**MBTA P&L Internal Tools –Web Stack Onboarding Document**

We are a **MEAN (MongoDB, Express.js, AngularJS, Nodejs)** shop.

While we use **node.js/express.js** as are web service and load/routing we do almost nothing but install and run it. The only thing to remember is at time you may need to shut-down and restart the webserver for recent changes to take effect.

**MongoDB**-Mongoose (a collection of objects with key-value pairs there in, intentional Not SQL)

We use Mongoose to validate data. We on occasion will go in and direct edit or update or remove but it’s not your tradition SQL database and we spend next to no time think about “proper” way to structure things. At this stage in our development cycle we just want a way to persist data in some semblance of order so we use MONGODB—it’s flexible and easy. (Ref: <https://docs.mongodb.com/manual/tutorial/>)

On the server--

> mongo

>use collection-name

(db is th alias for the current collection you are on)

> db.user.remove({username: "USername"})

Remove method on collection-name.user field

**\*AngularJS—the heart of what you need to learn to be a front end-developer with us**

1. *Go through Prof. Jose Annunciato videos below*
   1. [*https://www.youtube.com/playlist?list=PL\_GGiAMracOWHBKiyJ0hUUG\_yMfBaWSxg*](https://www.youtube.com/playlist?list=PL_GGiAMracOWHBKiyJ0hUUG_yMfBaWSxg)
   2. *Specifically start with 3.16****, 3.17\*,*** *3.18; 4.2, 4.4, 4.6, 4.10*
2. *Read and Complete* ***Steps 0-7 & 9*** *on the official AngularJS tutorial* ***(CRITICAL STEP)***
   1. *\*Link is here:* [***https://docs.angularjs.org/tutorial/***](https://docs.angularjs.org/tutorial/)
   2. SKIP ALL of the TESTING section with each step—our project is not big enough to justify test-driven development—our testing is straight-up end-user testing and our own debugging, we do no formal testing
3. Skim “AngularJS: Up and Running-Enhanced Productivity with Structured Web Apps” skip all TESTing related parts as well (Ask me what parts to focus on).

**Github—Our Version Control System:**

[**https://github.com/MBTA-Procurement-Analysts**](https://github.com/MBTA-Procurement-Analysts) **(DO NOT push anything that can’t be public here, luckily we are a quasi-public agency and almost all of our work is public information)**

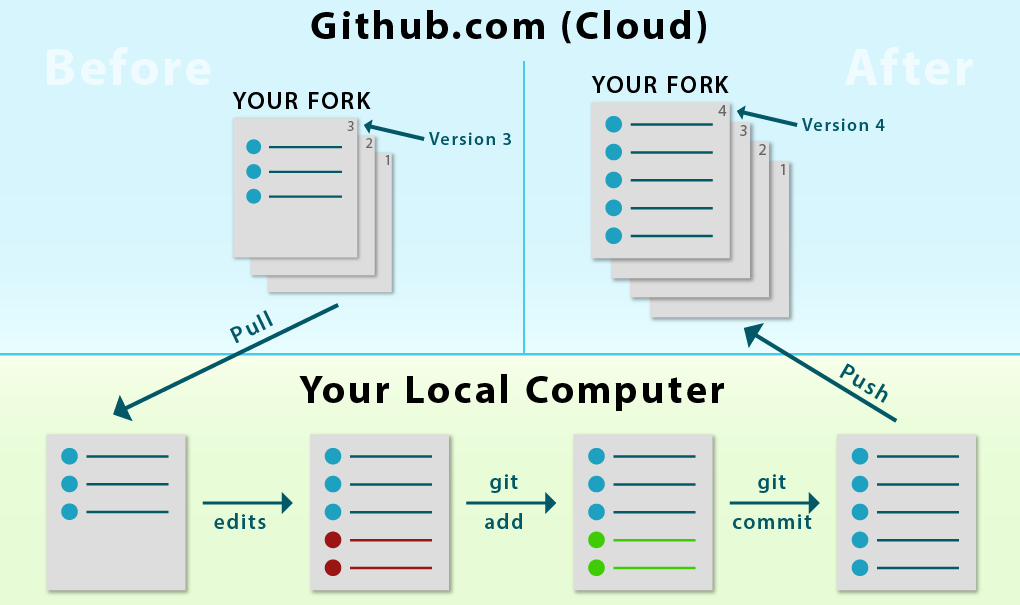
Know the git basic - https://guides.github.com/activities/hello-world/- Basically here is the codebase and were we stay synced, it’s not truly, truly an issue for such a small team but using git has been a time-saver in that we don’t ever think about how to coordinate our coding path. No more “What version are we working on, is this the file you want me to use?” Also, it’s comforting for me to know if shit hits the fan I can figure out how to revert back to some prior working version.

I will go more in-depth about the major concepts of git we use. Most of the actions such as clone or push equate to some command in git.

First you must create an account on github. From github go to the mbta’s procurement repository <https://github.com/MBTA-Procurement-Analysts>. If you’re working on the silver back (local computer that is already set up for web-dev) then you can skip to the diagram. But if you are setting up your own environment you must first fork the repository. This is the same as forking food from someone else’s plate except it is a project we’re taking. In the git terminal type:

> git clone <url address of your fork> (no not the one in the address bar)> (don’t type “<>” its just to show its something that must be replaced with your input. This copies the files onto your local system in whatever file you were when you ran the command. From this same location use

This copies the project over with its commit history. You don’t have to git when you clone projects in this manner. You only initialize git in a directory with git init when the project is starting from your local pc.



Just like you can move through this diagram you can go backwards. Git has 3 areas: Your working directory, your staging area, and your commits. Your working directory is the files themselves. When you change them git notices. When you want to keep your changes you git add them to your staging area. The staging area is a collection of changes that you want to keep. Once you are ready to keep these changes you commit. You can view a log of your commits. You can check the status of your staging area and the changes that have yet to make it to the staging area. This is all useful because if you make changes that break something so bad you don’t know how to fix it or what to fix then you can simply go back to the last commit. You can undo changes at all 3 levels. Now you don’t have to worry about what you change or where you change it as long as you are using git correctly.

When you create a change or feature you typically want to create a branch. A branch is an exact copy of the current repository and its history. You can then switch to this branch and test/break stuff. If all the changes are working in the branch you can merge it to the master branch or you can push it to your forked repository. Once you push to your forked repository you can make a pull request on github.com to merge your changed branch with the main repository that you forked. You can check the address of the location to where you push to. Pulling can also be done and it’s the same as synchronizing your local files with your remote repository. Beware of if you’re syncing from the main project or your fork this can be changed by changing the remote location. This by no means supposed to make you a git expert. This is just to give you an overall idea of what the commands you’re using should be doing.

**JavaScript:**

**Knowing a little bit of JavaScipt is very helpful,** b/c AngularJS is a framework written in JavaScript, hence trailing “JS” in name. Part2 of the book “Maintainable JavaScript” book, skim through ”JavaScript the good parts” I have books on my desk if you need them, you may have academic access.