for commands with multiple arguments. For example, to create a new command with an optional argument type \new or \newc and then Esc three times to get

```
\newcommand{•}[•][•]{•}
```

with the first mark selected. After entering the new command's name, please use the Next Mark command to jump to the next argument, etc.

3.2 Abbreviations

In addition to command completion, there are many abbreviations for commands. The principal difference is that the abbreviations are not just the start of a command name. For example typing benu and then pressing Esc *at the beginning of a line*[9] will produce the complete enumerated list environment:

```
\begin{enumerate}
\item
•
\end{enumerate}•
```

as you might expect. Abbreviations like this exist for many environments as well as sectioning commands. Alternate command versions with one or more options or *-variants have names that end with 'o' (one or more) or 's' respectively: e.g., sec and two presses of Esc or secs and a single Esc at the start of a new line give \section*{•}. By the way, After typing the text for the first item, typing it and Esc on a new line will generate another \item with a selected Mark on the line below it; continued presses of Esc will give \item[•] with a Mark on the

9. Or after any other Word Boundary Character.

following line, \textit{•} and finally \itshape before returning to the original it.

Remember that you must have one of the Word Boundary Characters before use; otherwise the substitution won't operate properly. This a not a problem with environments and sectioning commands, since you usually start them on a new line, but it can be for other abbreviations. Therefore many abbreviation also have a '\' version; e.g., `tt and Esc will not expand properly since the '`' isn't a Word Boundary Character while `\tt and Esc will expand to `\texttt{•} and a second Esc will give the declaration `\ttfamily[10].

Many of the Greek characters and in-line math versions of the Greek characters have abbreviations with the following rules:

1. The abbreviations for Greek characters all start with an 'x' and a notation for the character: e.g., xa or \xa[11] and Esc give \alpha.

2. The var version of several Greek characters start with 'xv' and the notation for the character: e.g., xth gives \theta while xvth and Esc gives \vartheta.

3. To get capitals for some letters use an 'xc': e.g., xg gives \gamma while xcg gives \Gamma.

4. Finally, preceding by a 'd' gives the following Greek character as an in-line math equation: e.g., dxcd gives \(\Delta\).

Abbreviations will be completed and cycle through matches just like the command completions: e.g., both the abbreviation newcoo (note the 'oo' at the end of the abbreviation) and Esc or newc followed by three Esc key presses on a new line give \newcommand{•}[•][•]{•}, the \newcommand with two optional arguments. There are alternate abbreviations for some commands: e.g., ncm gives the same result as newc.

Read the CommandCompletion.txt file to see what abbreviations are available; all lines with ':=' are abbreviations. Naturally, you can change them to suit your needs, adding or deleting others.

10. Similar abbreviations exist for bf, sf, sc, etc. Math versions have a preceding m; e.g., mbf and Esc will give \mathbf{•}.
11. All of the Greek character abbreviations have \ versions.

```
\rule[#INS#•<lift>#INS#]{•<width>}{•<height>}

\rule[•<lift>]{•<width>}{•<height>}
```

Figure 4: New `CommandCompletion.txt` contents with comments and result in document.

3.3 Comments

I tend to remember the arguments for commands that I use fairly often but forget those I rarely use; these are the perfect candidates for comments. I can never remember the order of the arguments for the `\rule` command so I type `\rul` and `Esc` twice to get the result show in in Figure (4). Another example is the `wrapfigure` environment, from the `wrapfig` package, which has multiple versions with differing numbers and positions of optional arguments. To see the variations with the comments type `bwr` on an empty line and press `Esc` to get:

```
\begin{wrapfigure}{•<placement: r,R,l,L,i,I,o,O>}{•<width>}
•
\end{wrapfigure}•
```

and versions with optional arguments on succeeding presses of `Esc`.

Other Environments

Environments that aren't built into the `CommandCompletion.txt` file can always be added if you use them a lot but there is an alternative for occasional use. Built into the completion algorithm is a way to complete environments. First press `\b` and `Esc` to get `\begin{`, enter the environment name and the closing `}` and then `Esc` again; the closing `\end{...}` with the corresponding environment name will be generated on a separate line.

4 Making Additions to `CommandCompletion.txt`

If you are adding items to the `CommandCompletion.txt` there are a few things you should know about its structure:

– Each environment has three entries: a completion that removes the leading `\begin`, i.e., it starts with a leading '`{`' and the environment name; two abbreviations that have an abbreviation name without a backslash (\) and the same abbreviation with the backslash. Commands usually have three or more forms, with and without a leading \, as well as possible abbreviations, also with and without \.

– You should add all the variations with slightly different endings for the abbreviations. I use an '`o`' at the end of an abbreviation if that variation has an optional argument, '`oo`' for two optional arguments, '`s`' for starred forms of commands, etc.

– The order of similar items in the file *does* make a dramatic difference in the order in which items are found; items placed *later* will be found *earlier* (the file is searched backwards). E.g., the order of items obtained when you press `\b` and then `Esc` depends purely on the order of matches in the `CommandCompletion.txt` file.

– For maximum convenience place a Mark[12] within each argument of commands. Surround the very first argument with two `#INS#` commands so it comes out selected. If you want to have a comment in any arguments insert a Comment Skeleton[13] and fill it in.

I'd suggest taking a look in the `CommandCompletion.txt` file for examples.

5 What's Missing

I'd love to be able to have the completions preserve indentation but that is not in the books for now.

Any other suggestions are welcome and will be considered for inclusion in later iterations of the Command Completion code.

12. Using `Insert Mark` (`Cmd-8`) from the `Source→Completion→Marks` menu.
13. Using `Insert Comment` (`Ctl-Cmd-8`) from the `Source→Completion→Marks` menu.

6 Obtaining the version of TeXShop.

The enhanced version of Command Completion is incorporated into TeXShop 2.30 and later. It is available at the TeXShop web site, <http://www.uoregon.edu/~koch/texshop/texshop.html>. Make sure you read the `Help→About This Release` document to enable the updated `CommandCompletion.txt`, etc.