Ad Tech Ops Tools

A short document that highlights some of the new features of the updated Ad Tech Ops Tools platform, including information on how the site may be maintained and supported as usage or needs change over time.

August 2019 | Zain Shafique, Intern in Ad Technology

Introduction

One of the key changes to the new Ad Tech Ops Tools platform, beyond the new user interface and functionality, is the file structure.

Each element on each individual page is referencing some other HTML file, which allows items to be componentized. The benefit of this new file structure is that making updates to certain items on a page are much easier, since there are now individual files being referenced instead of an HTML file that contains over 500 lines of various tags and JavaScript code.
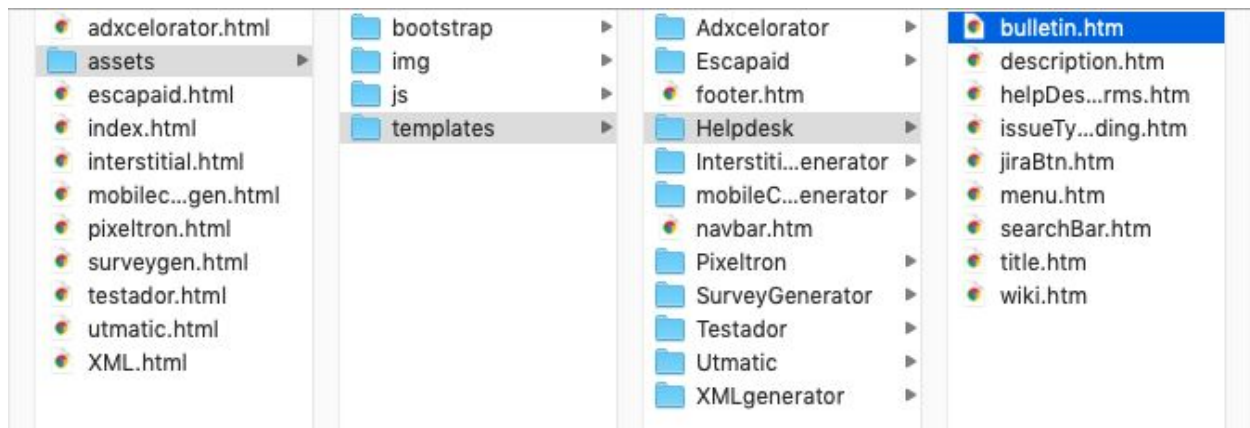
The above screenshot highlights this new component architecture for the file structure of the new platform. Under the "templates" directory there are folders representing each page, and within each directory we have all of that page's component files.

Using this method to represent items on a page is not only helpful for updates, but we can also duplicate components much more easily. If someone wanted to create another button, it would be as simple as duplicating an existing button file, editing the inner HTML to reflect the new values, and inserting its reference into the parent file (one of the .html files from the first tier on the far left).

```
<div class="container">
  <div ng-include="'assets/templates/Helpdesk/bulletin.htm'"></div>
  <div ng-include="'assets/templates/Helpdesk/searchBar.htm'"></div>
</div>
```

This code snippet shows how the files are represented in the HTML file. We have a container encapsulating some components called 'bulletin.htm' and 'searchBar.htm', which happen to be our bulletin board item and search bar on the main help desk page. If you were to copy the line that says

<div ng-include="'assets/templates/Helpdesk/bulletin.htm'"></div>

you would now have two instances of this bulletin board item, and the display would reflect it. This is the idea of the componentized structure, you can make as many components as you'd like, and reference them across one or many other pages.

In addition to each page's proprietary elements being componentized, there are also files for the navigation bar and page footer so that those items can be updated once and be reflected across all pages.

Bulletin Board

What is the bulletin board?

One of the new items on the Help Desk page is the implementation of a "bulletin board." The bulletin board is a component that can be displayed or hidden with certain information. For instance, if there is a known issue with a certain advertisement or tool, we could display this message across the top of the help desk page.

The impact of this feature is that we would be able to effectively reduce the number of redundant tickets for the same issue. This is especially relevant in cases where an end user assumes that the Ad Tech team is unaware of the issue, and adds to the queue with a ticket that could have been avoided.

Usage

In a nutshell, utilizing the bulletin board is a four step process:

1. Access the bulletin board
2. Add the message you'd like to display

3. Display the board
4. Push those changes to the server hosting the site

In order to access the bulletin board, the user must have access to the repository hosting the site's source code. Once the user has accessed that repository, they can enter the command

cd assets/templates/Helpdesk

This command now takes you to the directory containing every component for the Ad Tech Ops Help Desk page. Within this directory, you want to search for the bulletin file. As long as the name hasn't changed since this doc was written, it should be a file titled 'bulletin.htm'. Open this file with your favorite text editor, and you should see something similar to the below image.

```
<!-- BULLETIN BOARD COMPONENT -->
<form style="margin-bottom: 70px; display: none;">
    <p>Bulletin Board of Known Issues:</p>
    <textarea class="form-control" readonly="" style="height: 125px;">
    </textarea>
</form>
```

The two main aspects of this file you want to pay attention to are the parts that say, *"display:none;"* and the area between the *<textarea></textarea>*.

In order to display this bulletin board on the page itself, you need to change the area that says "display: none;" to

display: block;

Once this is set, you will need to add your message in between the textarea tags.

> **<textarea class="form-control" readonly="" style="height: 125px;">**
>   - **We know that this thing is not working as of February 30th, 3019 11:30 AM**
>   - **We also know that this thing isn't working as of February 30th, 3019 12:30 PM**
> **</textarea>**

The resulting file should resemble something similar to the image below.

```
<!-- BULLETIN BOARD COMPONENT -->
<form style="margin-bottom: 70px; display: block;">
    <p>Bulletin Board of Known Issues:</p>
    <textarea class="form-control" readonly="" style="height: 125px;">
      - We know that this thing is not working as of February 30th, 2019 11:30 AM
      - We also know that this thing isn't working as of February 30th, 2019 12:30 PM
    </textarea>
</form>
```

After completing these steps, you can save your file and push these changes to the repository that contains all of the source code. The webpage should now display the bulletin board message, and look similar to the screenshot below.
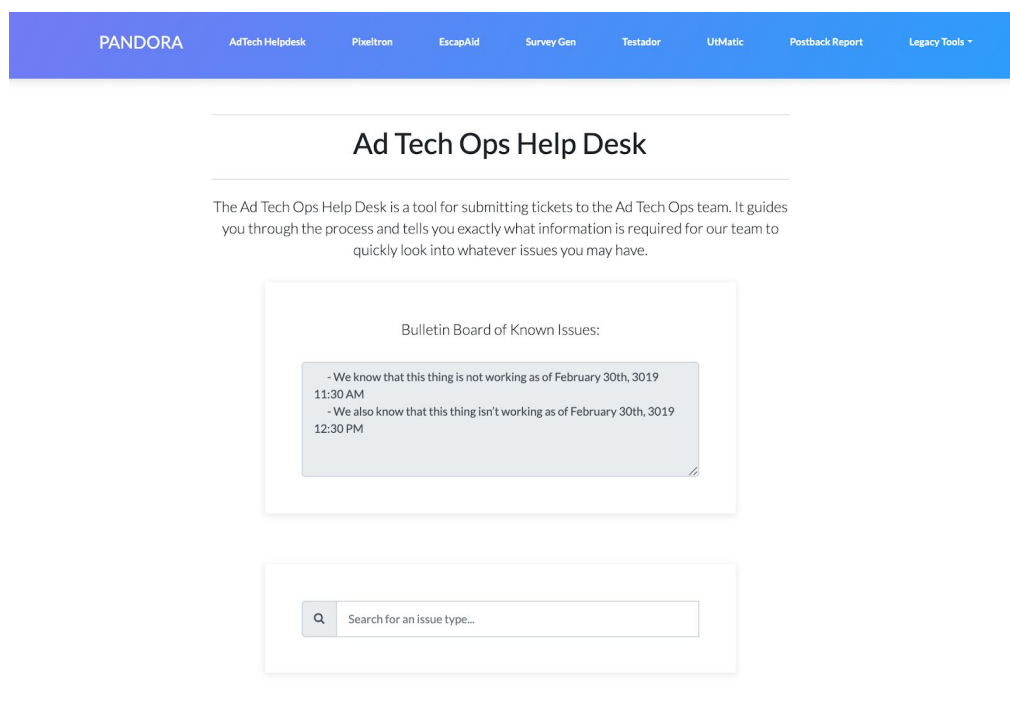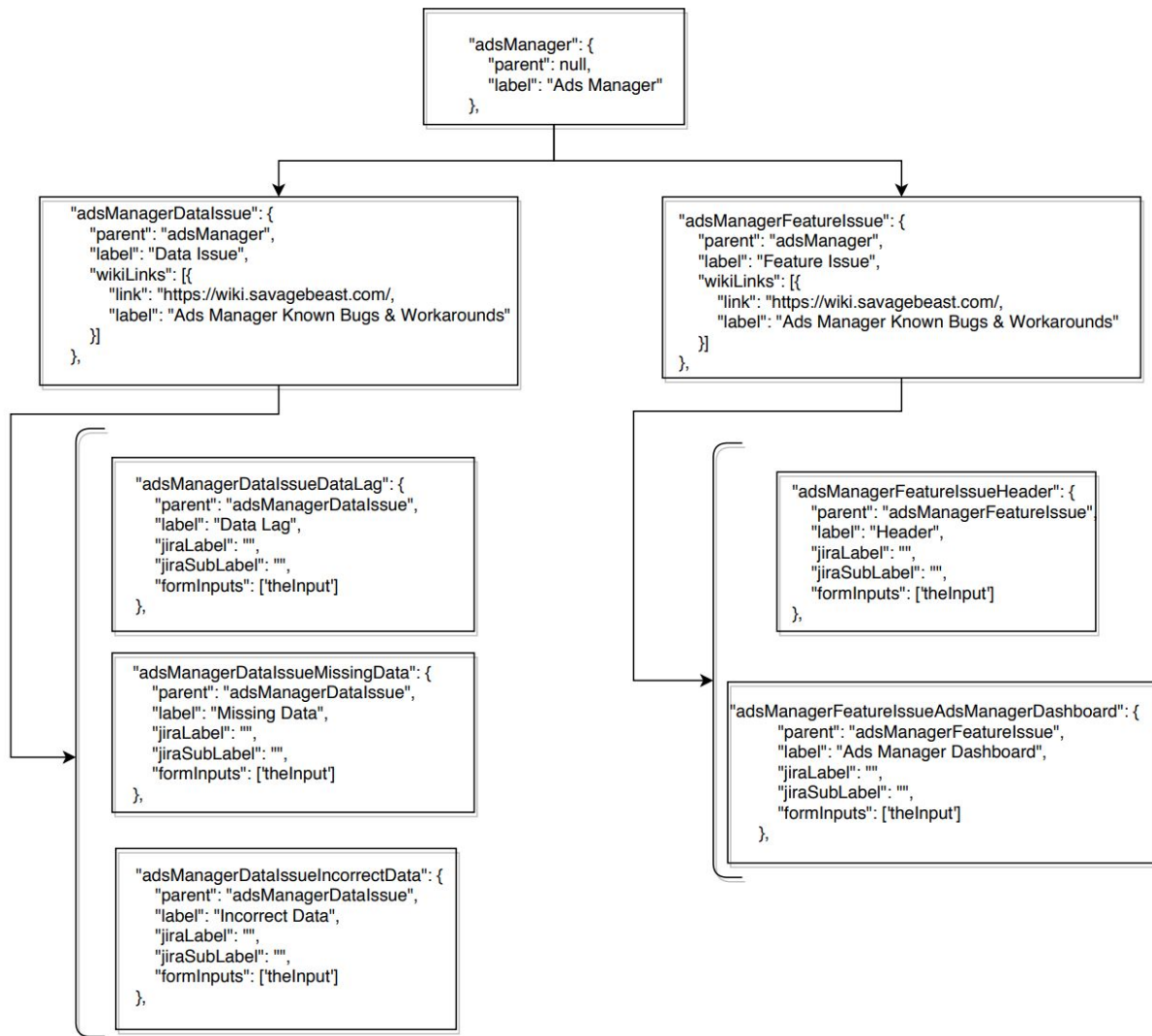
You can follow these same steps in reverse to remove the bulletin board after an issue has been resolved. <u>The key to remember is that the display value will need to be set back to "none" in order for the bulletin board component to disappear from the page.</u>

After setting it back to none, the process is as simple as saving your file and pushing these changes back to the hosting server.

Adding Labels

The next item to cover is adding labels to the existing help desk platform. Labels are the values you select through the dropdown when you are determining what your issue type is. The structure of the labels follow a tree relation. This relation works with three tiers, and you can think of it as a grandparent, parent, and child relationship.

The diagram below gives an example of how the layout looks in the case where we have a team type called "Ads Manager" with issue types titled "Data Issue" and "Feature Issue" each with their own sub issues in the bottommost tier.

```
"adsManager": {
    "parent": null,
    "label": "Ads Manager"
},
```

```
"adsManagerDataIssue": {
    "parent": "adsManager",
    "label": "Data Issue",
    "wikiLinks": [{
        "link": "https://wiki.savagebeast.com/,
        "label": "Ads Manager Known Bugs & Workarounds"
    }]
},
```

```
"adsManagerFeatureIssue": {
    "parent": "adsManager",
    "label": "Feature Issue",
    "wikiLinks": [{
        "link": "https://wiki.savagebeast.com/,
        "label": "Ads Manager Known Bugs & Workarounds"
    }]
},
```

```
"adsManagerDataIssueDataLag": {
    "parent": "adsManagerDataIssue",
    "label": "Data Lag",
    "jiraLabel": "",
    "jiraSubLabel": "",
    "formInputs": ['theInput']
},
```

```
"adsManagerFeatureIssueHeader": {
    "parent": "adsManagerFeatureIssue"
    "label": "Header",
    "jiraLabel": "",
    "jiraSubLabel": "",
    "formInputs": ['theInput']
},
```

```
"adsManagerDataIssueMissingData": {
    "parent": "adsManagerDataIssue",
    "label": "Missing Data",
    "jiraLabel": "",
    "jiraSubLabel": "",
    "formInputs": ['theInput']
},
```

```
"adsManagerFeatureIssueAdsManagerDashboard": {
    "parent": "adsManagerFeatureIssue",
    "label": "Ads Manager Dashboard",
    "jiraLabel": "",
    "jiraSubLabel": "",
    "formInputs": ['theInput']
},
```

```
"adsManagerDataIssueIncorrectData": {
    "parent": "adsManagerDataIssue",
    "label": "Incorrect Data",
    "jiraLabel": "",
    "jiraSubLabel": "",
    "formInputs": ['theInput']
},
```

IMPORTANT note about adding labels & search

Because our labels follow this relational model, it is very important to make sure you do not add a label or issue type with a non existent parent. But what does that mean?

Consider our bottommost tier on the right where we have two labels titled "Header" and "Ads Manager Dashboard." Both of those labels' parent value is "adsManagerFeatureIssue" which is fine, because it exists.

If we were to change one of these parent values to something that did not exist, such as "adsManagerThisDoesntExist," we've now broken our relational model, and our search bar's expected input will no longer work.

The takeaway from this note is to always make sure you are referencing an existing parent value when adding a label. It's fine to add a new parent label, and then reference it afterwards, but if you simply reference a parent without adding it in the bottom two tiers, the code will break since it's not receiving its expected output.

So how do I add labels?

Before you can go ahead and add a label to the help desk tool, you must first make sure that the label has been added to JIRA first, and you have a label id you can add to "jiraLabel" field within the sub issue type. Once you have this, you can move onto the next part.

Adding labels requires that the user has access to the repository containing the source code, and can edit and push changes to the server that is hosting the site. Once these requisites are met, you can enter the command

cd assets/js/Controllers/helpdesk

which will then take you to the directory containing the labels data file. You can enter the "ls" command, which will list all files in the directory. The file you want to open is called "data.js"
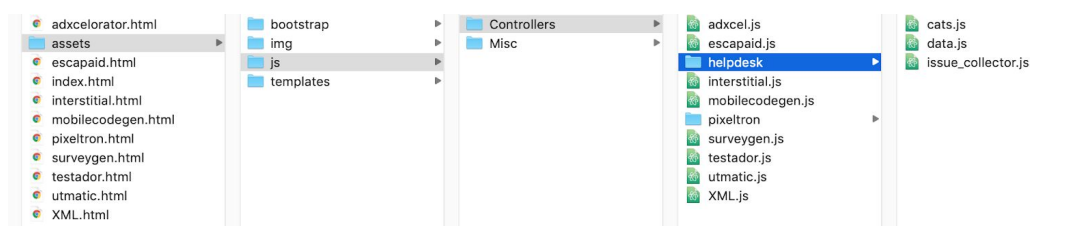
Once you've opened this file, you'll notice that its very long, so zooming out will be helpful in figuring out where in the file you need to append a new value to.

There will generally be three scenarios for adding new labels:

1. You need to add a new sub issue (bottommost tier)
2. You need to add a new issue type & sub issue(s) types
3. You need to add a new team type, issue type(s), & sub issue(s) types

In the event you need to add a new sub-issue, you can cut and paste an existing sub issue block, and edit it's values to reflect whatever parent issue type it needs to sync with. You'll also want to add the JIRA label id, and whatever forms you want to display when selecting it.

In the event you need to add both an issue type and a sub issue type, you want to start with the issue type. You can cut and paste an existing block, make a reference to its team type label, and update its appropriate Wiki link. After this, you can add its respective sub issue types and their JIRA label ids, as well as the forms you would like to display upon selection.

In the event you need to add a new team type with both issue types and sub issue types to reflect, you will need to start with the team type. You can cut and paste an existing team name and update its values to reflect the new team type. The parent for the team type must be "null", since it is the topmost in the relation. After this, you can repeat the steps as if you were adding a new issue type and sub issue type, making sure to reference the new parent team type you just created.

Once these changes have been added, the process is as simple as saving your data.js file and pushing these changes to the server hosting the site. It is a good idea to check these new labels and make sure they're populating the forms you expect them to, as well as making sure they were entered properly and are not breaking any existing functionality.

Accessing Source Code

The process for making updates requires that you have access to the source code for the help desk platform. The source code is stored on bitbucket, and can be accessed in a few steps if you have the rights to update the repository.
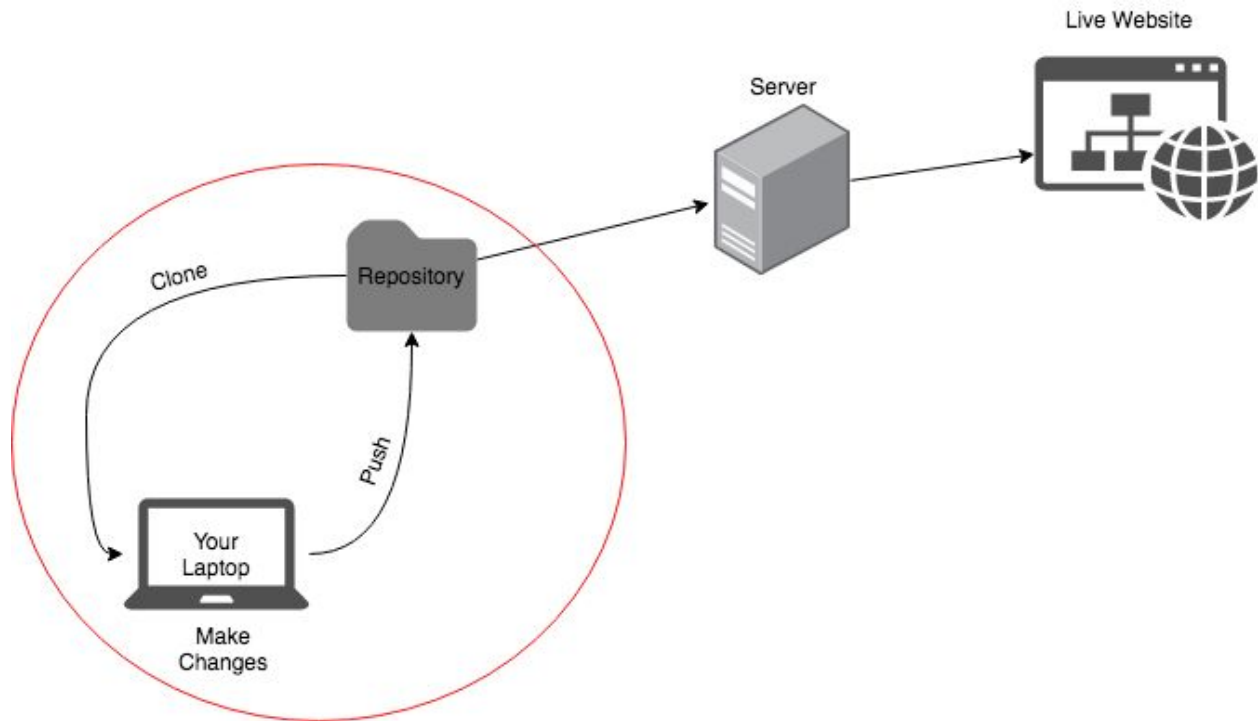
The image above gives a high level overview of how the platform exists from code to the live site. The code in stored in a repository (a fancy name for a folder) where it has some configurations set up to deploy it onto an internal server at Pandora. The server then hosts our website, and displays all of the HTML that we see onto the webpage.

Think of the server as a box with an address, and everytime we go to the URL, we're just going to this address where the contents in the box are being displayed.

In the scope of making updates, all we need to focus on is what is inside of the red circle. The red circle represents how a user would go about making changes. Simplified, it is a three step process.

1. Clone the source repository from Bitbucket
2. Make your desired changes and save the files
3. Push your new updated files back to overwrite what already exists in the repository

In order to do this, you'll want to first ensure you have all of the proper rights to access and make changes to the Bitbucket repository. Next, you'll want to make sure you have Git installed on your computer.

Git is a version control tool that we need in order to clone and push changes to our source repository. Git is very useful because it allows you to keep old versions of code

through things called "commits." Commits act as a reference to a point in time where you made a change. Let's say we made some changes to our source code, and now the website isn't working at all and we can't figure out why. Git will allow us to reset our code to a previous point in time where the code was working.

To begin the process, you'll want to fire up a new terminal and enter the command:

git clone https://bitbucket.savagebeast.com/scm/adtech/adtech-ops-help-desk-2.0.git

This command will clone the existing repository directly from bitbucket. When you enter this command, your terminal should look something like this:

```
bash-3.2$ git clone https://bitbucket.savagebeast.com/scm/adtech/adtech-ops-help-desk-2.0.git
Cloning into 'adtech-ops-help-desk-2.0'...
remote: Enumerating objects: 168, done.
remote: Counting objects: 100% (168/168), done.
remote: Compressing objects: 100% (112/112), done.
remote: Total 168 (delta 54), reused 168 (delta 54)
Receiving objects: 100% (168/168), 1.23 MiB | 3.93 MiB/s, done.
Resolving deltas: 100% (54/54), done.
bash-3.2$
```

Now that we've cloned the repository, we can view all of the files in it. You can do this through the terminal using the "ls" command to list all files and folders, or you can navigate to this folder using your **finder** app on Mac.

In the event you can't find where the file is located, you can go back to your terminal and enter the command "pwd", which will list the file-path that contains this folder.

```
[bash-3.2$ pwd
/Users/zshafique/desktop
bash-3.2$
```

This tells us that our folder is located on our desktop. At this point, you can go to your desktop through your finder, and locate the files you'd like to make changes to.

After you've made changes to the file you wanted to update, be sure to save that file and then come back to your terminal in order to push the changes back to the repository.

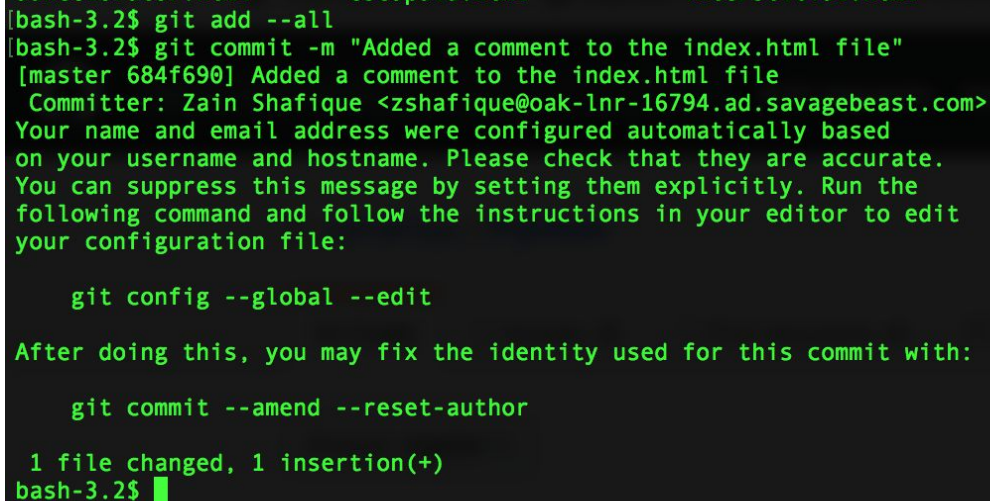Once back in the terminal, you will want to add all of your changes by entering the command:

git add --all

After you've done this, you now want to "commit" these changes. Committing is essentially adding a point on the timeline of your code. This is when you'll also add a message about the changes you made, so that other users know what you updated in the future. The command will need to follow the format of:

git commit -m "Added a comment to the index.html file"

NOTE: <u>You should put a message relevant to the changes you made in between the quotation marks, it doesn't have to be what I've included in the example.</u>

Your terminal should look similar to the image below:

NOTE: <u>If this is your first time making an update, or Bitbucket doesn't recognize you as a user, it may ask you for some configurations for the first time. You can enter the following commands below to configure (_with your name and email of course_):</u>

Configure Git for the first time

```
git config --global user.name "Zain Shafique"
git config --global user.email "zshafique@pandora.com"
```

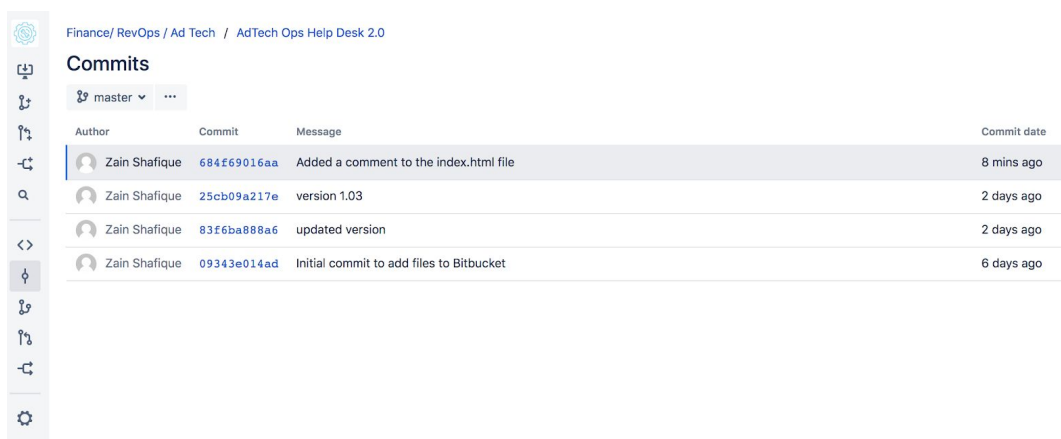Once you've successfully committed, you're now ready to push your changes! Enter the command:

git push -u origin

```
[bash-3.2$ git push -u origin
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 420 bytes | 420.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
To https://bitbucket.savagebeast.com/scm/adtech/adtech-ops-help-desk-2.0.git
   25cb09a..684f690  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
bash-3.2$
```

At this point, all of your updates should have been pushed back to the repository that contains all of the source code. Within a few minutes, the server will build the newly updated site and you'll be able to see the changes you made. You can check that your updates went through by going to:

https://bitbucket.savagebeast.com/projects/ADTECH/repos/adtech-ops-help-desk-2.0/commits

This page will display all of the "commits" for this repository. You should be able to see a timestamp on your last commit confirming that it went through, similar to the image below: