A CommandCompletion.txt Replacement for TeXShop

Herbert Schulz
herbs@wideopenwest.com

Introduction

TeXShop, v1.34 and later, has a feature that allows completions or substitutions (abbreviations) for a set of characters bounded on the left by a Word Boundary Character[1] with the use of the '⎋' (Escape) key.

The enclosed CommandCompletion.txt or RCommandCompletion.txt file is a replacement for the file found in the ~/Library/TeXShop/CommandCompletion/ directory. It adds many commands and abbreviations for the most commonly used LaTeX 2ε commands as well as variations for optional arguments and *-variants. The entries are ordered so that the most often used variations come first when the ⎋ key is pressed[2]. If the first completion/abbreviation doesn't match what you want continue pressing the ⎋ key to get variations and other matches. Eventually you will return to the original characters you typed.

Completions/abbreviations often contain bullet characters, '•', called "Marks", as placeholders for command arguments or to easily get to the end of an environment. Skipping forward and backward to these "Marks" is done by the enclosed Applescript Macros.

Most of the abbreviations were inspired by those used in the FasTeX[3] set used with TypeIt4Me[4].

Re-arrange the file entries to suit your needs. Feel free to delete commands you never use. Add commands or abbreviations using the ones in the file as examples.

This is an ongoing personal project and you can expect additions and changes as time goes on. I would appreciate any ideas, comments, wish lists, bug reports, etc., from anyone that tries using this version of CommandCompletion.txt.

Installation

CommandCompletion.txt

If you've already made changes to your CommandCompletion.txt file rename it before substituting the replacement file. You can copy any changes you made to the re-

[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] If you open the CommandCompletion.txt file using the Format->Completion->Open Completion File... menu item, and possibly other situations, the search direction through the list sometimes reverses. You can then use ⇧-⎋ to search through the other way.

[3] 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/>.

[4] TypeIt4Me, by Riccardo Ettore, is an input method that allows abbreviation replacement in most OS X programs. See the TypeIt4Me web page, <http://www.typeit4me.com/>, for more information.

placement later. You can always regenerate the original by emptying the ~/Library/TeXShop/CommandCompletion/ directory and restarting TeXShop.

One thing to note: under OS X 10.2 TeXShop scanned the file for completions from top to bottom; under OS X 10.3 and 10.4 (on my system) it seems to scan the file from bottom to top. To test which way your system is scanning the file:

1. Use the enclosed CommandCompletion.txt file (this is the one I use for OS X 10.3 and 10.4) and type \use and then the ⎋ key in a new document. You should get \usepackage{} with the cursor between the braces. A second press of ⎋ should give you \usepackage[]{•} with the cursor between the square brackets.
2. If you get these in reverse order the file is being scanned from top down. You need to use the RCommandCompletion.txt file. Delete the leading R and replace the original file with it. Try the test again; it should give you \usepackage{} first and on the second ⎋ the second form.

Applescript Macros

Add the following two Applescripts to the macros menu and assign convenient keyboard shortcuts to each one; I use ^-⌘-F (Ctl-Cmd-F) and ^-⌘-G[5]. Then you skip across argument boundaries and to the end of environments quickly[6] by executing the macros.

```
--AppleScript direct
--Remove the 'direct' for TeXShop v1.34
--Next "Mark"
--2005-10-10
set searchString to "•"
try
 tell application "TeXShop"
  activate
  set searchOffset to (offset of selection of front document)
  if (length of selection of front document) > 0
   set searchOffset to (searchOffset + 1)
  end if
  tell front document
   set firstOccurrence to (search for searchString starting from searchOffset)
  end tell
  if firstOccurrence > 0 then
   set offset of selection of front document to (firstOccurrence - 1)
   set length of selection of front document to (number of characters of contents of searchString)
--   set content of selection of front document to ""
  end if
 end tell
end try
-- end of script

--AppleScript direct
--Remove the 'direct' for TeXShop v1.34
--Previous "Mark"
--2005-10-10
set searchString to "•"
try
 tell application "TeXShop"
  activate
  set searchOffset to (offset of selection of front document)
  tell front document
   set firstOccurrence to (search for searchString searching backwards true starting from searchOffset)
```

[5] I was using ^-⌘-D, which I find easier to type, but that keystroke is now being used by OS X 10.4 for Dictionary Lookup.

[6] Well, not very quickly with versions of TeXShop before 1.35, Sigh. . . .

```
  end tell
  if firstOccurrence > 0 then
   set offset of selection of front document to (firstOccurrence - 1)
   set length of selection of front document to (number of characters of contents of searchString)
--   set content of selection of front document to ""
  end if
 end tell
end try
-- end of script
```

By default the bullet character, '•', is obtained by pressing ⌥-8 on English keyboards. You can copy and paste the scripts from the enclosed text file, `Applescripts.txt`.

To add the macros go to TeXShop's `Macro->Open Macro Editor...` menu command. For each macro in `Applescripts.txt`:

1. Choose `New Item`, paste a copy of the macro from the `Applescripts.txt` file into the `Content:` window and type its name into the `Name:` window.
2. Assign a convenient keystroke; '`f`' or '`g`' and check the `Control` box to get ^-⌘-F and ^-⌘-G respectively.
3. Drag the macros where you want them to appear in the `Macros` menu. I added a separator between the Applescripts and Begin/End and put both of the commands between them so they are set off from the rest of the commands.
4. Save the results.
5. Check that everything is working by adding some commands using Command Completion and executing the macros.

As given above the macros select the next/previous '•' but don't delete it. Immediately entering any text will than replace the '•' with that text. If you use many nested environments this usually means having to use the '⌫', delete key, to eliminate one or more '•' that denote the end of an inner environment. To have the '•' deleted before entering text un-comment the line

```
--   set content of selection of front document to ""
```

in the macros given above by removing the "`--`" from the start of the line. I actually have both versions of the macros on my system and assign the version that also deletes the '•' an extra '⌥' for its keystroke; e.g., ⌥-^-⌘-F runs `Next "Mark" (Del)`.

I also added a tiny macro that inserts a '•' directly and called it `Insert "Mark"` because I use TeXShop's "auto completion" to substitute '`\textbullet`' for '•' as I type.

Usage

Command Completion

A typical use of command completion is to set up environments. To do this type `\b` and the ⎋; this should get you `\begin{`. Then start to type the environment name; e.g., `eq` and the ⎋ will give

```
\begin{equation}
<- cursor
\end{equation}•
```

while the next ⎋ gives `eqnarray` followed by it's ⋆-variant. After entering your equation text at the cursor run the `Next "Mark"` macro and the cursor will select (and possibly delete, if so configured) the '•' so you can start to type following text.

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

```
\newcommand{}[•][•]{•}
```

with the cursor placed within the first set of braces. After entering the new command's name use the macro to jump to the next argument, etc.

Abbreviations

In addition to command completion there also exist 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 ↻ *at the beginning of a line* will produce the complete enumerated list environment:

```
\begin{enumerate}
\item
<- cursor
\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 *-versions have names that end with '`o`' (one or two) or '`s`' respectively: e.g., `sec` and two presses of ↻ or `secs` and a single ↻ at the start of a new line give `\section*{}` with the cursor between the braces. By the way, After typing the text for the first item, typing `it` and ↻ on a new line will generate another `\item` with the cursor on the line below it; continued presses of ↻ will give `\item[]` with the cursor between the square brackets and a "Mark" on the following line, `\textit{}` with the cursor between the braces and finally `\itshape` before returning to the original `it`.

You must remember to have one of the `Word Boundary Characters` before the use or 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 can be for other abbreviations. Therefore many abbreviation also have a '`\`' version; e.g., `` `tt `` and ↻ will not expand properly since the '`` ` ``' isn't a Word Boundary Character while `` `\tt `` and ↻ will expand to `` `\texttt{} `` with the cursor between the braces.

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`[7] and ↻ 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 ↻ 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 ↻ or `newc` followed by three ↻ 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`.

[7] All of the Greek character abbreviations have \ versions.

I suggest that you read through the CommandCompletion.txt file to see what abbreviations are available; all lines with ':=' are abbreviations. Naturally, you can change them to suit your needs and add and delete others.

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 ⎋ to get \begin{, enter the environment name and the closing } and then ⎋ again; the closing \end{} with the corresponding environment name will be generated on a separate line.

Actually, it is much easier to edit the Begin/End macro to have the returns, etc., the way you'd prefer add the "Mark" at the end and assign ^-⌘-B to that macro. Then you need only type the environment name, select it, type ^-⌘-B and you'll have the environment.

Technical Stuff

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 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.
- 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 (or earlier if you needed the RCommandCompletion.txt file) will be found earlier. E.g., the order of items obtained when you press \b and then ⎋ depends purely on the order of matches in the CommandCompletion.txt file.

I'd suggest that you take a look in the CommandCompletion.txt file for examples.

Try it… I hope you like it.

Good Luck,
Herb Schulz
(herbs@wideopenwest.com)