The TabSpace Twiddler Key Assignments

Brandon Craig Rhodes
brandon@rhodesmill.org
Georgia Institute of Technology
15 November 1999

This document reflects the thought and reason that informed my development of the TabSpace key assignments for the Twiddler. Although this may not be terribly interesting, it should be valuable reading for new users of the key mapping who by considering its spirit and logic will be able to learn it more rapidly.

Introduction

This document describes the Twiddler key mapping I developed after my recent purchase of my first Twiddler. Two mappings were already available. The default assignments provided by HandyKey were deeply dissatisfying because of their staunch disregard for efficiency and the difficult (at times approaching haphazard) placement of punctuation. The alternative developed by Brad Rhodes (which may be downloaded from HandyKey's web site) certainly showed a heightened sense of efficiency, but seemed to lack the strong spatial associations that would make a key mapping easily for me to remember.

The key map I developed in response to these misgivings was guided by several principles:

- The most common characters should be assigned simple key sequences, and combining two of these characters to make a chord should output both letters in the order they most often appear in English text. Longer combinations like "the" and "ain" should also be produced simply by pressing their constituent letters simultaneously. It is not acceptable for such shortcuts to be completely independent chords since this would impose significant cognitive overhead on learning.
- Every letter, number, and piece of punctuation should be chordable without using any modifiers. Not only do I find the modifier keys to be awkward, but I need the modifiers and chords to be orthogonal so I can enter combinations like Meta-3 (yes, I use Emacs).
- There should be deep order in the character arrangement. Even if I forget where an infrequently used key lies, I should know within two or three tries where to find it. The spatial arrangement should make the most sense when viewed from behind the Twiddler (the direction from which I view it while typing).
- Numbers should be in a 3×3 grid like on my telephone and keypad.
- The middle column of Twiddler keys should be considered the home row on which fingers will normally rest (though, of course, they are really on the home *column*). This column is marked with tactile dots on the Twiddler.
- Where possible chords should involve keys from the same column; and when a chord includes keys from different columns, the third and fourth fingers should always remain on the same column (since I find them awkward to place independently). This principle shall only be violated when necessary to satisfy the first principle. Four-finger chords should also be avoided because the leverage required to produce them makes them slower and more error-prone than other chords.

Note that the mapping as presented here and coded in the tabspace.ini file is designed for *left-handed* Twiddling; it is not clear whether those who Twiddle with their right hands will need to reverse the mapping.

Theory of Operation

The index finger is the pivot around which the map is designed. Its home-column key produces a space, that most gargantuan of keys on a normal keyboard, while its other keys produce tab and

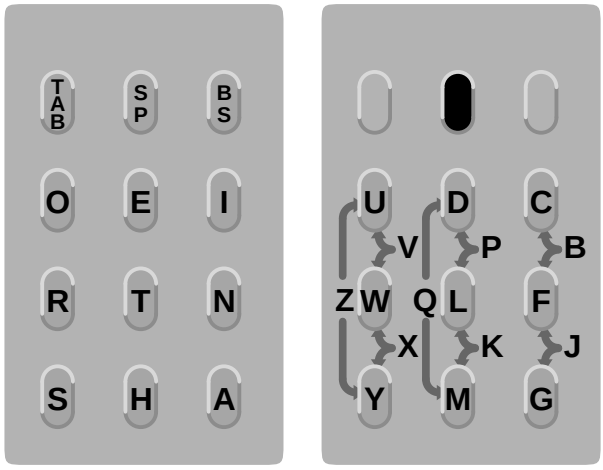
backspace (see the diagram below under the discussion of the alphabet). When viewed from behind the Twiddler, the backspace is to the left, and tab to the right, of the spacebar — the directions in which they normally move the cursor on the screen. These three keys are the fundamental ingredients both of completion-based Unix command line editing and of source code typing in the Emacs editor. I feel in particular that any map suitable for use with modern shells must make the tab character at least as accessible as the enter key.

The index finger is used to multiplex among different meanings for the chords formed by the second through fourth fingers. When the other fingers are used alone they produce common letters or sequences of these letters; with the index finger pressing the space bar they produce the rest of the alphabet. Thus normal text with spaces may be typed without ever moving the index finger from the home column (well, as long as no typing mistakes are made).

Making chords based on the tab character gives us punctuation, while all chords cascaded below the backspace key give numerals and arithmetic operators. Both of these sub-maps are heavily enough populated that they use the second finger to multiplex among smaller maps for the third and fourth fingers. Although this may sound awkward or artificial when stated verbally, the resulting key map is quite easy to learn and use once this structure has been understood by the fingers.

The Alphabet

Although the TabSpace map is probably best conceived from the point of view of someone looking at the back of the Twiddler, my diagrams follow the universal practice of showing the Twiddler from the front and thus reflecting properly the “LMR” names of the keys of each row. Letters are arranged as follows:



The Alphabet

Fortune has smiled upon us. English letter frequencies are divided into five distinct echelons: the letter “e” with frequency 0.12, the letters “taoinshr” that range from 0.09 down to 0.06, “dl” which are both near 0.04, “cumwfgypb” down around 0.015–0.028, and finally the outcasts “vkjxqz” which all fall below 0.01 in frequency. The order of letters within each frequency range varies depending on the author and type of document, though they rarely get very far out of the order in which they are listed in the quotes above. The fortunate aspect of this is that the top two frequency groups give us exactly nine letters to place on single Twiddler keys! We call these letters the first *tier* of the TabSpace alphabet, and they promise that over half (roughly 0.55) of the letters in a typical English document are available through single keystrokes.

Chords made by combining these first-tier letters produce common sequences. All two-letter pairs have been mapped, although some of them were close calls frequency-wise; so while no one will be surprised that pressing `00mm` produces “th” (0.045) instead of “ht” (0.0022), it may not be practical to remember that `0ml0` gives “er” (0.020) instead of “re” (0.019) and many users

may simply avoid that chord. If a TabSpace typist remembers to use every two-letter combination available, then a full third (0.34) of the adjacent letter pairs in an English document will take only a single chord each. The three-letter combinations “the”, “ain”, and “are” are also recognized.

The second tier of letters are typed in conjunction with the space bar. The more common letters have keys to themselves, and less common ones are formed from decreasingly convenient chords. While frequency was the primary factor in their placement, effort has been made to make the map more easily remembered by placing alphabetically adjacent letters near one another. Thus the groups b-c-d, f-g, j-k, and k-l-m-p are placed in proximity, while v-w-x-y-z are arranged serially down the left column.

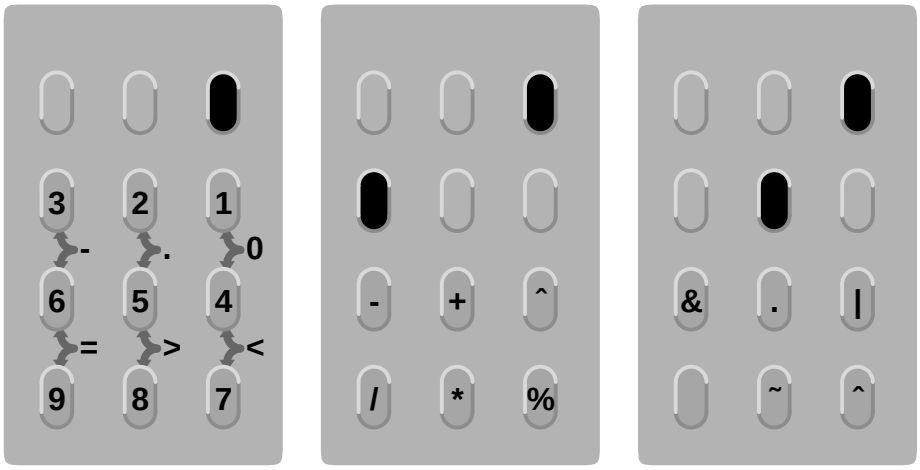
Stepping back and looking at both tiers of our alphabet, it is clear that both maps follow the rough plan of placing the beginning of the alphabet on the right side of the keypad and the end of the alphabet left. From our position as typists behind the Twiddler this plan is of course reversed, and the alphabet arcs from the lower-left corner down at our pinky with “a” and finishes along the right edge where we must reach to type the final “z”.

Chords that include both first and second tier letters not only fail to specify the order they will appear in, but do not even uniquely determine which letters will be produced. The combination `mmr0`, for example, could be indicating either “e” or “d” with its second finger and “n” or “f” with its third — and thus any one of the three combinations e-f, d-n, or d-f (we know it does not mean e-n since these are both first-tier letters and would not be typed in conjunction with the space). As it turns out TabSpace defines this as “nd” — the user who learns to employ it will probably just remember that “nd” is available and then type the obvious chord for it, rather than remembering that `mmr0` is a defined chord and trying to figure out which letter combination it represents.

Because second-tier chords impose this memory load there are only five pairs and three longer sequences currently defined: “of”, “nd”, “ch”, “qu”, “wh”, “ing”, “and”, and “ong”. Users will probably vary in which of these they remember and choose to employ; they should feel free to suggest others that they discover are useful in their own Twiddling. Note that “qu” is not actually the combination of the “q” and “u” chords, but rather is an alteration of “q” in the direction of the “u”. These five pairs together account for about 0.04 of the adjacent letters in an English document.

Mathematics

The digits and mathematical operators are arranged below the backspace key. Remember that Twiddler drivers allow a sequence of similar chords to be entered without releasing the keys they have in common — so you can enter a sequence of digits without releasing backspace. Note as well that many of the mathematical operators can also be accessed as punctuation (see the next section).



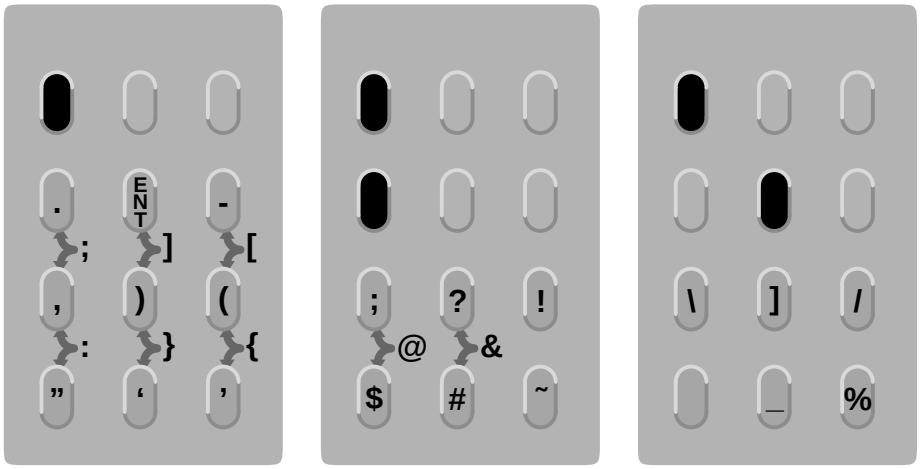
Numbers and Operators

The digits are in a typical 3×3 matrix, which reads from left to right and top to bottom when the Twiddler is viewed from behind. Zero and the two other characters necessary to enter numbers (the negative sign and decimal point) are alongside the zero, and two-fingered combinations along the bottom row yield comparison operators. Notice that the inequality operators are positioned so that, if we think of them as angle brackets instead, they follow the conventions established for all other brackets in the punctuation section of the key map (see the next section).

When the first and second fingers are placed to produce the minus sign they also make available several other operators — one row of additive (and exponential) operators and a lower row of multiplicative ones. Below the decimal point is another group of symbols, this time supporting logic operations under many procedural languages.

Punctuation and Brackets

The most heavily populated group of symbols are the punctuation. Like the numeric keypad, this category has a main group of symbols and two additional groups of symbols selected with the second finger.



Symbols and Brackets

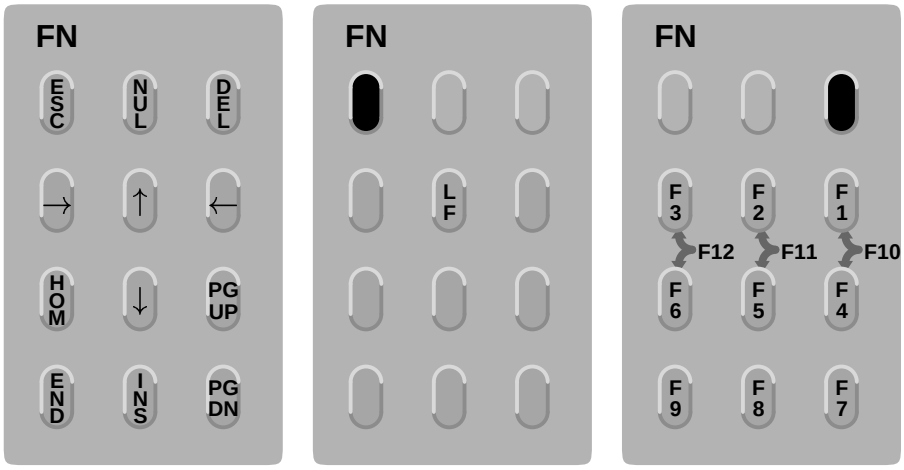
The two most easily accessible keys are the period, which is typed with a very compact and solid two-finger chord, and the enter key, which (when viewed from behind) slants down and to the left — the direction that a carriage return usually sends the cursor. Pressing both the period and comma produces a semicolon (of course), and the colon is one row beneath it. The dash earns second row status, and all the kinds of quotes are draped across the bottom.

Remember that I look at my Twiddler from behind, so opening brackets and quotes which are positioned the “right way” relative to the corresponding closing brackets appear reversed in the diagram (which looks at the Twiddler head-on).

Arranging the remaining punctuation marks is a difficult issue bound to make many people unhappy. Do something nice for Perl programmers, and TEX enthusiasts will wail; optimize for normal English sentences and punish Bash programmers. Here the interrogative and exclamation marks are placed together, with some other common text symbols placed beneath them. The biggest and fattest symbols — the at sign and ampersand — are the only characters assigned big four-finger chords in TabSpace. The final group of symbols is composed of various sorts of slashes and lines. Note that the slash chord actually looks like a slash from behind.

Function Keys

TabSpace defines only a few of the chords marked with the FN modifier; see the next section for a discussion of what might be done with the rest. Chording a digit enters the corresponding



Function Keys

function key, and most of the rest of the keys are from the navigation portion of a typical PC keyboard.

Chord Population and User Extension

Since many Twiddler users like to extend their keymaps, we should observe which chords this key map chooses *not* to define. The most important feature of TabSpace in this respect is that it does not and will never employ the NUM modifier — it bequeaths to its users forever all the chords they wish to invent that are modified with NUM. If managed carefully this can provide more than two hundred chords related to each user's machine and activities.

TabSpace does not make very efficient use of the area below FN; it monopolizes all of the single-key chords and a smattering of the more complex ones while leaving most multi-finger chords undefined (those corresponding to normal punctuation, to second-tier letters and multi-letter chords, and to most mathematical symbols). This creates a quite inviting area for application-specific chords. The default TabSpace definitions exemplify this by populating some of this area with common Emacs escape sequences which would otherwise be less convenient to chord.

It may be desirable for the community of TabSpace users to work together on a set of standard application-specific FN extensions; there would be a series of TabSpace chord maps with FN chord sets for users of Emacs, vi, et cetera. Users could dynamically reload the appropriate map for the application they are currently running. All TabSpace users are therefore encouraged to inform the TabSpace maintainer of useful application extensions they develop that could benefit other users if incorporated into the standard distribution.

The status of the Shift, CTRL, and ALT modifiers is less readily defined since (unlike FN) they actually have conventional meanings. Even though X-Windows (for example) will recognize such combinations as Shift-5, CTRL-$ and ALT-=, it is not clear whether these are useful to most users. Even an Emacs user will not usually use non-ASCII CTRL sequences (since they can not be represented over a serial connection) nor will he be likely to use the chord for Shift-5 to enter a percent sign when there is another perfectly good unshifted chord that makes it.

TabSpace makes the conservative decision that it will never molest the combinations of SHIFT with any letter, CTRL sequences that map to ASCII codes, and all ALT characters. But the status of SHIFT with non-letters and irregular CTRL sequences may be susceptible to later definition by TabSpace if any natural uses for such sequences are developed.

Epilogue

I hope these chord assignments are useful to other users of the Twiddler. Feel free to contact me with any questions or comments.