Code Indentation: Tabs Versus Spaces

When it comes to code indentation, which is better, tabs or spaces?

> Hard tabs should be used for (left-side) indentation, spaces for (mid-line) alignment.

This document explains why.[1]

Table of Contents

1 Originally published on-line as PDF.

Hard and Soft Tabs

Pressing Tab moves the cursor to the next tab stop; the horizontal position where the next column starts. Nowadays, most IDEs can be set to either insert hard or soft tabs when Tab is pressed. Hard tabs add ASCII character 0x09 (\t), soft tabs add spaces. Coders who prefer using spaces for indentation can - and often do - use the (physical) Tab key.

Tab Stop Width

Hard and soft tabs create the same visual effect for the same tab stop width. This raises the question: do coders, projects and companies agree on what tab stop width should be used; is there at least a growing consensus or popular choice? No, not at all.

Roughly speaking, there are strong proponents of using tab stop width 2 (two), of using 4 (four), and of using 8 (eight). Some prominent examples:

- **2**, soft tab: GNU (source[2]), including, for example, GNU Octave (source); Scala (source); Drupal (source); Mozilla (source); Google (sources: 1, 2, 3, 4).
- **4**, soft tab: PHP Framework Interop Group (source[3]), including, for example, Laravel (source[4]) and probably[5] also the Zend Framework (source); Python (source[6]); Microsoft (sources: 1, 2, 3); WebKit (source); Java (source[7])
- **8**, hard tab: Linux kernel (sources:[8] 1, 2); Go (source[9])
- hard tab (in general): MediaWiki (source); WordPress (source[10])

Programmers have different tastes, *even* those working with the same programming (e.g. C) or scripting (e.g. PHP) language. At the same time, projects and companies expect programmers to use specific tab stop widths that vary, *even* those working with the same languages.

There is no consistency. One reason might be that the default - *de facto* standard - width for fixed tab stops of old printers and manual typewriters was 8, instead of 4 or 2. For many coders this width is excessive and non-optimal (source[11]), thus coders started experimenting with alternative widths.

---

2 Recommended, not required. The recommendation is also the default style of the indent program, and corresponds to command-line options that include "-i2", which sets the indentation level to two space characters.
3 This was accepted on June 4, 2012 (source).
4 Laravel currently follows PSR-2, but previously used tabs for indentation (source).
5 The documentation is for 2.4, I cannot find their indent standard for 3.0.
6 The status of this PEP 8 style guide is "Active".
7 This may be outdated. Also, hard tabs may be used. Regular tabs are set to width 8.
8 As tabs; as stated at the end of the section, "spaces are never used for indentation".
9 As stated, tabs are used, "gofmt emits them by default", and gofmt uses width 8 (source).
10 Spaces are used for alignment.
11 Quoting: "The levels of indentation we tested (0-6 spaces) gave strong results favoring 2 or 4 spaces." This is from 1983 though; computers and screens have changed.

Visual Control

Ideally, indentation should be flexible, to give *readers* - who may be fellow coders - the same control over the visual effect of tab stops as the (initial) *author*. This allows all coders to look at code as they prefer, regardless the preference of other coders, projects and companies.

Can either hard or soft tabs provide this freedom without problems?

Proponents of hard tabs may say: "Soft tabs cannot provide this freedom."
Proponents of soft tabs (spaces) may say: "Hard tabs cause problems."

It's easy to disregard either position if you're unfamiliar with other people's experiences, work methods and tools. For instance, the *under* toilet paper orientation never made sense to me, since *over* makes it easier to locate and grasp the loose end. Then a cat owner told me that *under* reduces the risk of house pets unrolling the paper. Interesting, right?

This is where the difference between indenting and aligning becomes relevant.

Indenting and Aligning

Below is example code that uses a soft tab of width 2.

```
void.FunctionA.(int.iX,
..............int.iY)
{
..int.iVar1...........=.1;
..int.iVarLonger2.....=.2;
..int.iVarEvenLonger3..=.3;

..FunctionF2.(iX,
.............iY,
.............iVar1,
.............iVarLonger2,
.............iVarEvenLonger3);
}
```

The code uses spaces for both indenting and aligning. This means it can be difficult to tell apart indentation from alignment. There is no semantic clarity.

When using hard tabs of width 2, the *visual effect* is similar.

```
void.FunctionA.(int.iX,
->->->->->->->->->int.iY)
{
->int.iVar1>->->->->->->=.1;
->int.iVarLonger2>->->=.2;
->int.iVarEvenLonger3>=.3;

->FunctionF2.(iX,
->->->->->->->->iY,
->->->->->->->->iVar1,
->->->->->->->->iVarLonger2,
->->->->->->->->iVarEvenLonger3);
}
```

But *not* the same.
Look closely at the distance between `iVarEvenLonger3` and `=.3;`.
Using a hard tab of width 4 shows the problem more clearly.

```
void.FunctionA.(int.iX,
--->--->--->--->--->--->--->--->int.iY)
{
--->int.iVar1-->--->--->--->--->--->=.1;
--->int.iVarLonger2>--->--->=.2;
--->int.iVarEvenLonger3>=.3;

--->FunctionF2.(iX,
--->--->--->--->--->--->--->iY,
--->--->--->--->--->--->--->iVar1,
--->--->--->--->--->--->--->iVarLonger2,
--->--->--->--->--->--->--->iVarEvenLonger3);
}
```

What a mess.
There are two ways to fix this problem.

The first solution is to distinguish between indentation and alignment. Sometimes called "smart tabs".

```
void.FunctionA.(int.iX,
...............int.iY)
{
--->int.iVar1...........=.1;
--->int.iVarLonger2......=.2;
--->int.iVarEvenLonger3..=.3;

--->FunctionF2.(iX,
--->............iY,
--->............iVar1,
--->............iVarLonger2,
--->............iVarEvenLonger3);
}
```

The second is to use elastic hard tab stops.

```
void.FunctionA.(--->int.iX,
------------------->int.iY)
{
--->int.iVar1---------->=.1;
--->int.iVarLonger2---->=.2;
--->int.iVarEvenLonger3>=.3;

--->FunctionF2.(--->iX,
--->--------------->iY,
--->--------------->iVar1,
--->--------------->iVarLonger2,
--->--------------->iVarEvenLonger3);
}
```

Elastic hard tab stops keep columns aligned by turning the Tab character into a delimiter of variable length.

Pros and Cons

Don't rely on your text editor to decide whether hard or soft tabs are used for formatting. Using soft tabs complicates the process required to get the visual effect matching the preferred tab stop width. With hard tabs, it's easy to switch from, for example, tab stop width 2 to 8. With soft tabs, the same action is more complicated, because a single indentation level is neither a single character nor programmatically distinguishable from alignment. Spaces alone are ambiguous. Even if *your* IDE can analyze these syntactic elements, the difference in semantic value remains. Even if *your* IDE can convert (leading) spaces to tabs and vice versa, code conversion shouldn't be - and is not - necessary.

A frequently used argument in favor of soft tabs is that spaces guarantee that code renders the same in any environment. If hard tabs are used either without elastic hard tab stops or for both indentation and alignment, things do indeed go wrong. While there are plug-ins and scripts to work with elastic hard tab stops, it is a relatively obscure solution that forces coders to use specific editors.[12] Using hard tabs *for indentation only*[13] immediately fixes the problem. With very few exceptions, it's *good* if code does not render the same in every environment. It doesn't matter whether you personally think your code looks good on the screen of someone else, because it's not an ASCII art competition. It matters whether that other person can read (and work with) your code. And tab stop preferences vary widely, from (at least) 2 to 8. For example, width 2 may feel claustrophobic to you, while it's all I ever work with. With hard tabs your code does not lose legibility.

Another point of discussion is related to copying code. Let's say we want to combine two code blocks, one uses tab width 2 and one uses 8. This is not a problem with hard tabs, because these cannot half-indent, they are always the same number of characters - namely one - in the code. Also, the cursor cannot be placed halfway the indentation, which, if possible, can cause problems not just when copying but also when un-indenting soft tabbed code.

The statement that "most people use spaces" is, in and of itself, not an argument in favor of using soft tabs. If true, it is (just) a factual statement about the *status quo*; about what is commonly used. The conclusion that it should *thus* be used, is an *argumentum ad populum*; an appeal to popularity. The statement that "famous people and prominent companies use spaces" is an argument from authority. It makes sense to look for consensus, but "use spaces" wouldn't be a solution anyway: the tab width preference, and thus the number of preferred spaces, still varies widely.

One potential problem with hard tabs is that, visually, they are indistinguishable from spaces. However, if you've chosen to work with hard tabs and make mistakes while

12 It is, however, the only solution when working with proportional fonts.

13 In some cases, hard tabs can also be used for alignment of continuation lines, as long as the starting point is the *start* of the first line.

getting used to them, there are editors that can display whitespace as 'ghost' characters, e.g. hard tabs as arrows, and spaces as dots. One potential problem with soft tabs is that spaces take up more storage space. However, on, for example, the web automatic whitespace removal (minification) and compression (gzip) can be used.

Conclusion

Tab width preference varies widely, and there's a single character whose visual representation is dynamic and can be configured manually (even on the web[14]), without modifying code. The hard tab. Use it for indentation. Don't use it for alignment, because, without the obscure and *non*-out-of-the-box elastic hard tab stops, that messes things up.

> Hard tabs should be used for (left-side) indentation, spaces for (mid-line) alignment.

Avoid pasting hard tabbed code directly into the body of emails, and be careful with programming languages that use whitespace to determine functionality (e.g. Haskell, F#, Makefiles).

14 With the tab-size CSS property.