**How Apple Does Controlled Leaks**

[**M**HYPERLINK "http://www.macobserver.com/tmo/article/wsj\_apple\_to\_release\_tablet\_in\_march\_for\_1000\_maybe\_including\_wifi/"onday's article](http://www.macobserver.com/tmo/article/wsj_apple_to_release_tablet_in_march_for_1000_maybe_including_wifi/) at the *Wall Street Journal*, which provided confirmation of an Apple tablet device, had all the earmarks of a controlled leak. Here's how Apple does it.

Often Apple has a need to let information out, unofficially. The company has been doing that for years, and it helps preserve Apple's consistent, official reputation for never talking about unreleased products. I know, because when I was a Senior Marketing Manager at Apple, I was instructed to do some controlled leaks.

The way it works is that a senior exec will come in and say, "We need to release this specific information. John, do you have a trusted friend at a major outlet? If so, call him/her and have a conversation. Idly mention this information and suggest that if it were published, that would be nice. No e-mails!"

The communication is always done in person or on the phone. *Never* via e-mail. That's so that if there's ever any dispute about what transpired, there's no paper trail to contradict either party's version of the story. Both sides can maintain plausible deniability and simply claim a misunderstanding. That protects Apple *and* the publication.

In the case of yesterday's story, Walt Mossberg was bypassed so that Mr. Mossberg would remain above the fray, above reproach. Also, two journalists at the *WSJ* were involved. That way, each one could point the finger at the other and claim, "I thought *he* told me to run with this story! Sorry."

Finally, the story was posted online late Monday, eastern time, so no one could ever suggest there was any attempt to manipulate the stock market.

The net result is that Apple gets the desired information published by a major Wall Street news outlet, but can always claim, if required, it was all an editorial misunderstanding. The *WSJ* is protected as well.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**C**ontrolled leaks are almost always the solution to a problem. In this case, it could have been that Apple needed to release the tablet information early because they wanted:

* to light a fire under a recalcitrant partner
* to float the idea of the US$1,000 price point and gauge reaction
* to panic/confuse a potential competitor about whom Apple had some knowledge
* to whet analyst and observer expectations to make sure the *right kind and number* of people show up at the (presumed) January 26 event. Apple hates empty seats and demands SRO at these events.

Of course, if Wall Street draws the right conclusions, and AAPL goes up, as it has, then everybody benefits. But the manipulation of stock is never the purpose. It's simply a favorable outcome of the process. Again, Apple is protected.

That's how Apple does controlled leaks, and the *WSJ* article from yesterday was a classic example.

**Disclosing Your Finances**

Prominent startups such as [Balsamiq](http://balsamiq.com/) and others like [Squarespace](http://www.squarespace.com/) and [37signals](http://37signals.com/) (to some extent) have been open about disclosing their revenue — and in Balsamiq's case also their profit. I'm not sure whether that's such a good long-term strategy and I'll argue why below. You've got to be careful that you're in the right position to share these confidential numbers, and I think that for anyone else the disadvantages outweigh the advantages.

**What's the Advantage?**

The obvious advantage for anyone is that as long as your numbers are large enough to talk about then you're going to generate buzz when you release them. You'll in fact get a lot of good press and praise for being so open. On Sunday, January 2nd Peldi Guilizzoni (Founder of Balsamiq) [posted](http://www.balsamiq.com/blog/2010/01/03/a-look-back-at-2009/) his company's 2009 revenue ($1.6 million), profits ($1.1 million), number of transactions per month, average revenue per transaction, and cash flow. You can't get more detailed than that. In under 24 hours his blog post received over 200 [re-tweets](http://tweetmeme.com/story/405970649/span-classhighlightbalsamiqspan-company-blog), there were over 140 comments spanned across his blog and [Hacker News](http://news.ycombinator.com/item?id=1028418), and over 250 up votes on Hacker News. That's a lot of attention.

Balsamiq isn't the only startup to share their finances. Squarespace shocked people when they released their figures earlier this year in [Inc Magazine](http://www.inc.com/inc5000/2009/company-profile.html?id=200903390), reporting revenue of $269,545 in 2005 and $2.2 million in 2008 — a 723.3% increase. Nobody knew the jump was so big, and now it's known that they are running away with their market. Even 37signals, who as far as I know had never disclosed their revenue, mentioned kind of nonchalantly in their [third podcast](http://37signals.com/podcast/) what they were generating with their new Haystack service (now known as [Sortfolio](http://sortfolio.com/)).

But who are the people who care about this information? Only two sorts of people really care about these figures: (1) other entrepreneurs who want an inspiring story to motivate them, and (2) potential acquirers whom you could interest by other means. Who doesn't care? Most of your customers or potential customers, who don't fall into either of those categories.

Suffice it to say that I don't think that the press you're going to receive outside the entrepreneurial community is enough to justify exposing your finances to the public. There are just too many disadvantages in doing so. I don't think it's going to significantly move your bottom line. I'd actually like to see how many referrals Peldi Guilizzoni generated from his post. I doubt too many.

**What are the Disadvantages?**

What do these three companies have in common? They're all self-funded startups that are all leading their respective markets with no immediate competitive threats. There's no better small business software company than 37signals, Balsamiq is clearly leading the wire-framing and mockups business, and Squarespace has a chokehold on providing managed website building tools. They have nothing to lose. But suppose they had major competition?

One disadvantage of disclosing assets would be if you're a startup hoping to raise money; even bootstrapped startups consider fund-raising after they get their feet wet. I can assure you that openly talking about your revenue if it's not mind-blowing is going to turn off investors. If you show that your company did $100,000 a year in revenue last year and increased to $150,000 this year, then fewer investors are going to be interested in chatting with you. That's just not enough growth. As soon as you publicly reveal your finances you're looked at differently than you were before.

[Jason Fried](http://twitter.com/jasonfried) and many others have argued that many startups don't try to generate revenue from the beginning because that prejudices their valuation when they do try to raise money. The investor can look at your numbers when the time comes and get a true valuation of the company rather than making up ballpark figures based on speculation and spreadsheets. I completely agree.

Another disadvantage is that your competitors know exactly how much you're making. If I know that one of my competitors is making $2.5 mil a year I can work out a lot of information from those figures. In Balsamiq's case they included everything but the kitchen sink, so I can see nearly everything I need to know.

You might say: "Well, how is that information going to help you surpass them?" For one thing, you may be seen as less as of a threat if your competitor doesn't know how much revenue you're generating. I'd bet that most companies undervalue their competition, so they may not try as hard in certain areas to combat them. Why give them the motivation of knowing how much you make? Additionally, it may spawn new competitors who see how well you're doing.

The third disadvantage is that it may be off-putting to your customers. In the case of Carbonmade specifically, our customers are mainly creative people who don't make a lot of money (for the most part), and shoving our finances in their face isn't the most neighborly act. It can also make you seem impersonal and corporate — which is why so many bank ads say they're small and user-friendly. Yes, publishing our revenue could show that we're a healthy and thriving company that's not going away any time soon, but I think there are more subtle ways to express this.

**Don't Do it**

I am all for being open about your business. You only have to read through my essays to see that. [Carbonmade](http://www.carbonmade.com/) even boldly shares our user count on our homepage (many won't even do this), but I don't think the advantages outweigh the disadvantages when it comes to sharing financial information.

If you do decide to publish your revenue, only do it if you're well established and far beyond the startup stage, like Balsamiq, 37signals and Squarespace. Don't do it if you're hoping to raise money or if you're not (or not certain that you are) the leader in your market. These three companies are the exception to the rule.

**Understanding Windows 7's 'GodMode'**

Although its name suggests perhaps even grander capabilities, Windows enthusiasts are excited over the discovery of a hidden "GodMode" feature that lets users access all of the operating system's control panels from within a single folder.

By creating a new folder in [Windows 7](http://www.cnet.com/windows-7/) and renaming it with a certain text string at the end, users are able to have a single place to do everything from changing the look of the mouse pointer to making a new hard-drive partition.

The trick is also said to work in Windows Vista, although [some are warning](http://www.sitepoint.com/blogs/2010/01/05/windows-7-god-mode/) that although it works fine in 32-bit versions of Vista, it can cause 64-bit versions of that operating system to crash.

To enter "GodMode," one need only create a new folder and then rename the folder to the following:

GodMode.{ED7BA470-8E54-465E-825C-99712043E01C}

Once that is done, the folder's icon will change to resemble a control panel and will contain dozens of control options. I'm not sure it's my idea of playing God, but it is a handy way to get to all kinds of controls.

I've asked Microsoft for more details on the feature and how it came to be. But so far, Redmond is silent on the topic.

[**10 Advanced PHP Tips Revisited**](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

Here, on the Smashing Editorial team, we always try to meet the expectations of our readers. We do our best to avoid misunderstandings, and we try to spread knowedge and present only the best design practices and development techniques. However, sometimes we do make mistakes. And when we do, we apologize and do our best to correct what we’ve done.

In November 2008 we published the article [10 Advanced PHP Tips To Improve Your Programming](http://smashingmagazine.com/2008/11/18/10-advanced-php-tips-to-improve-your-progamming/). Apparently, according to negative comments to the post, it contained some errors and some statements that are just wrong. **We sincerely apologize for our mistake**, and we are truly sorry for any inconvenience we caused by it. However, this simple apology is not good enough. To solve the problem, we asked Chris Shiflett and Sean Coates, two PHP gurus, to take a closer look at the article, explain its errors and make it perfectly clear what is actually right and wrong in the theory and practice. This article is a professional response to our article published a couple of months ago.

**10 Useful PHP Tips Revisited**

This article is a rebuttal to [10 Advanced PHP Tips To Improve Your Programming](http://smashingmagazine.com/2008/11/18/10-advanced-php-tips-to-improve-your-progamming/) — henceforth referred to as the previous article — published last November here on Smashing Magazine. The introduction sounds intriguing:

Listed below are 10 excellent techniques that PHP developers should learn and use every time they program.

Unfortunately, the intrigue devolves into disappointment. We disagree with many of the tips, and even when we don’t, the accompanying explanation is weak or misleading. In this article, we go through each and every tip from the previous article and provide our own commentary and evidence, either to validate and clarify the tip, or to refute it. Our hope is that you don’t just accept our opinion, but rather learn enough to form your own.

**1. Use an SQL Injection Cheat Sheet**

This particular tip is just a link to a useful resource with no discussion on how to use it. Studying various permutations of one specific attack can be useful, but your time is better spent learning how to safeguard against it. Additionally, there is much more to Web app security than SQL injection. [XSS (Cross-Site Scripting)](http://shiflett.org/articles/cross-site-scripting) and [CSRF (Cross-Site Request Forgeries)](http://shiflett.org/articles/cross-site-request-forgeries), for example, are at least as common and at least as dangerous.

We can provide some much-needed context, but because we don’t want to focus too much on one attack, we’ll first take a step back. Every developer should be familiar with good security practices, and apps should be designed with these practices in mind. A fundamental rule is to never trust data you receive from somewhere else. Another rule is to escape data before you send it somewhere else. Combined, these rules can be simplified to make up a basic tenet of security: filter input, escape output (FIEO).

The root cause of SQL injection is a failure to escape output. More specifically, it is when the distinction between the format of an SQL query and the data used by the SQL query is not carefully maintained. This is common in PHP apps that construct queries as follows:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | | <?php | | | |
| 2 | |  | | |
| 3 | | | | $query = "SELECT \* | | | |
| 4 | | | | FROM   users | | | | |
| 5 | | | | | | | WHERE  name = '{$\_GET['name']}'"; | | | |
| 6 | | | | | | |  | | |
| 7 | ?> | |

In this case, the value of $\_GET['name'] is provided by another source, the user, but it is neither filtered nor escaped.

Escaping preserves data in a new context. The emphasis on escaping output is a reminder that data used outside of your Web app needs to be escaped, else it might be misinterpreted. By contrast, filtering ensures that data is valid before it’s used. The emphasis on filtering input is a reminder that data originating outside of your Web app needs to be filtered, because it cannot be trusted.

Assuming we're using MySQL, the SQL injection vulnerability can be mitigated by escaping the name with mysql\_real\_escape\_string(). If the name is also filtered, there is an additional layer of security. (Implementing multiple layers of security is called "defense in depth" and is a very good security practice.) The following example demonstrates filtering input and escaping output, with naming conventions used for code clarity:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01 | <?php | | | |
| 02 |  | | |
| 03 | // Initialize arrays for filtered and escaped data, respectively. | | | | | | | | | | | | | | | | | | | | |
| 04 | $clean = array(); | | | | | | | | | | | | | | | | |
| 05 | $sql = array(); | | | | | |
| 06 |  | | | | |
| 07 | // Filter the name. (For simplicity, we require alphabetic names.) | | | | | | | | | | | | | | | | | | | | | |
| 08 | if (ctype\_alpha($\_GET['name'])) { | | | | | | | | | | | | | | | | | | |
| 09 | $clean['name'] = $\_GET['name']; | | | | | | | | | | | | | |
| 10 | } else { | | | | | | | | | | |
| 11 | // The name is invalid. Do something here. | | | | | | | | | | | | | | | | | |
| 12 | } | | | | | | | | | | | | |
| 13 |  | | | | | | |
| 14 | // Escape the name. | | | | | | | | |
| 15 | $sql['name'] = mysql\_real\_escape\_string($clean['name']); | | | | | | | | | | | | | | | | | | | |
| 16 |  | | | | | | | | | | | | | | | |
| 17 | // Construct the query. | | | | | | | | | |
| 18 | $query = "SELECT \* | | | | | | | |
| 19 | FROM   users | | | | | | | | | | | |
| 20 | WHERE  name = '{$sql['name']}'"; | | | | | | | | | | | | | | |
| 21 |  |
| 22 | ?> | |

Although the use of naming conventions can help you keep up with what has and hasn't been filtered, as well as what has and hasn't been escaped, a much better approach is to use prepared statements. Luckily, with PDO, PHP developers have a universal API for data access that supports prepared statements, even if the underlying database does not.

Remember, SQL injection vulnerabilities exist when the distinction between the format of an SQL query and the data used by the SQL query is not carefully maintained. With prepared statements, you can push this responsibility to the database by providing the query format and data in distinct steps:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |
| --- | --- | --- | --- |
| 01 | <?php | | |
| 02 |  | |
| 03 | // Provide the query format. | | | |
| 04 | $query = $db->prepare('SELECT \* | | | | | |
| 05 | FROM   users | | | | |
| 06 | WHERE  name = :name'); | | | | | | | | |
| 07 |  | | | | | | |
| 08 | // Provide the query data and execute the query. | | | | | | | | | |
| 09 | $query->execute(array('name' => $clean['name'])); | | | | | | | | | | |
| 10 |  | | | | | | | |
| 11 | ?> |

The [PDO manual page](http://php.net/pdo) provides more information and examples. Prepared statements offer the strongest protection against SQL injection.

**2. Know the Difference Between Comparison Operators**

This is a good tip, but it is missing a practical example that demonstrates when a non-strict comparison can cause problems.

If you use strpos() to determine whether a substring exists within a string (it returns FALSE if the substring is not found), the results can be misleading:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 01 | <?php | | | | |
| 02 |  | | | |
| 03 | $authors = 'Chris & Sean'; | | | | | | | | |
| 04 |  | | | | | |
| 05 | if (strpos($authors, 'Chris')) { | | | | | | | | | |
| 06 | echo 'Chris is an author.'; | | | | | | |
| 07 | } else { | | | | | | | |
| 08 | echo 'Chris is not an author.'; | | | | | | | | | | |
| 09 | } | | |
| 10 |  |
| 11 | ?> | |

Because the substring Chris occurs at the very beginning of Chris & Sean, strpos() correctly returns 0, indicating the first position in the string. Because the conditional statement treats this as a Boolean, it evaluates to FALSE, and the condition fails. In other words, it looks like Chris is not an author, but he is!

This can be corrected with a strict comparison:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | | | <?php | | | | | |
| 2 | | |  | | | |
| 3 | | | | | | | | | | if (strpos($authors, 'Chris') !== FALSE) { | | | |
| 4 | | | | | | | | | | echo 'Chris is an author.'; | |
| 5 | | | | | | | | } else { | | |
| 6 | | | | | | | | echo 'Chris is not an author.'; | | | | |
| 7 | } | | | | |
| 8 |  | | |
| 9 | | ?> | | |

**3. Shortcut the else**

This tip accidentally stumbles upon a useful practice, which is to always initialize variables before you use them. Consider a conditional statement that determines whether a user is an administrator based on the username:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | | | <?php | | | | | | |
| 2 | | |  | | | | |
| 3 | | | | | | | | | if (auth($username) == 'admin') { | | | | |
| 4 | | | | | | | | | $admin = TRUE; | | | |
| 5 | | | | | | } else { | | | | |
| 6 | | | | | | $admin = FALSE; | | | | | |
| 7 | } | | | | | |
| 8 |  | | |
| 9 | | ?> | | |

This seems safe enough, because it’s easy to comprehend at a glance. Imagine a slightly more elaborate example that sets variables for name and email as well, for convenience:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 01 | <?php | | | | |
| 02 |  | | | |
| 03 | if (auth($username) == 'admin') { | | | | | | | | | |
| 04 | $name = 'Administrator'; | | | | | | |
| 05 | $email = 'admin@example.org'; | | | | | | | | | | | |
| 06 | $admin = TRUE; | | | | | | | |
| 07 | } else { | | | | | | | | | | | | | |
| 08 | /\* Get the name and email from the database. \*/ | | | | | | | | | | | | | | | | | |
| 09 | $query = $db->prepare('SELECT name, email | | | | | | | | | | | | | | | |
| 10 | FROM   users | | | | | | | | | | |
| 11 | WHERE  username = :username'); | | | | | | | | | | | | | | |
| 12 | $query->execute(array('username' => $clean['username'])); | | | | | | | | | | | | | | | | | | |
| 13 | $result = $query->fetch(PDO::FETCH\_ASSOC); | | | | | | | | | | | | | | | | |
| 14 | $name = $result['name']; | | | | | | | | | | | | |
| 15 | $email = $result['email']; | | | | | | | | |
| 16 | $admin = FALSE; | | | | | |
| 17 | } | | |
| 18 |  |
| 19 | ?> | |

Because $admin is still always explicitly set to either TRUE or FALSE, all is well, but if a developer later adds an elseif, there’s an opportunity to forget:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01 | <?php | | | |
| 02 |  | | |
| 03 | if (auth($username) == 'admin') { | | | | | | | | | | | | | | | |
| 04 | $name = 'Administrator'; | | | | | | | | | |
| 05 | $email = 'admin@example.org'; | | | | | | | | | | | | | | | | | |
| 06 | $admin = TRUE; | | | | | | | | | | |
| 07 | } elseif (auth($username) == 'mod') { | | | | | | | | | | | | | | | | | | | |
| 08 | $name = 'Moderator'; | | | | | | | | | | | |
| 09 | $email = 'mod@example.org'; | | | | | | | | | | | | | |
| 10 | $moderator = TRUE; | | | | | | |
| 11 | } else { | | | | | | | | |
| 12 | /\* Get the name and email. \*/ | | | | | | | | | | | | | | |
| 13 | $query = $db->prepare('SELECT name, email | | | | | | | | | | | | | | | | | | | | | |
| 14 | FROM   users | | | | | | | | | | | | | | | | |
| 15 | WHERE  username = :username'); | | | | | | | | | | | | | | | | | | | | |
| 16 | $query->execute(array('username' => $clean['username'])); | | | | | | | | | | | | | | | | | | | | | | | |
| 17 | $result = $query->fetch(PDO::FETCH\_ASSOC); | | | | | | | | | | | | | | | | | | | | | | |
| 18 | $name = $result['name']; | | | | | | | | | | | | | | | | | | |
| 19 | $email = $result['email']; | | | | | | | | | | | | |
| 20 | $admin = FALSE; | | | | | |
| 21 | $moderator = FALSE; | | | | | | | |
| 22 | } | | | | |
| 23 |  |
| 24 | ?> | |

If a user provides a username that triggers the elseif condition, $admin is not initialized. This can lead to unwanted behavior, or worse, a security vulnerability. Additionally, a similar situation now exists for $moderator, which is not initialized in the first condition.

By first initializing $admin and $moderator, it’s easy to avoid this scenario altogether:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 01 | <?php | | | | |
| 02 |  | | | |
| 03 | $admin = FALSE; | | | | | |
| 04 | $moderator = FALSE; | | | | | | | | |
| 05 |  | | | | | | | | | | | |
| 06 | if (auth($username) == 'admin') { | | | | | | | | | | | | | | | | |
| 07 | $name = 'Administrator'; | | | | | | | | | | | | |
| 08 | $email = 'admin@example.org'; | | | | | | | | | | | | | | | | | | |
| 09 | $admin = TRUE; | | | | | | | | | | | | | |
| 10 | } elseif (auth($username) == 'mod') { | | | | | | | | | | | | | | | | | | | | |
| 11 | $name = 'Moderator'; | | | | | | | | | | |
| 12 | $email = 'mod@example.org'; | | | | | | | | | | | | | | | |
| 13 | $moderator = TRUE; | | | | | | | | | |
| 14 | } else { | | | | | | |
| 15 | /\* Get the name and email. \*/ | | | | | | | | | | | | | | | | | |
| 16 | $query = $db->prepare('SELECT name, email | | | | | | | | | | | | | | | | | | | | | | |
| 17 | FROM   users | | | | | | | | | | | | | | | | | | | |
| 18 | WHERE  username = :username'); | | | | | | | | | | | | | | | | | | | | | | | |
| 19 | $query->execute(array('username' => $clean['username'])); | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | $result = $query->fetch(PDO::FETCH\_ASSOC); | | | | | | | | | | | | | | | | | | | | | |
| 21 | $name = $result['name']; | | | | | | | |
| 22 | $email = $result['email']; | | | | | | | | | | | | | | |
| 23 | } | | |
| 24 |  |
| 25 | ?> | |

Regardless of what the rest of the code does, it’s now clear that $admin is FALSE unless it is explicitly set to something else, and the same is true for $moderator. This also hints at another good security practice, which is to fail safely. The worst that can happen as a result of not modifying $admin or $moderator in any of the conditions is that someone who is an administrator or moderator is not treated as one.

If you want to shortcut something, and you’re feeling a little disappointed that our example includes an else, we have a bonus tip that might interest you. We’re not certain it can be considered a shortcut, but we hope it’s helpful nonetheless.

Consider a function that determines whether a user is authorized to view a particular page:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 01 | <?php | | | | | |
| 02 |  | | | |
| 03 | function authorized($username, $page) { | | | | | | | | | | | | | | | |
| 04 | if (!isBlacklisted($username)) { | | | | | | | | | | | | |
| 05 | if (isAdmin($username)) { | | | | | | | | | | | | | |
| 06 | return TRUE; | | | | | | | | | | | |
| 07 | } elseif (isAllowed($username, $page)) { | | | | | | | | | | | | | | | | |
| 08 | return TRUE; | | | | | | | | | | | | | | |
| 09 | } else { | | | | | | | | |
| 10 | return FALSE; | | | | | | | | | | |
| 11 | } | | | | |
| 12 | } else { | | | | | | |
| 13 | return FALSE; | | | | | | | | | |
| 14 | } | | | | | | | |
| 15 | } | | |
| 16 |  |
| 17 | ?> | |

This example is actually pretty simple, because there are only three rules to consider: administrators are always allowed access; those who are blacklisted are never allowed access; and isAllowed() determines whether anyone else has access. (A special case exists when an administrator is blacklisted, but that is an unlikely possibility, so we’re ignoring it here.) We use functions for the rules to keep the code simple and to focus on the logical structure.

There are numerous ways this example can be improved. If you want to reduce the number of lines, a compound conditional can help:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 01 | <?php | | | | | |
| 02 |  | | | |
| 03 | function authorized($username, $page) { | | | | | | | | | | | | |
| 04 | if (!isBlacklisted($username)) { | | | | | | | | | | | |
| 05 | if (isAdmin($username) || isAllowed($username, $page)) { | | | | | | | | | | | | | | |
| 06 | return TRUE; | | | | | | | | | | | | | |
| 07 | } else { | | | | | | | | |
| 08 | return FALSE; | | | | | | | | | | |
| 09 | } | | | | |
| 10 | } else { | | | | | | |
| 11 | return FALSE; | | | | | | | | | |
| 12 | } | | | | | | | |
| 13 | } | | |
| 14 |  |
| 15 | ?> | |

In fact, you can reduce the entire function to a single compound conditional:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 01 | <?php | | | | |
| 02 |  | | | |
| 03 | function authorized($username, $page) { | | | | | | | | | |
| 04 | if (!isBlacklisted($username) && (isAdmin($username) || isAllowed($username, $page)) { | | | | | | | | | | |
| 05 | return TRUE; | | | | | | | |
| 06 | } else { | | | | | |
| 07 | return FALSE; | | | | | | | | |
| 08 | } | | | | | | |
| 09 | } | | |
| 10 |  |
| 11 | ?> | |

Finally, this can be reduced to a single return:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | | | <?php | | | | |
| 2 | | |  | | | |
| 3 | | | | | | | | | function authorized($username, $page) { |
| 4 | | | | | | | | | return (!isBlacklisted($username) && (isAdmin($username) || isAllowed($username, $page)); | |
| 5 | } | | | | |
| 6 |  | | |
| 7 | | ?> | | |

If your goal is to reduce the number of lines, you’re done. However, note that we’re using isBlacklisted(), isAdmin(), and isAllowed() as placeholders. Depending on what’s involved in making these determinations, reducing everything to a compound conditional may not be as attractive.

This brings us to our tip. A return immediately exits the function, so if you return as soon as possible, you can express these rules very simply:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 01 | <?php | | | | | |
| 02 |  | | | | |
| 03 | function authorized($username, $page) { | | | | | | | | | | | | | |
| 04 |  | | | | | | | | | | |
| 05 | if (isBlacklisted($username)) { | | | | | | | | | | | |
| 06 | return FALSE; | | | | | | | |
| 07 | } | | | |
| 08 |  | |
| 09 | if (isAdmin($username)) { | | | | | | | | |
| 10 | return TRUE; | | | | | | |
| 11 | } | | | |
| 12 |  | |
| 13 | return isAllowed($username, $page); | | | | | | | | | | | | |
| 14 | } | | | | | | | | | |
| 15 |  |
| 16 | ?> | | |

This uses more lines of code, but it’s very simple and unimpressive (we’re proudest of our code when it’s the least impressive). More importantly, this approach reduces the amount of context you must keep up with. For example, as soon as you’ve determined whether the user is blacklisted, you can safely forget about it. This is particularly helpful when your logic is more complicated.

**4. Drop Those Brackets**

Based on the content of this tip, we believe the author means "braces," not brackets. "Curly brackets" may mean braces to some, but "brackets" universally means "square brackets."

This tip should be unconditionally ignored. Without braces, readability and maintainability are damaged. Consider a simple example:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | | <?php | | | | | |
| 2 | |  | |
| 3 | | | | | | if (date('d M') == '21 May') | | | |
| 4 | | | | | | $birthdays = array('Al Franken', | | | | | | |
| 5 | | | | | 'Chris Shiflett', | | | | | | |
| 6 | | | | | 'Chris Wallace', | | | |
| 7 | | | | | | | 'Lawrence Tureaud'); | | | | | | |
| 8 | | | | | | |  | | | |
| 9 | ?> | |

If you’re good enough, smart enough, secure enough, notorious enough, or pitied enough, you might want to party on the 21st of May:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01 | <?php | | | |
| 02 |  | | |
| 03 | if (date('d M') == '21 May') | | | | | |
| 04 | $birthdays = array('Al Franken', | | | | | | | | |
| 05 | 'Chris Shiflett', | | | | | | | |
| 06 | 'Chris Wallace', | | | | |
| 07 | 'Lawrence Tureaud'); | | | | | | | | | |
| 08 | party(TRUE); | | | | | | |
| 09 |  |
| 10 | ?> | |

Without braces, this simple addition causes you to party every day. Perhaps you have the stamina for it, so the mistake is a welcome one. Hopefully, the silly example doesn’t detract from the point, which is that the excessive partying is an unintended side effect.

In order to promote the practice of dropping braces, the previous article uses short examples such as the following:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | | <?php | | | | |
| 2 | |  | | |
| 3 | | | | | | if ($gollum == 'halfling') $height --; | | |
| 4 | | | | | | else $height ++; | |
| 5 |  | |
| 6 | ?> | | |

Because each condition is constrained to a single line, such mistakes might be less likely, but this leads to another problem: inconsistencies are jarring and require more time to read and comprehend. Consistency is such a valued quality that developers often abide by a coding standard even if they dislike the coding standard itself.

We recommend always using braces:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 01 | <?php | | | | |
| 02 |  | | | |
| 03 | if (date('d M') == '21 May') { | | | | | | |
| 04 | $birthdays = array('Al Franken', | | | | | | | | | |
| 05 | 'Chris Shiflett', | | | | | | | | |
| 06 | 'Chris Wallace', | | | | | |
| 07 | 'Lawrence Tureaud'); | | | | | | | | | | |
| 08 | party(TRUE); | | | | | | | |
| 09 | } | | |
| 10 |  |
| 11 | ?> | |

You’re welcome to party every day, but make sure it’s deliberate, and please be sure to invite us!

**5. Favor str\_replace() Over ereg\_replace() and preg\_replace()**

We hate to sound disparaging, but this tip demonstrates the sort of misunderstanding that leads to the same misuse it’s trying to prevent. It’s an obvious truth that string functions are faster at string matching than regular expression functions, but the author’s attempt to draw a corollary from this fails miserably:

If you’re using regular expressions, then ereg\_replace() and preg\_replace() will be much faster than str\_replace().

Because str\_replace() does not support pattern matching, this statement makes no sense. The choice between string functions and regular expression functions comes down to which is fit for purpose, not which is faster. If you need to match a pattern, use a regular expression function. If you need to match a string, use a string function.

**6. Use Ternary Operators**

The benefit of the ternary operator is debatable (there’s only one, by the way). Here is a line of code from an audit we performed recently:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | | <?php | | |
| 2 | |  | |
| 3 | | | | | | $host = strlen($host) > 0 ? $host : htmlentities($host); | |
| 4 | | | | | |  |
| 5 | ?> | |

Oops! The author actually means to escape $host if the string length is greater than zero, but instead accidentally does the opposite. Easy mistake to make? Maybe. Easy to miss during a code audit? Certainly. Concision doesn’t necessarily make the code any better.

The ternary operator may be fine for one-liners, prototypes, and templates, but we strongly believe that an ordinary conditional statement is almost always better. PHP is descriptive and verbose. We think code should be, too.

**7. Memcached**

Disk access is slow. Network access is slow. Databases typically use both.

Memory is fast. Using a local cache avoids the overhead of network and disk access. Combine these truths and you get memcached, a “distributed memory object caching system” originally developed for the Perl-based blogging platform LiveJournal.

If your application isn’t distributed across multiple servers, you probably don’t need memcached. Simpler caching approaches — serializing data and storing it in a temporary file, for example — can eliminate a lot of redundant work on each request. In fact, this is the sort of low-hanging fruit we consider when helping our clients tune their apps.

One of the easiest and most universal ways to cache data in memory is to use the shared memory helpers in [APC](http://pecl.php.net/apc), a caching system originally developed by our colleague George Schlossnagle. Consider the following example:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 01 | <span><?php | | | | |
| 02 |  | | | |
| 03 | $feed = apc\_fetch('news'); | | | | | | | |
| 04 |  | | | | | |
| 05 | if ($feed === FALSE) { | | | | | | | | | | | |
| 06 | $feed = file\_get\_contents('<a href="http://example.org/news.xml">[http://example.org/news.xmlHYPERLINK "http://example.org/news.xml%3C/a"<HYPERLINK "http://example.org/news.xml%3C/a"/a](http://example.org/news.xml%3C/a)>'); | | | | | | | | | | | | |
| 07 | // Store this data in shared memory for five minutes. | | | | | | | | | | |
| 08 | apc\_store('news', $feed, 300); | | | | | | | | | |
| 09 | } | |
| 10 |  |
| 11 | // Do something with $feed. | | | | | | | | |
| 12 |  | | | | | | |
| 13 | ?></span> | | |

With this type of caching, you don’t have to wait on a remote server to send the feed data for every request. Some latency is incurred — up to five minutes in this example — but this can be adjusted to as close to real time as your app requires.

**8. Use a Framework**

All decisions have consequences. We appreciate frameworks — in fact, the main developers behind [CakePHP](http://cakephp.org/) and [Solar](http://solarphp.com/) work with us at OmniTI — but using one doesn’t magically make what you’re doing better.

In December, our colleague Paul Jones wrote an article for PHP Advent called [The Framework as Franchise](http://phpadvent.org/2008/the-framework-as-franchise-by-paul-jones), in which he compares frameworks to business franchises. He refers to a suggestion by Michael Gerber from his book "The E-Myth Revisited":

Gerber notes that to run a successful business, the entrepreneur needs to act as if he is going to sell his business as a franchise prototype. It is the only way the business owner can make the business operate without him being personally involved in every decision.

This is good advice. Whether you’re using a framework or defining your own standards and conventions, it’s important to consider the value from the perspective of future developers.

Although we would love to give you a universal truth, extending this idea to suggest that a framework is always appropriate isn’t something we’re willing to do. If you ask us whether you should use a framework, the best answer we could give is, “It depends.”

**9. Use the Suppression Operator Correctly**

Always try to avoid using the error suppression operator. In the previous article, the author states:

The @ operator is rather slow and can be costly if you need to write code with performance in mind.

Error suppression is slow. This is because PHP dynamically changes error\_reporting to 0 before executing the suppressed statement, then immediately changes it back. This is expensive.

Worse, using the error suppression operator makes it difficult to track down the root cause of a problem.

The previous article uses the following example to support the practice of assigning a variable by reference when it is unknown if $albus is set:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | | <?php | | | |
| 2 | |  | | |
| 3 | | | | $albert =& $albus; | | | |
| 4 | | | |  | | |
| 5 | ?> | |

Although this works — for now — relying on strange, undocumented behavior without a very good understanding of why it works is a good way to introduce bugs. Because $albert is assigned to $albus by reference, future modifications to $albus will also modify $albert.

A much better solution is to use isset(), with braces:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | | | <?php | | | | | |
| 2 | | |  | | | | |
| 3 | | | | | | if (!isset($albus)) { | | | | |
| 4 | | | | | | $albert = NULL; | | | |
| 5 | } | | | | | |
| 6 |  | | |
| 7 | | ?> | | |

Assigning $albert to NULL is the same as assigning it to a nonexistent reference, but being explicit greatly improves the clarity of the code and avoids the referential relationship between the two variables.

If you inherit code that uses the error suppression operator excessively, we’ve got a bonus tip for you. There is a new PECL extension called [Scream](http://pecl.php.net/package/scream) that disables error suppression.

**10. Use isset() Instead of strlen()**

This is actually a neat trick, although the previous article completely fails to explain it. Here is the missing example:

[view source](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

[printHYPERLINK "http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/"?](http://www.smashingmagazine.com/2009/03/24/10-useful-php-tips-revisited/)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | | | <?php | | | | |
| 2 | | |  | | | |
| 3 | | | | | | | | | if (isset($username[5])) { |
| 4 | | | | | | | | | // The username is at least six characters long. | |
| 5 | } | | | | |
| 6 |  | | |
| 7 | | ?> | | |

When you treat strings as arrays, each character in the string is an element in the array. By determining whether a particular element exists, you can determine whether the string is at least that many characters long. (Note that the first character is element 0, so $username[5] is the sixth character in $username.)

The reason this is slightly faster than strlen() is complicated. The simple explanation is that strlen() is a function, and isset() is a language construct. Generally speaking, calling a function is more expensive than using a language construct.