The No Bullshit Bible : Creating Web 2.0 Startups & Programming

Written by James Gillmore
Edited by Holly Welch

FaceySpacey
www.faceyspacey.com

Process

FaceySpacey Bible - The No Bullshit Bible: Creating Web 2.0 Startups & Programming

TABLE OF CONTENTS

Non-Technical

Chapter 9

9-1 PROCESS DEVELOPMENT TOOLS | VERSION CONTROL (Mercurial)

So if you've read the Server Setup tutorials, you've heard me mention "version control." Version control has typically been called a "time machine" to look at your code. What that means is you can look at your code at any moment time as it has been developed. You can do things like revert back in time to what your code looked like at a previous point in time, i.e. "roll back" your code to how it was last week. Or, just look at what your code was a while ago before you altered it. It serves the purpose of giving you confidence to do things like delete and replace old code with new and better solutions. So in the case that the new code isn't quite right, you can go back and look at how it was before. It also greatly aids multi-developer teams. It allows each developer to work with an unchanging set of code. So that means if another developer changes something, it won't break your code. However, there will be a time when you have to merge your code with the code of other developers. What this does is define a clear cycle of coding without worrying about changes, and then merging with everyone else. When you merge, your version control system will try to do it automatically, but if there are conflicts it will compare your code with the code of other developers and allow you to make edits so that you don't break each other's code. That's the idea.

The most popular version control systems these days are SVN, Git and Mercurial. We, at FaceySpacey, use Mercurial. SVN is a little older. Git and Mercurial are very comparable. They allow for what's called "distributed version control." I won't go into what that means too much right now, but I will say that Git has been compared to the *MacGyver* of version control and Mercurial to the *James Bond* of version control. What that means is Git has more features (and therefore more to learn and figure out), whereas Mercurial has less features but is easier to get up and running with. The features it does have are very precise and solve what it is you will actually need most of the time really well. The reason we use Mercurial is because it integrates nicely with our bugtracker, Fogbugz. Fogbugz's makers, Fogcreek, has a complementary product called "Kiln" which works in conjunction with Fogbugz. Basically it allows you to associate committed code to fixed bugs and completed tasks/features within Fogbugz.

So here's how we use Mercurial.

1) Signup for Kiln here:

http://www.fogcreek.com/kiln/

They have a free startup version for 2 developers. Once you create your account, you'll be directed to create your first repository. Create it. Some directions will be provided about how to "push" your code to this storage space on Kiln's servers for your code.

2) Those directions will look like the following basically. At the command line navigate to your code and execute the following commands:

```
# cd /var/www/yoursite.com/

# hg init

# hg add

# hg commit

# hg push https://yoursite.kilnhg.com/repo/project-name/Group/repo-name
```

That will take the code on your server (or your local host if that's where your're working) and send the code to your storage space (i.e. respository) on Kiln's servers. In essence, Kiln will keep a back up of all your code--and of course with all the "time machine" magic previously talked about. In other words, it will store snapshots of your code at all points in time--well, specifically, after all commits. A commit is made every time you type "hg commit" at the command line while within your application directory. If you type "hg commit -m 'some notes'" you can leave some notes about the recent changes you made, and then you can easily see what code is being commited within the Kiln web interface.

The next thing to note is that in the file, */var/www/yoursite.com/.hg/hgrc*, you can permanently enter your Kiln login credentials so that with every commit and push to the Kiln server you don't have to enter the login credentials again:

```
[paths]
default = https://yoursite.kilnhg.com/repo/project-name/Group/repo-name
[auth]
kiln.prefix=https://yoursite.kilnhg.com/repo/project-name/Group/repo-name
kiln.username=your-kiln@email-address.com
kiln.password=yourpassword
[ui]
username= your-kiln@email-address.com
password=yourpassword
```

The /.hg folder fyi may be hidden, so make sure to unhide your hidden files while in your root application directory to see it.

The last thing to do is to navigate to another hidden file `/var/www/yoursite.com/.hgignore` and add the following lines (create this file if it does not exist already):

```
syntax: glob
runtime/**
assets/**
```

Those lines will make it so you don't commit code that is unique to each computer that has the code on it. You may end up having an /uploads folder in your root application directory. If my app has functionality for users to do things like upload photos, i'll make it so those large images are also not committed and pushed. Photos are big files and will make it so your commits/pushes take a long time, and won't be needed on the computers/servers of other developers since they'll have their own.

So that's basically it. One of the biggest benefits of doing this is you don't need to worry about losing all your code if your server goes down. All you're going to have to remember really after it's installed is to type "*hg commit -m 'some message*'" after you write new code, and "*hg push*" to push it to the central repository. And then "*hg pull*" and "*hg update*" which you run to pull in code commits other developers pushed to the main Kiln repository. You'll also at times have to type "*hg merge*" but the beauty of mercurial is that when you run any one of these commands, if there is a problem, Mercurial will let you know at the command line and suggest other commands such as the "hg merge" one that you should execute.

Let me summarize the common flow:

PUSHING YOUR NEW CODE OUT:

```
# hg commit
# hg push
```

PULLING OTHER DEVELOPERS' CODE IN:

```
# hg pull
# hg merge
# hg update
```

That's basically all you need to get productive quickly. When you get serious about using Mercurial, read the following longer tutorial by the Joel Spolsky, the creator of Fogbugz and Kiln (as well as Stackoverflow). He's a prolific software developer that got his reputation from his work at Microsoft, specifically on the Excel project. Joel Spolsky's the man. Research/google him.

http://hginit.com/

9-2 PROCESS DEVELOPMENT TOOLS | IDEs (Netbeans)

When learning about coding and how to code, one of your initial questions will be simply: “Where do I do my coding?” The reality is you can do it in a basic text editor like Notepad on windows, or Gedit on Linux, etc. However, there are many applications, IDEs (Interactive Development Environments), that provide a bunch of extra functionality to make coding easier. Here’s a list of basically all the PHP IDEs:

http://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments#PHP

Our recommendation is Netbeans. Why? Well it’s the one I’ve found to be most popular among Yii developers, plus I’ve tried them all and compared all their features blow for blow, and I found it to be the beast. It has built-in Mercurial support, support for debugging with Xdebug (which we’ll cover in another tutorial, great code completion support, refactoring tools and more.

You can download it here: http://netbeans.org/ . Well, here’s the download page (download the PHP version obviously):

http://netbeans.org/downloads/index.html

Netbeans is open source. It has many plugins to augment its functionality, and its well-maintained with constant improvements.

This is going to be a short tutorial, and it's more of a recommendation. But I will share the main things to expect from Netbeans (and most IDEs):

*1) **Code Completion** - what this will do is display menus as you type code to help you complete your code, i.e. offering hints of what you might type next. When coding, it's not about remembering every function or method, etc, that you may type. It's about generally knowing what you're doing. With the internet at your fingertips, it's all about looking up a solution you know exists (perhaps by reading these tutorials) or even a solution you previously used to remind yourself. Similarly with code completion, you're given the names of possible solutions based on what you're typing. The solutions will generally come from the Yii framework powering your application (or your own code). The idea is that the IDE is knowledgeable about all the methods, properties, etc that lie deep within the Yii library code. So you don't have to worry about remembering it all.*

*2) **Code Hiding** - you can expand/collapse blocks of code that you're not currently working with to make it easier to focus on what you are working with.*

*3) **Code Navigation** - Netbeans provides lists of all the methods and properties in your current PHP class, so you can navigate around the code by browsing a simple list of the main (hopefully informationally named) methods/properties/etc.*

*4) **Refactoring** - you can do things like change the name of a method everywhere it is used in your code by changing its name in one place. And more. The general idea behind refactoring is that you often come up with more concise ways to code solutions after you've already developed a partial solution. Netbeans makes it easy to globally make a change across all your code.*

*5) **Debugging** - Debuggin is the process of executing your code in your web browser while your IDE "steps" through the code. That means you can visit your website, and then watch basically an arrow point to each part of the code as it's executed. What happens is your browser hangs on a white screen, while you watch the sequence of your code execute step by step. This is required tool for all FaceySpacey developers. It allows you to pinpoint where the problem is, rather than see an error page in the browser and wonder where the problem is. I won't go to deep into debugging right now, but just imagine yourself being able to execute a script, and watch what values are being dealt with in just one block of code in perhaps many blocks of code in the execution without having to worry about those other blocks.*

*6) **Formatting** - this is perhaps the most basic feature of any IDE, but a very important one. Netbeans will automatically space and tab words apart from each other in a consistent fashion so your code always is structured similarly and is therefore easier to read.*

*7) **File Browing** - Netbeans makes it very easy to navigate your tree of files in your application, and add new files, etc.*

Netbeans offers a lot more features, but that should give you an idea of what its purpose is. Get it immediately, and don't look back.

9-3 PROCESS DEVELOPMENT TOOLS | PROJECT MANAGEMENT SOFTWARE (Fogbugz)

There are many project management tools to choose from. They're often called simply "bugtrackers." Here's a Wikipedia comparison list of all the tools too choose from:

http://en.wikipedia.org/wiki/Comparison_of_project_management_software

37signals' Basecamp has become the poster child of such software. It's too basic of a tool for real software development in our opinion. Our pick here is Fogcreek's Fogbugz. Here are the main reasons:

*1) **Custom Task/Bug Organization** - Fogbugz allows you to create lists of tasks/bugs and group them in an almost infinite number of ways, i.e. by area, priority, tags, milestone, assignee, and many other criteria. However, that's not the magic. The magic is you can have multiple "views" into your tasks accordion to different criteria. Fogbugz calls these "Filters." In essence, the tasks exist in the system and are tagged with various criteria. Then you can build Filters that sort, group and filter the tasks shown according to a configuration of all that criteria. And then you can build as many of these filters as you want. For example, you can see just all tasks for Project ABC with priority 5+ in the current milestone that are past due. You can then share these filters you create with other developers so you know you're all looking at the same system.*

*2) **Plugins** - Fogbugz has a great plugin system where 3rd party developers can enhance it. One popular once is Kanban. Search in google images "Kanban chart": http://www.google.com/search?q=kanban+chart&hl=en to see what it is. It's basically a chart with several vertical columns, and then you move post-it notes that correspond to tasks across these columns from left to right as they get closer to true completion. So the Kanban plugin helps display your fogbugz tasks in a similar visual.*

*3) **Integration with Mercurial** - Fogbugz's sister product (as mentioned in the Version Control tutorial) Kiln is deeply with Fogbugz. It allows you to associate actual committed code with completed (or near-completed) tasks. It also allows you to do code reviews. It's pretty sick.*

*4) **Evidence-based Scheduling** - Fogbugz can produce pretty graphs that estimate when projects/tasks/milestones will be completed based on the measured efficiency of your developers and what tasks they've already completed. It takes a lot of discipline to use this properly, as you need to do things like estimate how long tasks will take and mark how long they ultimately take you, but if you can get your developers into a groove using this, you can get great estimates of when your milestones will be reached.*

*5) **Wikis** - Fogbugz has wikis that can be two-way linked between tasks. So you can produce documentation for your code or product and closely correlate it to your task.*

*6) **Email Support System** - It comes with an email support system that turns any emails into tasks in your system automatically, and your developers can do things like respond to bugs inside of fogbugz just like they would any other task, and the person that reported the bug (or complain) will get an email. And of course that thread can continue accordingly.*

*7) **Forum System** - It has a forum system that also interlinks with the bugtracker similar to the wiki system.*

*8) **Snapshop plugins** - There are some great plugins that allow testers to take screenshots of issues your app is having and automatically send the screenshot (with some notes and other tagged criteria) to the bugtracker at the core of Fogbugz.*

*9) **Nested Tasks** - Tasks can have sub-tasks for basically an infinite # of tiers (well, i think it's 8 tiers). A lot of bug trackers only have 2 tiers of tasks, i.e. a group and a list of tasks in it. Fogbugz allows you to easily have child and parent tasks, and move tasks around between different parents.*

*10) **Quick Notation** - Fogbugz has a quick way to enter tasks one after another in a quick to do list. This is very powerful for just banging out your task list quickly.*

The list goes on. We highly recommend it. Check out all the features here:

http://www.fogcreek.com/fogbugz

Now I'm going to end this article explaining the importance of bug-tracking and project management software in general. The importance is simple: one of the key skills required of a great developer is storing many to-dos in their head at a time. But the reality is it's impossible to store them all. Project management software allows developers to focus on doing the actual work. They'll inevitably always find sub-tasks they must remember. So help them make it so those tasks they have to remember are very few and at a very granular level for a clear single task. For product owners (i.e. the non-technical participants in the project), it will give you insight into the progress of your developers. It's pretty self-explanatory. The biggest takeaway is just get Fogbugz and don't spend 2 weeks researching all the bugtrackers like I and many startup guys have done. It is sort of a right of passage to study them all for any Web 2.0 guru, but really, just get to coding your app and take my word for it: Fogbugz is the shit!

9-4 PROCESS TESTING 1 | DEBUGGING (Xdebug)

Debugging is perhaps the most important part of software development. If you're not a coder yet, and you manage some coders, you may be inclined to measure their productivity by how many lines of code they've coded, but the reality is most of their time is spent hunting down bugs and often changing one line of code. So therefore tools to track down those bugs are very important.

Debugging tools allows you to isolate the problem area and "step" through its lines of code to see quickly find the precise problem. You can do things like see the values of each variable at each line of code until you see where the value is not what it should be.

The best and most popular tool for PHP debugging is called XDebug. You can check it out here:http://xdebug.org/ .

Installing Xdebug can be quite problematic. Here's the best tutorial I've found, and exactly what I used to install it: http://devzone.zend.com/article/2803

The basic idea is you install the Xdebug application on your server, and then tell PHP it can use it by configuring you php.ini file you've seen edited in the Setup tutorials.

The next thing you need to do is configure Netbeans to make use of it. Here are the tutorials to do so:

http://wiki.netbeans.org/HowToConfigureXDebug

http://netbeans.org/kb/docs/php/debugging.html

When completely installed, which again is the hardest part, you finally get to learn the joys of debugging. The main idea is you can watch your script execute while the browser hangs and basically press pause/play buttons to step through your code line by line, while viewing in a window the values of all the variables in the current scope.

You can do things like dive deeper into called functions or skip over them and just step through the current top level client code. You can go as many layers deep as you want

and back out as many layers back as you want. So that means if a function or method is called, you can choose to step through all the lines within it, or just treat it as one line that you skip over and move to the next, i.e. if you deem it not to be problematic. But if you determine that a specific method call is where the problem is coming from, you can dive into it and go line by line through its code while watching the values of variables used within it.

Other things you can do are create “breakpoints” which are points in the code sequence the debugger should stop at while it’s playing through your code. Think of the debugger like a music track. You can basically say at 1 minute 33 seconds pause, or rather at line 77 pause. And then take a quick peak at what’s going on. You can set several break points, and just pause/play until you reach them, without having to waste your time stepping through other lines you’re pretty sure are kosher.

You can also specify “watches,” which are basically variables and properties, whose values you’d like to watch more closely. Xdebug within Netbeans unfortunately is a little buggy, or rather, is just not capable of tracking all variables. I think it’s because it just takes too much computing power. That’s why it allows you to specify “watches” that will contain the values of variables/properties you’re really interested in.

The last really important thing it does is maintain a “call stack” for you. A call stack is treed outline of all the methods called and the methods they call and so on. The Yii framework will produce similar call stacks when your pages produce errors. The idea is you can track the code execution path in terms of methods called. So one method usually will call several methods in it, and those methods will do the same and so on. And if there is a problem, you’ll be looking at the last method called before the error or exception occurred. When you’re there you can get a list of all the previous methods called to get to that point. This way you can go up and down the tree looking for other

factors that might have caused the problem, without having to examine methods that aren't part of the sequence.

Again, Debugging is very important. Without it, developers do this: they make their code echo the value of variables--often arrays--in their code that they think are causing the problem. When they do so, they stop the script from executing further. To do this they will enter something like*var_dump($variableName); exit;* in their code to make the script echo out to the browser the value contained in *$variableName* on to a white screen, and stop all else. This is problematic because you then have to remove that code, or comment it out, and if you comment it out, you're left with lots of extra code that won't be used when the script is in production and used on a live site. Debugging adds a layer of extra tools to do all this for you without cluttering up your code. Master it asap. If you know Javascript, a good place to start is with the debugger built into the Firebug plugin for Firefox since it takes no installation, and as I said before Xdebug is hard to install.

9-5 PROCESS TESTING | UNIT TESTING (Using Yii Tools)

Unit testing is the testing the of individual code blocks at a granular level. Therefore tools to aid this process make it very easy to isolate these blocks of interest and make sure they're kosher independently from the rest of your application code. "Test Driver Development" is a methadology where you code tests before you even write code. For new developers--that these tutorials are written for--it's really hard to get why it's so important since you usually want to get to coding your features immediately. That's why I won't go too deep into it yet, but I will say that for large teams and apps already in production iterating into post-launch phases it's very important. The reason is because you can run tests that will verify past code you've written still works. As you improve upon your app and add new features, it's inevitable that you will break old

code. With a proper suite of tests for your most important features, you can quickly generate a report that lets you know if anything broke.

The way unit testing works is generally that you define sets of input, then the test passes the input to a block of code, waits for the output, and then tests that the output equals what you want it to equal. You have to write code to setup what the input is, and you have to write code to test the values of the output and if they match your expectations.

Yii offers built-in testing tools (based on PHPUnit). To install them and learn how to use them, read these tutorials on the Yii site:

http://www.yiiframework.com/doc/guide/1.1/en/test.overview

http://www.yiiframework.com/doc/guide/1.1/en/test.unit

With this birds-eye-view overview you should know when this will become important. To me, the biggest thing is the mindset that unit testing cultivates. It helps you imagine your code in parts, rather than features. It lets you visualize the motor, the steering wheel, etc, if we were to use a car metaphor. It helps you pin-point how each part should interface with other parts so that you can get a rock solid interface between sets of code so that less breakdowns happen. It helps you know where the problematic parts will be, i.e. what points of interface may be weak or strong.

In another article I'll cover Functional Testing, but not in this series. However, I'd like to point out the difference: Unit Testing is all about automating the testing of blocks of code, methods basically, while Functional Testing is all about automatically simulating a user on your site and catching any issues at the graphical user interface level. In my experience, doing Functional Testing is a lot more difficult to pull of and without the gains of Unit Testing. Functional testing is very appealing to non-technical product guys that don't trust their developers--at least the overall concept is. Functional testing

is great when your app is truly done, and you want to keep it that way as you add new features. It's not as useful in the development stage and just getting to your initial launch, for which these tutorials are mainly written. It's hard to prepare useful functional tests when your app barely works. Whereas unit testing--or especially test driven development where you write the tests first--is key to giving developers a mindset where they visualize their code in chunks. Overall, unit testing allows developers to really focus on their tasks and guarantee that what they've been assigned to do is as closed to perfect as possible. It requires great product specs, as do most aspects of software development for GUI heavy apps, so that you can also create great tech specs where you pin-point precise pockets of code to write. When a project can be broken down that granularly, unit testing does wonders by helping developers focus on and insure their assignments will interface with the assignments of other developers perfectly, or at least in the way the other developers expected.

9-6 PROCESS TECH CONCEPTS | APIs

An understanding of APIs is key to modern web development. The reason this tutorial is in the "Process" section is because building your application from an API perspective leads to long-lasting code that all the developers in your team can use and less throw-away code. In this tutorial, we'll cover the use of 3rd party APIs to give you an idea of how you can incorporate your own to isolate separate layers of your codebase into clear interfaces.

Prolific East Coast investor (from Union Square Ventures), Fred Wilson, I remember once mentioned in a video that he forecasts the future of the web being all about mashups of APIs. What that basically means it that apps will be getting data from tons of places (currently Facebook, Twitter, etc) and sending data back. These days you can build tools using other tools that are mashups. For example Twitter Feed helps you post to Facebook and Twitter from your RSS feeds. So basically you're going indirectly through another mashup to accomplish your automated posting to the said social

networks. That's a basic example, but the basic idea is still the same: everyone's building stuff on top of other applications, and people are building on top of that and so on.

So what is an API? An API is a programmatic way to send input to another web application and get output back over the internet. So rather than visiting a web page in a URL, your application can visit special URLs that return data in consistent formats that your app can then interpret and use to go about its business. It's a way for one app to communicate to another. This is so powerful that large soon to be public companies like Zynga have made billion dollar businesses using APIs of other companies, i.e. specifically Facebook. Zynga's going to go public before Facebook! Isn't that crazy.

Back to what Fred Wilson was saying. He's basically saying that an internet behind the scenes is emerging. It still uses the HTTP protocol like your web browser does, but its your app code that's doing the communication to accomplish its own unique goals. As a result we have entire companies, like the new Ness (http://www.likeness.com), which do nothing but mine data from other sites and build an intelligence based on that information. Ness, for example, plans to harvest data from tons of places to build a minor form of artificial intelligence that will help you search and get results based on your tastes. A lot of guys have tried to do this to a degree, but it's about time for this to get really real--because of the shear amount of data you're putting out on Twitter, Facebook, iTunes, etc, about what you like.

So how does this actually work in an actual application? Basically you write code to ping a URL, and in the URL you specify for example a user ID of a Facebook user, and you can get their information in a consistent structured format that your code can then parse. For example here is the basic information about one of the most famous Facebook engineers, Bret Taylor:

```
{
  "id": "220439",
  "name": "Bret Taylor",
  "first_name": "Bret",
  "last_name": "Taylor",
  "link": "http://www.facebook.com/btaylor",
  "username": "btaylor",
  "gender": "male",
  "locale": "en_US"
}
```


You can ping the Facebook API at the following endpoint URL to get that data about him:

https://graph.facebook.com/btaylor

That data happens to be public. For private data you would specify basically a password in the URL (that you usually have stored in your database) to get access to that facebook info. We'll cover in another article Facebook authentication, but basically Facebook, Twitter, etc, all provide ways for end users in their browser to pass you a special password ("access tokens") to gain access to their data and post updates/tweets on their behalf.

Back to the structured data above. As you can see the data is presented in a consistent format. All users will have a gender, first_name, etc. It will all be presented in a similar format. Therefore you can write code to iterate over that list of data and grab each value by its key, i.e. you can expect to get all first names by the key, "first_name" and so on. That's the idea.

When you submit data, you also get a response, e.g. that what you posted was successful. For example, if you're using the Twilio API to send SMS text messages you will get a response like this:

```
<TwilioResponse>

   <SMSMessage>
      <Sid>SM90c6fc909d8504d45ecdb3a3d5b3556e</Sid>
      <DateCreated>Wed, 18 Aug 2010 20:01:40 +0000</DateCreated>
      <DateUpdated>Wed, 18 Aug 2010 20:01:40 +0000</DateUpdated>
      <DateSent/>
      <AccountSid>AC5ef872f6da5a21de157d80997a64bd33</AccountSid>
      <To>+14159352345</To>
      <From>+14158141829</From>
      <Body>Jenny please?! I love you &lt;3</Body>
      <Status>success</Status>
      <Direction>outbound-api</Direction>
      <ApiVersion>2010-04-01</ApiVersion>
      <Price/>
<Uri>/2010-04-
01/Accounts/AC5ef872f6da5a21de157d80997a64bd33/SMS/Messages/SM90c6fc909d8504d45ec
db3a3d5b3556e</Uri>
</SMSMessage>

</TwilioResponse>
```

And as you can see the <status> node says it was successful, and your code can now continue as normal because it knows it executed its job of sending an SMS text message successfully. in the <Uri> node you receive an ID # for that SMS text message so you can refer to it later. You could store that in your database and request another API endpoint url and get back the contents of this text message, i.e. similar information to what you see above. Maybe you want to produce a history of all text messages sent through your app--so that's why you would do that.

That's a quick runthrough of what APIs are about. If you plan to post tweets to twitter and updates to Facebook with your App, you're going to want to understand how APIs

work even if you're not going to be a coder. More specifically, you need to be able to read API documentation, e.g. like the one Facebook provides for their "graph API":

http://developers.facebook.com/docs/reference/api/

If you can read the documentation you will know specifically what you can't and can do, and can therefore better instruct your developers of what to develop. You'll avoid sending them on a wild goose chase to code something Facebook won't even allow you to do. I've seen so many failed companies spend ages thinking they can do something they can't, and waste money forcing developers to figure it out on your dime.

To read API documentation you need to understand 3 things:

1) PARAMETERS - the data you provide to the URL endpoints, and what parameters are available and what types of values are accepted. These parameters basically configure what sort of output you should expect. They are the input, just like parameters to functions in PHP.

2) RESPONSE FORMATS - above you saw 2 types of response formats: JSON and XML. That's basically all you'll need to know. It's how the response is formatted. If you can examine these tree structures, you can see what sort of data you can get out of the APIs you're using.

3) POST vs GET - i've been referring to these URL endpoints as the only way to supply input parameters, but the reality is you can pass more parameters outside of the URL via "POST fields." I won't go into it at a deep level here, but I'll give you the gotcha you need to know: basically you can attach more values and send them along with the URL you request. That's it. So instead of having parameters at the end of the URL like this ?key=value&key2=value, you'll send them in similar pairs but as POST fields. In PHP you'll use a tool called cUrl to aid you in attaching these key/value pairs.

After you understand that, you'll be able to read the documentation of very many APIs very quickly. They're all quite similar. Study Facebook's and Twitter's until you get the hang of it.

Conclusion & Further Reading

We at FaceySpacey hope you've enjoyed our FaceySpacey Bible, and are coming away many times more ready to succeed at your next software startup. At the very least, you should have a birds-eye-view of what you need to do to get your startup, and have quelled a lot of insecurities you may have had regarding how you should execute it. That said, I will point to you to what you should do next.

As promised, the following is a list of the precise books I read to master web development using HTML/CSS, Javascript, PHP & MySQL. They are presented in the best order to most efficiently learn the subject at hand. It's similar to the order I read them in, but enhanced based on what I learned and the order I wish I read them in. Good luck:

HTML/CSS:

CSS Mastery: Advanced Web Standards Solutions

http://www.amazon.com/CSS-Mastery-Advanced-Standards-Solutions/dp/1430223979/

Before you start coding PHP, Javascript, etc, understand how HTML works. This is where you start. HTML is easy. Read this book in combination with studying the HTML & CSS tutorials on w3schools.

PHP & MySQL:

PHP & MySQL For Dummies, 4th Edition

http://www.amazon.com/PHP-MySQL-Dummies-Janet-Valade/dp/0470527587

This book--well an older edition--I read a year before I got serious about learning to code. I read it and didn't actually code anything i learned, but what it did was plant seeds in my head with regards to what programming is all about and what databases are all about, and how to connect the two. It assumes very little in what you may

already know, and is an excellent start in your journey to becoming a master programmer.

PHP Object-Oriented Solutions

http://www.amazon.com/PHP-Object-Oriented-Solutions-David-Powers/dp/1430210117

This book is where I learned what OOP is. I didn't get the hang of it until reading the following book. Don't worry if you read this and have a hard time with it. This book and the next each have introductory chapters that go over how OOP works. It took me reading basically this book and the next book about the same stuff to get it. This book is a lot less complicated than the following and dives into practical examples & problems, whereas the next is a lot more theoretical.

PHP Objects, Patterns and Practice

http://www.amazon.com/Objects-Patterns-Practice-Experts-Source/dp/143022925X

After reading this book, I basically mastered OOP. It's a very hard book to get through if you're new to OOP, and goes into some very advanced stuff, specifically tons and tons of "design patterns." The design patterns are presented in as basic of a form as possible, but they weren't very practical like examples from the previous book in that you probably will never actually need any of the code used in the book. Either way, this is my favorite Programming of all time because it taught me how to think like a coder and how to solve complex problems with concise refactored solutions. It really showed me what is possible with OOP. You don't know PHP unless you've read this book.

Pro PHP: Patterns, Frameworks, Testing and More

http://www.amazon.com/Pro-PHP-Patterns-Frameworks-Testing/dp/1590598199

I read this book too just to solidify my experience with PHP, and cover all my bases. Check it out. It's optional.

PHP Functions Essential Reference

http://www.amazon.com/Functions-Essential-Reference-Torben-Wilson/dp/073570970X

At some point during my study of PHP I found this book and decided just to learn every PHP function available so that I could better understand the examples in the above books. Start reading this early on, and complete the whole thing. You'll quickly learn patterns in how PHP functions are named, and as a result be able to guess what a function does within the context of the examples in the above books--even if you don't remember precisely what it does.

Agile Web Application Development with Yii 1.1 and PHP5

http://www.amazon.com/Agile-Web-Application-Development-PHP5/dp/1847199585

I read this book in combination with reading the Blog Tutorial and Definitive Guide on YiiFramework.com. When you're done studying all these materials, you'll be amazed with how much power you have. This book isn't hard to read either. You'll love it if you reach this stage!

Javascript & jQuery:

Learning jQuery, Third Edition

http://www.amazon.com/Learning-jQuery-Third-Jonathan-Chaffer/dp/1849516545

jQuery is a framework built on top of the native browser language of Javascript. Usually one would recommend you learn the base language--Javascript--before learning an abstracted framework on top of it--jQuery. However because of the nature of jQuery and how comprehensive it is and because of how quirky Javascript is coming from PHP, I found it best to jump to jQuery and immediately start accomplishing the DOM manipulation tasks I needed. And ultimately because of syntax similarities between PHP and Javascript I was able to get productive in Javascript without studying a single book just on Javascript. One thing I did different

when studying this book from the PHP books is I did every single tutorial as I read it. The reason is because when I learned PHP, I was learning my first real programming language--so it took me a lot of time to just digest things before I could code a single line, which is why I just read PHP book after PHP book before I got started until it all made sense. However, by the time I got to Javascript & jQuery, I understood how programming in general works and found it helpful for memorization purposes to immediately start doing the tutorials.

jQuery 1.3 with PHP

http://www.amazon.com/jQuery-1-3-PHP-Kae-Verens/dp/1847196985

With this book I didn't do all the tutorials like I did with *Learning jQuery*, but what reading this book did for me is taught me precisely how Ajax works and what it's all about. After reading it, coding features that required Ajax using Yii and PHP was obvious and a no-brainer.

JavaScript: The Good Parts

http://www.amazon.com/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742

This book gave me a deep understanding of the Javascript language and what it's truly all about. After reading it, many hours of debugging and head-scratching when coding Javascript & jQuery were removed from my schedule--because I finally learned the quirks of the Javascript language I needed to know.

Pro JavaScript Design Patterns

http://www.amazon.com/JavaScript-Design-Patterns-Recipes-Problem-Solution/dp/159059908X

Now this book took my Javascript game to the next level, gave me an idea of how jQuery was built, taught me how to do things similar to how you would in a "classical" OOP language like PHP, and completely ended any remaining head-scratching I was having with Javascript, particularly with how "scope" works in Javascript.

Linux:

The Official Ubuntu Server Book, 2nd Edition

http://www.amazon.com/Official-Ubuntu-Server-Book-2nd/dp/0137081332

Note: by the time I read this book I had already learned Linux through blogs on the internet. The best thing I can recommend you do is install Linux on your computer from the Ubuntu website, and start navigating around the command line, practicing Linux commands you learn off the web. Just google "linux tutorials" and you'll be off to a running start. That said, by the time I got proficient in Linux and after I read this book, I felt confident that I really knew what I was doing and had practical solutions for the most common problems you'll face at the command line.

Apache Cookbook: Solutions and Examples for Apache Administrators

http://www.amazon.com/Apache-Cookbook-Solutions-Examples-Administrators/dp/0596529945

This book I treat like a pocket reference and still refer to it often since it's impossible to remember all the different Apache configurations, given how comprehensive this web server application is. I did read it through when I first got it. I kinda skimmed it though--just to get an idea of what is possible. Getting an idea of what is possible without mastering a subject matter is so important in programming because you'll know where to look when you face a challenge that the subject matter can solve.

Pro Bash Programming

http://www.amazon.com/Bash-Programming-Experts-Voice-Linux/dp/1430219971

This is a little advanced for readers of the *FaceySpacey Bible*, but I'm putting it here because it really took my Linux skills to the next level.

NON-TECHNICAL BOOKS:

Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent

http://www.amazon.com/Smart-Gets-Things-Done-Technical/dp/1590598385

If you plan to grow a small application into a large company, this book is a must. It's short. Read it.

SEO Book.com

http://www.seobook.com

This obviously isn't a book, but I read their entire site like a book, and its creator, Aaron Wall, expects you to read it like a book. When I was done reading it, I felt I was completely up to speed regarding what SEO is, how search engines work, and practical techniques to get better rankings in search engines.

For daily Startup Wisdom, checkout FaceySpacey.com/blog daily. And don't forget to download the entire No Bullshit FaceySpacey Bible or more individual chapters here: http://www.faceyspacey.com/resources?section=book .

*Thanks again from FaceySpacey and be sure to check out **FaceySpacey.com** often for further knowledge we kick to take your Startup to the Stars!*