Various Projects in the ASSISTments Foundation

A Major Qualifying Project
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

by
Adam Goldsmith

Date:
August 9, 2019

Submitted to:

Professor Neil Heffernan
Worcester Polytechnic Institute

Abstract

This Major Qualifying Project (MQP) consisted of several sub-projects, all relating to the ASSISTments Foundation and associated projects. The first sub-project created a way to inject JavaScript code into ASSISTments' web tutor application via the PeerAssist system for proving hints using stored Cross Site Scripting techniques to allow performing a study. The second sub-project consisted of improvements to the mobile app written in C term of 2019 during my IQP, including an unsuccessful attempt to add support for the new version of ASSISTments. The third sub-project was the successful migration of the entire codebase between version control systems, from Subversion to Git, and to new hosting on Github.

Acknowledgments

I would like to thank Professor Neil Heffernan for offering the opportunity to continue working on projects in the ASSISTments Foundation. Thank you to Anthony Botelho for his continued assistance with the mobile app, Cristina Heffernan for her UI advice on the mobile app, and March Patikorn for his mentoring on PeerAssist and the ASSISTments development environment. Thank you to Chris Donnelly and David Magid for their development knowledge and willingness to work through the Git migration with me.

I would also like to thank all the contributors to ASSISTments for providing an environment to build upon.

Authorship

The sole authorship of this paper and the associated code belongs to Adam Goldsmith. The development of this project was completed exclusively at Worcester Polytechnic Institute (WPI) for ASSISTments, a free public service of the ASSISTments Foundation created by Neil and Cristina Heffernan.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

ASSISTments is a free homework and tutoring service, as well as a scientific research platform. It provides automatic grading and feedback for a wide variety of existing and user-supplied content. ASSISTments also supports a number of scientific studies to improve student learning. Currently, it is in the process of transitioning to a major new version (2.0), which brings a number of new features. I worked on several projects for this MQP, including injecting code into the 1.0 tutor, updates to the mobile app, and migrating the code base to a new version control system.

2 TeacherAssist Code Injection

2.1 Introduction

PeerAssist and TeacherAssist are components of ASSISTments that provide hints and explanations to students on demand. The hints can be provided by ASSISTments, the teacher, or other students, depending on the teacher's settings. An example of their normal usage in ASSISTments 1.0 can be seen in Figure 1.

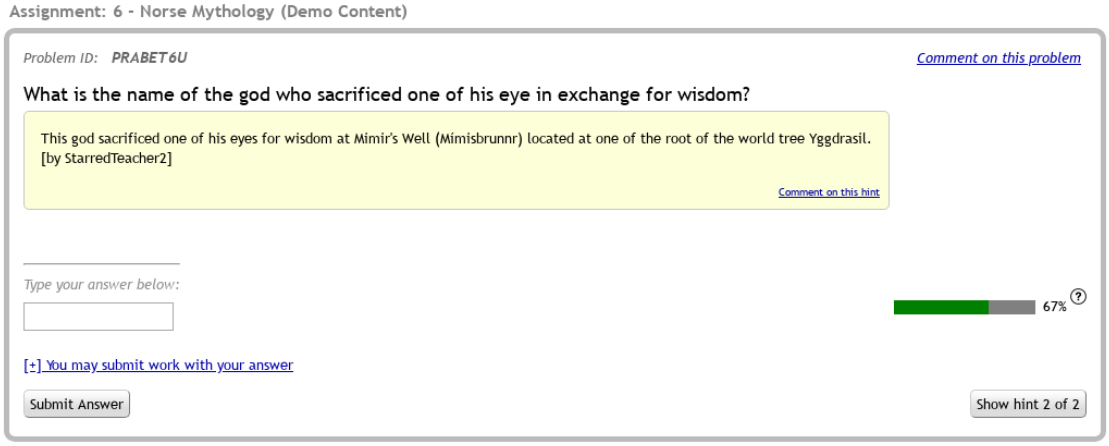


Figure 1: Normal Hint Display

A proposed study would try to gather data about how helpful these hints are, and how the wording of the prompt (ex. "did this help" vs "was this hint helpful") affects these results. To do this, some buttons and a bit of JavaScript had to be added to the tutor. However, PeerAssist and TeacherAssist are currently only implemented in ASSISTments 1.0, and the tutor in 1.0 is in a feature freeze since all development effort is going into 2.0, so no code could be added to it to implement this. However, the `PeerAssistService` project, which provides the hints and explanations to the tutor, is a separate project which is still editable. I was therefore asked to find a way to implement this study in such a way that it only required editing the `PeerAssistService` code.

2.2 Implementation

I started by trying to see if I could get code injected into the client page at all. For testing, I just manually inserted the HTML directly into the problem in the Postgres testing database. The naive implementation would just be to inject a `<script>` tag into each Hint/Explanation and be done with it. However, for unclear reasons this did not work, and so a more creative method had to be implemented. My solution was to use a stored Cross Site Scripting (XSS) technique to run a small amount of JavaScript which could then load a larger JS file, allowing arbitrary code execution.

Cross Site Scripting is an injection attack in which an attacker exploits a vulnerability in a site to cause an end user to download and run code of the attacker's choosing.[1] There are two main types of XSS attack: reflected and stored. A reflected attack relies on the attacker convincing the user to interact with a malicious link, form, etc that has been modified to insert their code. In a stored attack the attacker instead stores the malicious code on the server in such a way that an end user will be sent the code during normal usage of the site. The second is much more dangerous as it can hit every user with no further effort by the attacker, but is typically much harder to achieve.

An XSS is very similar to this problem; in both the author is trying to get code into a page for which they cannot modify the original source. This is clearly closer to a stored XSS, as I need to have it affect all users, not just those who click a link, and I already have access to the ability to store or modify data in `PeerAssistService`. After experimenting with several different types of stored XSS with no success, I found that an `<img>` tag's `onerror` did evaluate, when the image file did not exist, although it did cause a lot of 404 errors due to the frequent requests for the specified file. A basic example is shown in Listing 1, and Figure 2 shows what a user would see if this were injected (minus the alert window), as the browser interprets this as a broken image.

```
1 <img src="fail" onerror="alert('test')">
```

Listing 1: Basic Image `onerror`

Some more experimenting revealed that I could use a custom URL scheme, which will always fail without making a network request as long as no one defines it. I chose `fail:`, as that seemed unlikely to be defined in the future. This is shown in Listing 2 This does still produce warnings in the JavaScript console, but it was the best solution I could find.

```
1 <img src="fail:" onerror="alert('test')">
```

Listing 2: URL Scheme `onerror`

Now that I had code injection working, I wrote up a bit of JavaScript to fetch a remote script and insert it into the page source, shown in Listing 3.

However, this caused the page to retrieve and run many copies of the script. For whatever reason, the script was executed far more times than the number of hints actually displayed.

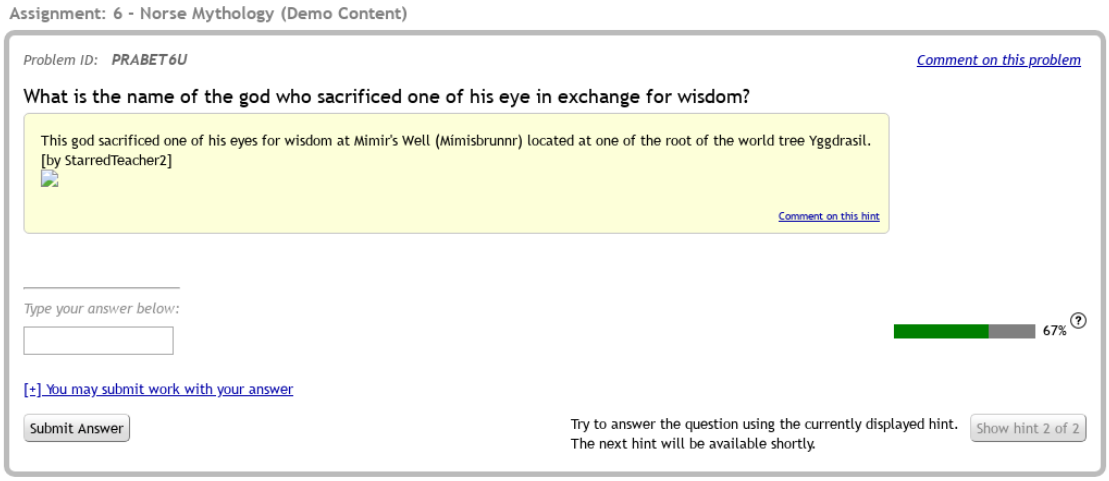


Figure 2: Failed Image Load

```
<img src="fail:"
  onerror="document.body.appendChild(document.createElement('script')).src =
  '/inject.js'" />
```

Listing 3: External Script Injection

So I adjusted the `onerror` script to run only if the script had not already been injected, as shown in Listing 4.

```
<img src="fail:"
    class="teacherassist-inject"
    onerror="window.tainjected ||
     (document.body.appendChild(document.createElement('script')).src =
     '/inject.js')"
    />
<!-- inject.js sets window.tainjected to true when loaded -->
```

Listing 4: Injecting External Script Exactly Once

Now that I had the ability to load an external script, I wrote the code shown in Listing 5. This is mostly straightforward creation of buttons and text and POSTing the choice to a server. The `MutationObserver` at the end of the file allows us to re-run the `tryReplace` function every time the page changes. This function looks for an element with the class `teacherassist-inject`, indicating that it hasn't been handled yet. It then strips that class and the `<img>` tag used to inject the code, and replaces them with text and buttons. The output of this can be seen in Figure 3.

```
window.tainjected = true;

function handleSubmit(event) {} // Omitted, shown below
```

```

function makeButton(text, value) {
  let button = document.createElement("button");
  button.className = "gwt-Button";
  button.textContent = text;
  button.setAttribute("value", value);
  button.addEventListener("click", handleSubmit);
  return button;
}

function tryReplace() {
  // should never have more than one at a time, but just to be sure
  [...document.querySelectorAll(".teacherassist-inject")].map(async el => {
    // remove XSS img tag and CSS class
    el.removeChild(el.firstChild);
    el.className = "";

    let buttonDiv = document.createElement("div");
    buttonDiv.append(makeButton("No", "No"), makeButton("Yes", "Yes"));
    el.append(
      document.createElement("hr"),
      el.getAttribute("data-prompt"),
      buttonDiv
    );
  });
}

// replace the already existing element (ie the one that loaded this)
tryReplace();

new MutationObserver(tryReplace).observe(document, {
  // watch the whole page
  childList: true,
  subtree: true
});
```

Listing 5: Simplified TeacherAssist `inject.js`
(see Appendix A for the full file)

On the server side, a relatively simple patch, shown in Listing 6 was added to the PeerAssistService project to inject the HTML and handle the submission. This allows for setting the prompt text, via `data-prompt`, as well as passing in other data to retrieve later in the client side submission function, shown in Listing 7. This function pulls out all of the `data-` attributes on the element, and passes them back to the `submitUsefulnesReport` handler on the Java side. This allows passing arbitrary identifiers to the client, and then getting them back with the user's selection later.

```
--- a/src/main/java/org/assistments/peerassist/rest/controller/PeerAssistController.java
+++ b/src/main/java/org/assistments/peerassist/rest/controller/PeerAssistController.java
@@ -3,8 +3,9 @@ package org.assistments.peerassist.rest.controller;
 import java.time.Instant;
 import java.util.ArrayList;
```

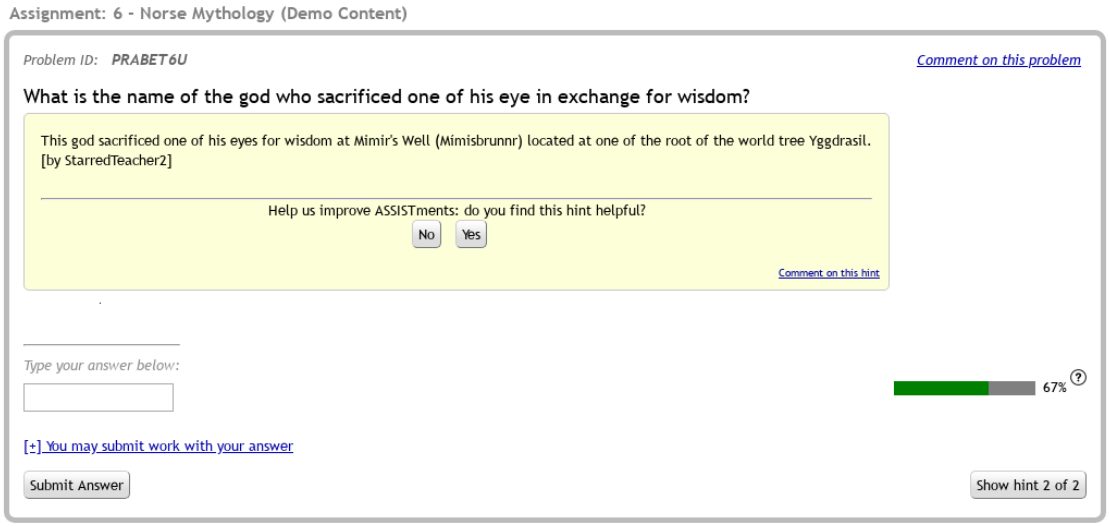


Figure 3: Feedback Form Successfully Injected

```
 import java.util.List;
 import java.util.Map;

+import org.assistments.domain.content.tutor.Hint;
 import org.assistments.domain.content.tutor.ManifestCode;
 import org.assistments.domain.content.tutor.ManifestDTO;
 import org.assistments.domain.content.tutor.Persistable;
@@ -48,6 +49,16 @@ public class PeerAssistController {
    private boolean debugPrint = false;
    private boolean disablePeerAssist = true;
    private boolean trackProcessingTime = true;
+
+  @RequestMapping(method = RequestMethod.POST, value = "/submit_usefulness_report")
+  public ResponseEntity<String> submitUsefulnesReport(@RequestBody Map<String,
 ↪  Object> request) {
+    if (request.containsKey("did_help")) {
+      System.out.println("Got submission:" + request.get("did_help") +
+                  " key: " + request.get("key"));
+      // do something with the value
+    }
+    return new ResponseEntity<String>(HttpStatus.OK);
+  }

    @JsonView(View.Exposed.class)
    @RequestMapping(method = RequestMethod.POST, value = "")
@@ -154,7 +165,21 @@ public class PeerAssistController {
                    .getTutoringManifestsContents(userId, classId, assignmentId,
                     ↪  problemId, classType, isPreview);
            if (!Util.isNullOrEmpty(teacherAssists.getFirst())) {
                manifestList.addAll(teacherAssists.getFirst());
-        contentList.addAll(teacherAssists.getSecond());
+
+        // nasty hack to inject js into the tutor
```

```
+        List<Object> tempContents = teacherAssists.getSecond();
+        for (Object content : tempContents) {
+          if (content instanceof Hint) {
+            Hint hint = (Hint) content;
+            // could change prompt text here
+              hint.setText(
+              hint.getText() +
+              "<div class=\"teacherassist-inject\" data-prompt=\"Did that help?\"
 ↪  data-key=\"" + peerHintKey + "\">" +
+              "<img src=\"fail:\" onerror=\"window.tainjected ||
 ↪  (document.body.appendChild(document.createElement('script')).src =
 ↪  'https://users.wpi.edu/~asgoldsmith/inject.js')\" />" +
+              "</div>");
+          }
+        }
+        contentList.addAll(tempContents);
               error.addAll(teacherAssists.getThird());
               ManifestDTO tutorStrategyDTO = ((ManifestDTO) teacherAssists.getFirst()
                       .get(teacherAssists.getFirst().size() - 1));
```

Listing 6: PeerAssistService Patch

```
function handleSubmit(event) {
  // get all data attributes and add to request
  const prefix = "data-";
  let data_attrs = [...event.target.parentElement.parentElement.attributes]
    .filter(a => a.name.startsWith(prefix))
    .reduce((obj, a) => {
      obj[a.name.substring(prefix.length)] = a.value;
      return obj;
    }, {});

  fetch("/PeerAssistService/get_peer_tutoring/submit_usefulness_report", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "assistments-auth": 'partner="PeerAssistService"'
    },
    body: JSON.stringify({
      did_help: event.target.value,
      ...data_attrs
    })
  });

  let thanks = document.createElement("div");
  thanks.textContent = "Thanks for your feedback!";
  event.target.parentElement.replaceWith(thanks);
}
```

Listing 7: `inject.js` `handleSubmit` function

2.3 Future Work

The actual study that utilizes this code has not been run yet, so there is no data as to how well this actually works. This is also written entirely for the 1.0 version of the tutor, which is what necessitated such a messy hack in the first place. Ideally, this functionality could be written in much more cleanly to the 2.0 tutor, completely bypassing all of the work I did to get it working in 1.0.

Once this is no longer required, it would be good to add some Content Security Policy (CSP) rules to prevent a real XSS attack from being implemented. Unfortunately, I suspect that some of Google Web Toolkit (GWT) may not allow for a strict policy, but even just blocking inline scripts would be an excellent step in the right direction.

3 Mobile App Improvements

3.1 Introduction

In C term of 2018-2019, Ben Emrick and I wrote a cross-platform mobile app for ASSISTments 1.0 using NativeScript-Vue[1]. It essentially just provides a webview wrapper around the ASSISTments web interface with login saving, native handling of image submission for showing work, and better handling of offline mode.

3.2 Style and Misc Improvements

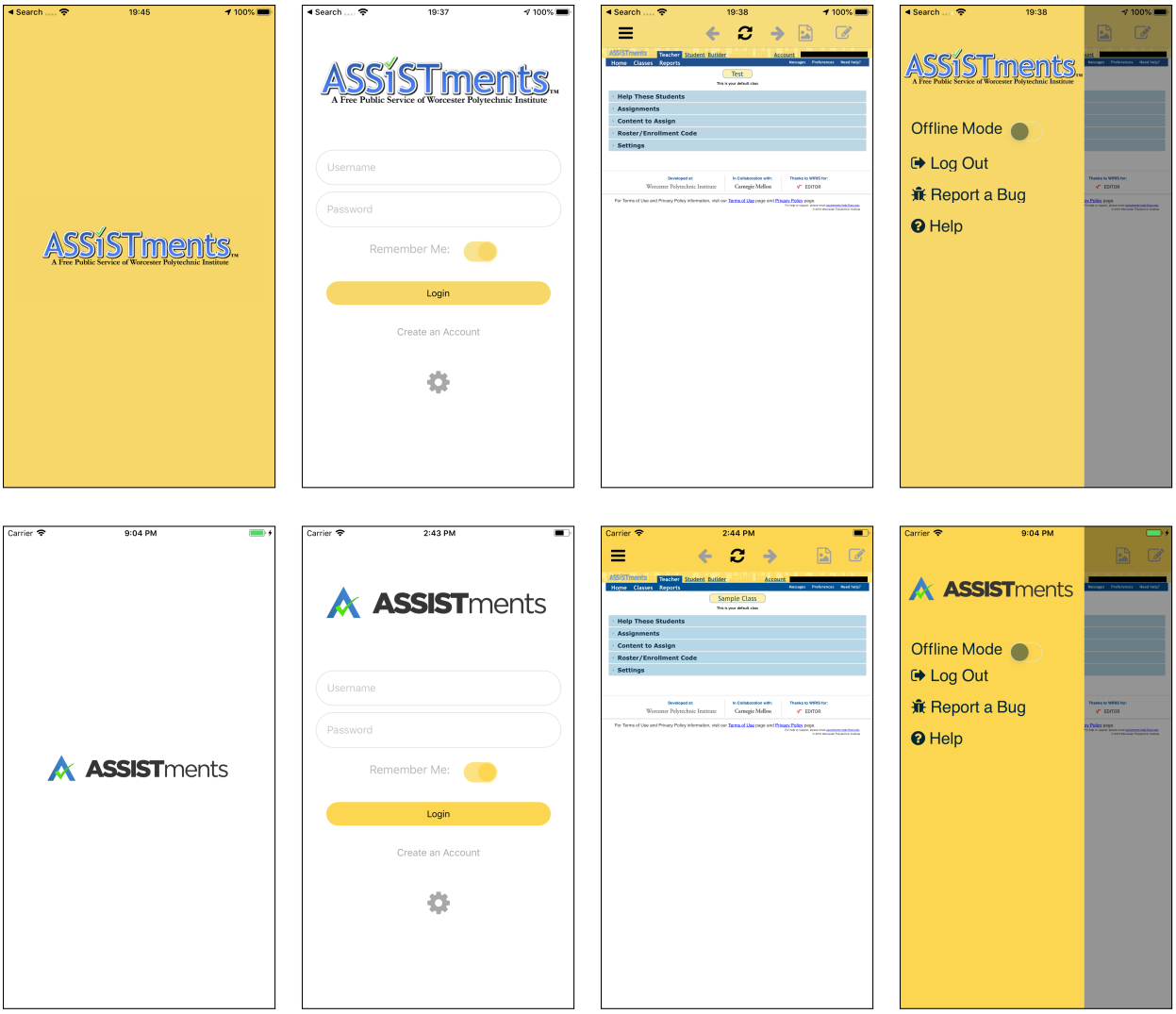
The most visually obvious change was updating the app to use 2.0 branding. The app icon and logo are essentially the only thing that changed. These changes can be seen in Figure 4.

There were a number of library updates since the end of my IQP, including a major NativeScript version (6.0). The webview library that we used, `nativescript-webview-ext`, implemented an option to disable its setting of the viewport in WKWebView at my suggestion, allowing me to drop the hacky patch disabling it locally. That project also added handling of JavaScript dialogs, allowing me to remove a good chunk of code that I had written for that purpose.

The main fix I wrote for this release was fixing the ActionBar element, which was implemented somewhat oddly due to differences in behavior across iOS and Android. The old solution, as shown in Listing 8 was the "correct" way to do it, using an actual `ActionBar` element, but on iOS you cannot place arbitrary buttons on the left, and on android you can only place `ActionItem`s on the right. This solution was to use a `GridLayout` inside the `ActionBar` with nested `StackLayout`s that are aligned to the left and center, and `ActionItem`s for the right buttons. However, this lead to the incorrect placement of buttons seen in Figure 4c since the `GridLayout` overlaps with the `ActionItem`s on Android, but not on iOS.

```
1 <ActionBar class="action-bar">
2   <GridLayout class="action-bar-contents">
3     <StackLayout class="hstack btns-left">
4       <IconButton @tap="$drawer.showDrawer()" text="&#xf0c9;" />
```

[1] https://nativescript-vue.org/



(a) Splash Screen (b) Login Screen (c) Home Screen (d) SideDrawer

Figure 4: Comparison of v1.0.0 (top) vs v1.1.0 (bottom)

```
    </StackLayout>
    <StackLayout class="hstack btns-center">
      <IconButton
        @tap="webview.goBack()"
        :isEnabled="webview && webview.canGoBack"
        text="&#xf060;"
      />
      <IconButton @tap="webview.reload()" text="&#xf021;" />
      <IconButton
        @tap="webview.goForward()"
        :isEnabled="webview && webview.canGoForward"
        text="&#xf061;"
      />
    </StackLayout>
  </GridLayout>
  <ActionItem ios.position="right" @tap="showWork">
    <IconButton text="&#xf1c5;" :isEnabled="canShowWork" />
  </ActionItem>
  <ActionItem ios.position="right" @tap="openScratchPad">
    <IconButton text="&#xf044;" :isEnabled="canShowWork" />
  </ActionItem>
</ActionBar>
```

Listing 8: Old `ActionBar` Implementation
[from `Assistments-Mobile/app/components/Home.vue`]

After experimenting with several other ways to write this structure, I discovered that I could just drop the `ActionBar` component, and use a `GridLayout` directly. This required slightly restructuring the rest of the Home screen, but seems to work quite well on all platforms. The code for this implementation can be seen in Listing 9, and the results at the bottom of Figure 4c. At some point in this set of library upgrades, the font rendering on the sidebar was fixed from being slightly cut off in v1.0.0, as seen in Figure 4d.

```
<StackLayout>
  <GridLayout v-show="actionBarVisible" columns="*" class="action-bar">
    <StackLayout class="hstack btns-left">
      <IconButton @tap="$drawer.showDrawer()" text="&#xf0c9;" />
    </StackLayout>
    <StackLayout class="hstack btns-right">
      <IconButton
        @tap="showWork"
        text="&#xf1c5;"
        :isEnabled="canShowWork"
      />
      <IconButton
        @tap="openScratchPad"
        text="&#xf044;"
        :isEnabled="canShowWork"
      />
    </StackLayout>
    <StackLayout class="hstack btns-center">
      <IconButton
```

```
        @tap="webview.goBack()"
        :isEnabled="webview && webview.canGoBack"
        text="&#xf060;"
      />
      <IconButton @tap="webview.reload()" text="&#xf021;" />
      <IconButton
        @tap="webview.goForward()"
        :isEnabled="webview && webview.canGoForward"
        text="&#xf061;"
      />
    </StackLayout>
  </GridLayout>
  <GridLayout rows="*"> <!-- WebViewWrapper Ommitted --> </GridLayout>
</StackLayout>
```

Listing 9: New `ActionBar` Implementation
[from `Assistments-Mobile/app/components/Home.vue`]

3.3 Implementing a Service Worker for Offline Support in 2.0

In the 1.0 Tutor, an Application Cache is used to store the resources (ex HTML and JavaScript Files) needed for running the tutor offline. However, according to Mozilla's Documentation, Application Cache has been marked as deprecated since 2015 [2]. While it hasn't actually been removed quite yet, and may never be, AppCaches also have some significant downsides, primarily in the fixed nature of the files that they cache and the caching behavior that they provide. The replacement is Service Workers, which are much more powerful and flexible, but somewhat more complicated. They allow running arbitrary JavaScript in a separate worker thread, primarily to intercept network requests and provide cached versions of the files instead.

The implementation in 1.0 was based on a GWT Linker, which are sort of like steps in a bundler, operating on and producing Artifacts that eventually become files added to the final build. GWT builds a large number of different permutations of its compiled JavaScript output; as configured, one per browser and language pair. The 1.0 implementation used a Linker called `SimpleAppCacheLinker` to make a large number of `.nocache.manifest` files, one per permutation, as well as `.nocache.manifest.html` files that just applied those manifests. The manifests only differed by two files, a single `.cache.js` and the `.nocache.manifest.html`, as these are the only files that are permutation specific. At runtime, the tutor created an `iframe` tag with the appropriate `.nocache.manifest.html` as its source, causing the page to load the right manifest.

In the 2.0 implementation, I could use JS to dynamically retrieve the right `.cache.js` file, so I only needed to produce one service worker file instead of several `.nocache.manifest` and `.nocache.manifest.html` files. The implementation of the Linker is rather similar to the `SimpleAppCacheLinker`, although greatly simplified, and with use of Java 8's `Stream`s instead of large numbers of `for` loops. It can be seen in Appendix B. Essentially, it creates a `PermutationArtifact` for each permutation, which stores a list of all the files associated with that permutation, then uses those `PermutationArtifact`s to to create a mapping

from the permutation name to a list of files. It then creates a `service-worker.js` from `service-worker.template.js`, shown in Listing 10, by prefixing it with the lists of files. At runtime, instead of making an iframe with the `.nocache.manifest.html`, it just registers the service worker with a query string to pass the permutation name. The service worker itself is very basic; it just adds all the relevant files to the cache when it starts, and then uses them when it can. There are a lot of more powerful things that could be done with this service worker, but a present it works roughly the same as an Application Cache, with the benefits of being much simpler and not being deprecated.

```
const CACHE_NAME = "v1";

this.addEventListener("install", event => {
  const permutation = new URL(location).searchParams.get("permutation");

  event.waitUntil(
    caches.open(CACHE_NAME).then(cache => {
      return cache.addAll([
        ...STATIC_FILES,
        ...COMMON_ARTIFACTS,
        ...PERMUTATION_ARTIFACTS[permutation]
      ]);
    })
  );
});

this.addEventListener("fetch", event => {
  event.respondWith(
    new Promise(async res => {
      // return a cached response if we have one
      let match = await caches.match(event.request);
      if (match) res(match);
      else {
        // retrieve the response from the network and add it to the cache
        let response = await fetch(event.request);
        let cache = await caches.open(CACHE_NAME);
        cache.put(event.request, response.clone());
        res(response);
      }
    })
  );
});
```

Listing 10: Service Worker JS Template
[from `tutor/client/src/main/resources/service-worker.template.js`]

3.4 Mobile App Login Handling for 2.0

The current mobile app only supports ASSISTments 1.0, which is in the process of being phased out in favor of 2.0, so support for 2.0 was desired for the app. In 1.0, signin is done through a webpage in ASSISTments directly to ASSISTments accounts, whereas 2.0

uses external authentication providers via OAuth. At the time of this writing, only Google Classroom is supported. 2.0 also differs in that presently it has no UI for students to view their assigned work, as that is all handled by Google Classroom.

Google Classroom gives students a link of the form `https://app.assistments.org/xis/xiid/[problem id]`, which then redirects to the OAuth page for Google if necessary, and then redirects to the actual assignment. To handle a link from Google classroom, I used the `nativescript-urlhandler` plugin[2]. Seamless link handling requires a few files placed in the server's `/.well-known/` directory confirm to the OS that the App is actually related to the website. These can be seen in Listing 11 and Listing 12 for iOS and Android, respectively.

```
{
  "applinks": {
    "apps": [],
    "details": [
      {
        "appID": "X9723G3J94.org.assistments.mobile",
        "paths": [ "*"]
      }
    ]
  }
}
```

Listing 11: Debug iOS `apple-app-site-association`

```
[{
  "relation": ["delegate_permission/common.handle_all_urls"],
  "target": {
    "namespace": "android_app",
    "package_name": "org.assistments.mobile.client",
    "sha256_cert_fingerprints":
      ↪ ["7A:31:B1:7C:9C:A1:E3:CE:C0:44:29:BF:68:72:52:19:71:9A:D7:5E:45:87:EE:3E:FE:DB:DB:84:5A:85:E6
      ↪ "39:66:7D:2C:16:9D:CA:40:AE:5C:99:99:DA:91:57:A3:0D:5A:6A:64:57:4E:18:C0:7E:62:E5:66:7F:85:E5
  }
}]
```

Listing 12: Debug Android `assetlinks.json`

The combination of these and the client side code is, however, extremely difficult to debug, as there is minimal feedback as to whether or not the remote side is being detected correctly by the OS. However, I did eventually get the redirection working, and could proceed to attempt get authentication working.

[2] https://github.com/hypery2k/nativescript-urlhandler

Google Classroom uses normal Google Accounts for authentication. In theory, this means that I could use the native Google Sign-In libraries provided for both iOS and Android. Unfortunately, there is no NativeScript library that supports both platforms and allows for authenticating against a remote website, so I had to work with native code to implement this feature, which slowed things down significantly.

The Android app, or more specifically it's store page, is owned by `the.assistments.teacher@gmail.com`. There is also a Firebase project containing both the iOS and Android Apps, which gives us Crashlytics and some metrics, which is also owned `the.assistments.teacher@gmail.com`. Google's docs are somewhat inconsistent, but Firebase seems to be the recommended way of managing OAuth credentials; as a Google product, is integrated into the rest of the ecosystem and automatically creates OAuth2.0 credentials in the Google API Console for the managed project. Unfortunately, The ASSISTments CAS has it's OAuth2.0 Client ID in a project from `ASSISTmentsOwner@gmail.com`. There is a method called `requestServerAuthCode` in the Google Sign In API which is supposed to be used in exactly this scenario, with a app client authenticating for a remote server. As far as I can tell, although I can find no source explicitly confirming this, to do a `requestServerAuthCode`, the server's client ID has to be in the same project as the client's client ID. I believe that without those being in the same project, Google's Sign-In returns `DEVELOPER_ERROR`.

This is as far as I got unfortunately, as I was unable to get this working before the end of the term. I am uncertain how much work would be required to finish this authentication procedure; it could just be fixing a minor bug in the call to the Google Sign In API, or changing to a shared project, or I could be fundamentally misunderstanding the correct OAuth signin procedure.

3.5 Future work

Once the authentication problems are worked out, there are some UI issues with how the LoginScreen would work with 2.0. Additionally, some UI for Offline support in 2.0 would have to be written, specifically a button to download assignments. However, I am growing less convinced that updating the app for 2.0 is entirely worthwhile. In terms of features, most would not yet work in 2.0, and the offline support could just as easily be done with browser support. The amount of odd behavior and inconsistency between platforms means that what should be a very simple app to write and maintain quickly becomes much more difficult. Just maintaining the app, with both platforms frequently changing their requirements, might be difficult.

The problem with the app in it's current state is simple: the more native features that are developed, the more work will have to be put into maintenance in the future. As no one besides me is working on directly, it will inevitably decay; things will change in the rest of ASSISTments or the Android/iOS ecosystem, but the app will not be updated for compatibility. This will lead to parts of the app breaking until it is unusable. To me, the solution seems clear: integrate the mobile app more closely into the ASSISTments ecosystem. Perhaps not this app in particular, just a mobile app in general. To do this, I see a two main options:

The first option would be to re-write the whole of ASSISTments (or at least the tutor) in a modern JavaScript or TypeScript framework, instead of the existing Java and Google

Web Toolkit (GWT). Switching to something more modern and featureful would be a good thing in general, but would also allow for significant code reuse between the web app and the native app, via tools such as NativeScript and React Native. This would be the better option, but is an extremely large amount of work, and that time could probably be put to better use elsewhere.

The second option, which seems more likely, is adding Progressive Web App (PWA) features to the existing web app, removing the need for a native app entirely. If done right, it would require little overhead to maintain, and would provide a native-feeling experience. A PWA can do essentially all the things that our current app does, with much less overhead and no need to download a separate app. The addition of a service worker, as discussed in Section 3.3, is a big step towards being able to run the site as a PWA.

4 SVN to Git Migration

4.1 Introduction

Version control refers to keeping and managing versions and changes to files, typically source code. There are many tools that allow for this functionality, known as Version Control Systems (VCS). Apache Subversion (SVN), started in 2000, was quite popular for many years. Git is a somewhat newer VCS, started in 2005 to manage the Linux kernel source. It has since become one of the most popular VCSs available, especially in Open Source development. SVN is "centralized," meaning that it stores historical data on a central server, which is required for operation. In contrast, Git is a Decentralized Version Control System (DVCS), meaning that every user has a complete copy of the repository. This has distinct advantages for usability, as each user has much more access and control over the historical data for the repository. Since everything is local until pushed, it allows for experimentation without affecting the state of the repository for other users.

Another major difference between SVN and Git is how they store the structure of the repository. Many version control systems have the concept of "branches," which are parallel versions of code with a separate history, of and "tags," which are named versions. SVN is essentially just a versioned filesystem which imposes little structure on the repository, and only has conventions as to how branches and tags should be created. In SVN's "standard" layout shown in Figure 5, "trunk" is the main place for development. Branches and tags are created by making a copy operation from trunk to a folder in "branches" or "tags," respectively. A tag is not functionally different from a branch, although they have different semantic meaning. The "branches" and "tags" folders are not created by default, and branches/tags do not need to be made in them; they can be made anywhere in the repository. In SVN, a copy is a distinct operation from an add, which creates a sort of hard link that shares history and storage between the old and new files, diverging when one of them is changed. Despite it being the way to make branches and tags, it can also just be used to make a normal copy between files or folders. This can make it difficult to keep track of which copies are branches or tags, and which are just actual copies. The standard layout gives some structure, and therefore adds some clarity as to the structure of the repo. SVN also has a completely linear history, with a serial revision number. All changes to the filesystem in the repository create a

revision, so branching and other operations on the repository itself get stored in the history.

```
/
├── /branches
│   └── ...
├── /tags
│   └── ...
└── /trunk
    └── ...
```

Figure 5: SVN "Standard" Layout

In Git, history is stored in a tree of commits, such that the meta-structure of the repository is independent of the history of the tracked data. Commits are referred to by their hash, and are connected to their parents such that if the parent changes, the child's hash will change as well. (The structure is similar to a blockchain, for those familiar with the concept.) Branches and tags are simply pointers to commits in the tree, with a tag referring to a named commit and a branch referring to the string of parents before the marked commit. A simple example of this structure can be seen in Figure 6, and a comparison to SVN in Figure 7. This structure allows for very cheap branches as well as distributed development and history manipulation. Since each user has their own copy of the entire repository, they can manipulate it locally without affecting the state of other developers. Git also has quite a lot of tooling built around it due to its popularity with open source developers, including UIs, hosting sites, continuous integration support, and many other tools.

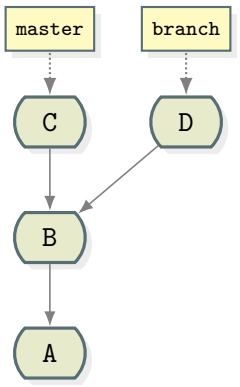


Figure 6: Simple Git History with Branch
(Note: History flows from bottom to top)

ASSISTments has been using a single SVN repository hosted on WPI's FusionForge server since around 2012, which contained all of the code for all of the sub-projects. ASSISTments also has had a lot of developers coming and going due to it's association with WPI. It is

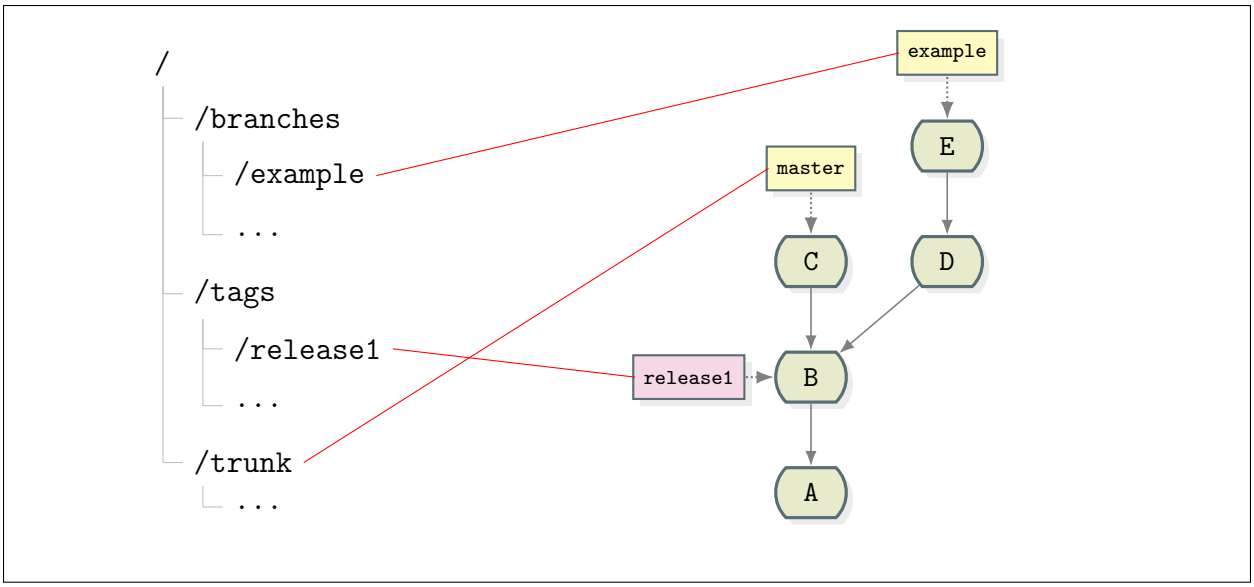


Figure 7: Simple SVN → Git Mapping

therefore somewhat disorganized, and was up to 9762 revisions at the time of the conversion. Due to the significant advantages of Git over SVN, and the fact that most new developers are much more likely to be familiar with Git, it was decided that ASSISTments should switch to Git. Additionally, it was decided that the single repository should be split up in to several because the projects are separated as Maven modules, so they can be built independently, and should never have commits that affect more than one project. Github [3], a popular Git repository hosting site, was chosen as the server because it is well known and it provides some useful features for organizations.

4.2 Tool Selection

When looking for tools to do the migration, the main thing that came up was git's official `git svn` [4] subcommand. I had already been using this for interacting with other parts of ASSISTments, as it provides a nice way to use git commands and workflow with two-way synchronization between SVN and Git. However, it is has some severe downsides, most of all its speed. Earlier in the term I downloaded the entire SVN repo via `git svn`, which took around 9 hours to complete. Since I would have to change the settings and rerun this command frequently, this would be wildly impractical.

The other major problem is that the ASSISTments SVN repo does not follow the standard layout. There are nested folders in `/branches` and `/tags`, making it rather difficult to determine which folders are branches/tags and which just contain branches or tags. Branches are sometimes from `/trunk`, sometimes from logically separate projects (ex `/trunk/WebApp`), and sometimes from subfolders of those projects (ex just the `service` folder of `/trunk/ASSISTmentsServices/loginPortal`). Occasionally, branches are made by re-creating

[3] https://github.com/
[4] https://git-scm.com/docs/git-svn

files in another location rather than correctly copying the folder. While `git svn` has support for defining patterns to match for creating branches and tags, it is not expressive enough to match all of the messy organization in the ASSISTments repository. Additionally, it doesn't convert SVN merges into git merges, meaning that historical commits would be lost without manual post-processing.

Further research lead me to find KDE's `svn2git`[5]. (Note that there are other projects named `svn2git` which are not compatible.) This is based on declarative rule definitions, and is much faster due to using a a local server version of the SVN repo instead of constantly calling out to the remote repo. It also supports merges in history and automatic generation of `.gitignore` files from SVN properties. However, the documentation for `svn2git` was somewhat lacking so I wrote my own docs for the rules syntax as shown in Appendix C, which has now been merged into the official `svn2git` `README.md`.

4.3 Implementation

Yet another difference between git and SVN is how authors are stored. In SVN, the author of a commit is just the username of the person who committed it, whereas Git stores a name and email address since there is no central database of usernames to authenticate against. To translate these, `git2svn` (as well as `git svn` and several other tools) takes an identity-map/authors file, which I created based on the members list on FusionForge. This included a list of real names to email address mappings, which with some manipulation was possible to turn into an identity map. `git2svn` also takes an `--identity-domain` option, which it uses as the email domain when the user does not exist in the authors file. Conveniently, authentication to the SVN repo was done through WPI accounts, so all of the unknown usernames simply map to a `[username]@wpi.edu` email address.

`git2svn` needs a local copy of server version of the SVN repository to operate. To download the repo, I used `svnrdump` from the official subversion package. This creates a "dump file" that can then be loaded with `svnadmin load`, creating a local SVN repo, accessible via the `file://` scheme. This is faster than any tool that calls out to a remote server, as it doesn't need to make network requests for every operation.

Early on, I decided that the rules format of `svn2git` was needlessly verbose, so I wrote a simple Python program to read a YAML file and output a rules file which can be seen in Appendix D. It allows me to use a much more compact representation of the repository and rule definitions, especially with YAM L's inline map syntax, and omit the `/assistments/theNextGeneration` prefix on all of the matching rules. This representation does have the downside of not allowing duplicate keys, which was occasionally a problem for matching rules, but easy to work around by using regex character classes (`[]`) to differentiate.

I started writing rules with a 'catchall' rule, which ignored everything, as the final rule, since `svn2git` throws an error and stops when it finds an unmatched path. This allowed me to develop rules without having to worry about the rest of the repository. To work my way through the structure and history of the SVN repository, I used a few aliases for `svn` subcommands, shown in Listing 13. The most important of these is `slogstop`, which

[5] https://github.com/svn-all-fast-export/svn2git

allowed me to find where paths had been copied from. Using essentially just these commands, I started to piece together a few repositories.

```
# -*- mode: sh; -*-

function resolve_path() {
    if [[ "$1" =~ ^'/assistments/theNextGeneration/' ]]
    then
        echo "file://$PWD/assistments-svn-repo/$1"
    else
        echo "file://$PWD/assistments-svn-repo/assistments/theNextGeneration/$1"
    fi
}

function slogstop() {
    svn log -v --stop-on-copy $(resolve_path "$1")
}

function slogone() {
    svn log -v -l 1 $(resolve_path "$1")
}

function sls() {
    svn ls $(resolve_path "$1")
}
```

Listing 13: Helpful `svn` Aliases

However, it soon became clear that it would be difficult to figure out the history of parts of the repository in isolation, as this technique could not help me find branches of projects, which were scattered around the repository. So I switched to listing all unmatched paths explicitly, which also gave better feedback to help ensure I wasn't missing any paths. To find all the project folders in the repository, I first generated a list of all directories in the svn repo across the entire history with svneverever[6] and the command in Listing 14. I then manually pruned that down to only directories that looked like projects, and placed that in the `unhandled.yaml` file, evaluated last instead of the 'catchall' rule. As I figured out where things went, I removed them from the `unhandled` rules, such that I had a list of things left to do, and could see if I let anything slip through.

```
svneverever --tags --branches --flatten assistments-svn-repo > alldirs.txt
```

Listing 14: `svneverever` Command

During this process, I found `svn2git`'s `prefix` rules, which seemed rather ideal for this use case. They allow combining separate parts of the SVN repository into one Git repository, which seemed quite ideal for this case, as there are a number of projects that start out

[6] https://github.com/hartwork/svneverever

separated, but end up joined together later. However, this rule does not work as I had expected, leading to several odd behaviors that were difficult to debug. Mostly, they lead to files existing in commits when they shouldn't, since I had brought them in as prefixed paths, and then they were added normally later. They also broke branch and merge detection several times, for reasons that I haven't entirely been able to determine.

As an example of a completed ruleset, Listing 15 is a simplified version of the rules for the FindAndAssign project, pulled out from my `misc.yaml` rule file. It creates a repository `findAndAssign`, then matches various paths to create branches and releases. In this case, the releases are essentially tags, although that was not true in all cases. A heavily simplified version of the produced tree can be seen in Figure 8.

```yaml
repositories:
  findAndAssign:

rules:
  /trunk/ASSISTmentsServices/applications/findAndAssign/:
    repository: findAndAssign
    branch: master

# branches
  /branches/dm/0.12/findAndAssign/:
    repository: findAndAssign
    branch: dm/0.12

  /branches/CanvasAndLTI/applications/findAndAssign/:
    repository: findAndAssign
    branch: CanvasAndLTI

  /pgoel/ASSISTmentsServices/applications/findAndAssign/:
    repository: findAndAssign
    branch: pgoel

# releases
  /releases/ASSISTmentsServices/applications/findAndAssign/([^/]+)/:
    repository: findAndAssign
    branch: \1
  /releases/ASSISTmentsServices/applications/findAndAssign: {action: ignore}
```

Listing 15: Extracted `findAndAssign` Rules YAML

To actually run the extraction, I wrote a script called `doImport.sh` to make things easier to run repeatedly, shown in Listing 16. It uses zsh for easier array manipulation, although it could easily be bash or a makefile, as it is rather simple.

Once I had the repositories extracted, there were a few cleanup operations that needed to be done. First, I converted all files to use LF instead of CRLF for line feeds, as this is strongly preferred in Git. On Windows Git will (by default) automatically convert back to CRLF when the repository is cloned. For Java files I also added a trailing newline if there was

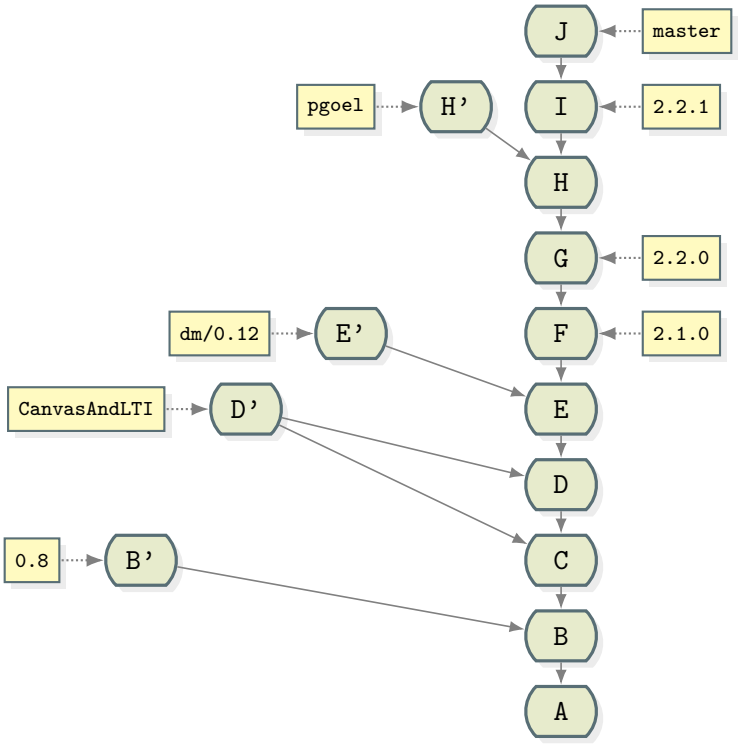


Figure 8: Simplified FindAndAssign Git Tree

```
#!/bin/zsh
# written in zsh for easier array manipulation

set -e # exit on errors

# reset repos dir
rm -rf repos
mkdir repos
cd repos

RULES=(findAndAssign catchall)

# parse yamls into svn2git rules
python3 ../rules/yamlParser.py

# run the import
svn-all-fast-export \
    --identity-map ../authors.txt \
    --identity-domain wpi.edu \
    --rules ${(j:,:)${:-../compiledRules/${^RULES}.rules}} \
    --svn-ignore --add-metadata $@ \
    $PWD/../assistments-svn-repo
```

Listing 16: `SVN-Git-Migration/doImport.sh`

none, and applied consistent a style using Artistic Style (astyle)[7]. These operations can be seen in `cleanup.sh`, shown in Listing 17. The call to `true` at the end prevents `filter-branch` from failing if there are no files, which causes astyle to return an error code. I originally had a number of other operations in this file, but replaced most of them with the call to astyle.

```
#!/bin/bash

# convert CRLF to LF
git ls-files -z | xargs -0r dos2unix -q

# ensure a trailing newline
find . -type f -name '*.java' -print0 | xargs -0r sed -i -e '$a\'

astyle --quiet --indent=spaces=2 --style=allman --indent-switches
 ↪  --break-closing-braces --lineend=linux --preserve-date --suffix=none
 ↪  --recursive "./*.java"

true
```

Listing 17: Line Endings and Style Cleanup Script `SVN-Git-Migration/cleanup.sh`

However, I wanted to apply this across history, as it would simplify diffs between revisions and generally be cleaner, without having a single commit that changes every line in every file. To do this, I used the `git filter-branch` subcommand, the `--tree-filter` argument of which checks out each commit and runs a command on it. I also passed the `--prune-empty` argument, which removes empty commits, as there are a lot of SVN revisions that just add folders which become empty commits in Git because Git only tracks files. However, this operation is extremely slow, taking several hours to run, as it needs to checkout each revision, and then run astyle on it. The solution to this was to run the operation on `rambo.wpi.edu`, a WPI compute server available to students. This provided a powerful CPU with many cores, as well as a SSD work disk and lots of RAM, allowing me to parallelize the process. I used GNU Parallel [3] to do this, which handles jobs quite nicely.

After running my cleanup script, I wrote another script that cleaned up merged branches and the unneeded refs created by `git2svn` and `filter-branch`. This script is shown in Listing 18, and was also run with `parallel`, although it was not particularly slow. The import process is further documented in Appendix E, the README I wrote.

Finally, I repacked all of the repositories, which improves their size and clone speed, with the command shown in Listing 19. This doesn't benefit quite as much from `parallel`, as it does it's own multithreading, but I ran it with 4 jobs because it never seemed to use more than about 10 of the 32 cores when run one at a time. I'm not entirely sure that this is necessary since this ended up being hosted on Github which may do this on their servers, but it only takes a few minutes and reduces the size by almost half, as shown in Table 1.

The result of this process was the 36 Git repositories shown in Table 2. This table only shows the master branches of each repository; many had several branches and tags as well.

[7] http://astyle.sourceforge.net/

```bash
#!/bin/bash

# remove backup tags
git tag | grep '^backups/' | xargs git tag -d

# remove git-filter-branch backup tags
git for-each-ref --format="%(refname)" refs/original/ | xargs -n 1 git update-ref -d

# remove merged branches
git branch --merged | grep -v master | xargs git branch -d
```

Listing 18: Remove Unused Branches and Refs `SVN-Git-Migration/repoCleanup.sh`

```bash
git gc; git repack -adf --window 250 --depth 250
```

Listing 19: Git Repack Command

Table 1: `time` and `du` Output for Commands in the Migration Process

Command	User	System	CPU	Total	Total Repository Size (GB)
`svn2git`	414.11s	47.91s	82%	9:21.38	1.3
`cleanup.sh`	4317.25s	2898.38s	189%	1:03:19.23	2.0
`repoCleanup.sh`	0.46s	2.79s	690%	0.471	2.0
`git gc; git repack`	423.49s	43.39s	312%	2:29.50	1.1
Total	5155.31 s	2992.47 s		01:15:09	

Table 2: Mapping of Created Git Repos to SVN Paths

Git Repository	SVN Path for `master` branch
ARRSService	`/trunk/ASSISTmentsServices/applications/ARRSService`
Assets	`/trunk/ASSISTmentsServices/assets`
AWSService	`/trunk/ASSISTmentsServices/applications/data_modeling/awsService`
CommonUI	`/trunk/ASSISTmentsServices/CommonUI`
CompetitionSite	`/branches/CompetitionSite`
ContentBuilder	`/trunk/ASSISTmentsServices/applications/builders/contentBuilder`
ContentMakerRESTfulService	`/trunk/ASSISTmentsServices/applications/ContentMakerRESTfulService`
CAS	`/trunk/ASSISTmentsServices/core`
DataDumper	`/branches/analytics/DataDumper`
Documents	`/trunk/documents`
ExperimentBuilder	`/trunk/ASSISTmentsServices/applications/ExperimentBuilder`
FindAndAssign	`/trunk/ASSISTmentsServices/applications/findAndAssign`
FlywayRestService	`/trunk/ASSISTmentsServices/applications/FlywayRestService`
GenerateSkillBuilder	`/trunk/ASSISTmentsServices/applications/GenerateSkillBuilder`
ImageForward	`/trunk/ASSISTmentsServices/applications/samples/imageForward`
LiveChart	`/trunk/ASSISTmentsServices/applications/LiveChart`
LoginPortal	`/trunk/ASSISTmentsServices/LoginPortal`
LRS	`/trunk/ASSISTmentsServices/applications/LRS`
Operations	`/trunk/ASSISTmentsServices/operations`
PeerAssistService	`/trunk/ASSISTmentsServices/applications/PeerAssistService`
PGStats	`/trunk/ASSISTmentsServices/applications/monitoring/pgStats`
PGWebApp	`/trunk/ASSISTmentsServices/applications/samples/pgWebApp`
Placements	`/trunk/ASSISTmentsServices/applications/placementsService` `/trunk/ASSISTmentsServices/applications/placementsWebApp`
ProblemSetRecommender	`/trunk/ASSISTmentsServices/applications/ProblemSetRecommender`
QuizAssistWebApp	`/trunk/ASSISTmentsServices/applications/QuizAssistWebApp`
ResourceService	`/trunk/ASSISTmentsServices/applications/resourceService`
Rest	`/trunk/ASSISTmentsServices/rest`
ServiceAdmin	`/trunk/ASSISTmentsServices/applications/ServiceAdmin`
Survey	`/trunk/ASSISTmentsServices/applications/survey`
TeacherUsage	`/trunk/ASSISTmentsServices/applications/teacherUsage`
TNG	`/trunk/ASSISTmentsServices/applications/TNG`
Tools	`/trunk/tools`
Tutor	`/trunk/ASSISTmentsServices/applications/tutor`
UtilTools	`/trunk/ASSISTmentsServices/applications/UtilTools`
WorkflowService	`/trunk/ASSISTmentsServices/applications/WorkflowService`
XmlBuilder	`/trunk/ASSISTmentsServices/applications/builders/XmlBuilder`

With all of the repositories cleaned up, they just needed to be uploaded to Github. However, since there are 36 repositories created from SVN, it would be very tedious to create and upload them all manually. While Github does not have a bulk import function, they do have an API, so I wrote a Python script to create Github repos and push to them. This script requires PyGithub[8], and can be seen in Listing 20.

There are a number of workflows that work well with Git, but the most popular style is to have long-running `master` and `dev` branches, with short-lived topic/feature branches. On Github, it is possible to enforce that all code goes through a Pull Request to get into `dev` via branch protection, which allows enforcing various rules on a branch. To set the repositories up for this workflow, I wrote another script using the Github API to create a `dev` branch and apply branch protection to it and `master`. It is written to run across all the repos in an organization, except archived repos and those explicitly listed. It adds a simple rule to enforce pull requests and reviews to merge into `dev`, and have only members of the Admin team push to `master`. The code can be seen in Listing 21.

4.4 Future Work

I didn't include a number of folders from the ASSISTments SVN repo, primarily branches that didn't connect to a folder in trunk. Those unhandled paths are listed in the `SVN-Git-Migration/rules/unhandled.yaml` file, along with a number of empty paths. If these are needed in the future, I have hopefully provided enough information in the README, shown in Appendix E, to help someone add those and re-run the process. There are also a few repositories in SVN that have some prior history in Git elsewhere on FusionForge; these could be integrated into the current repositories without too much work beyond tracking them down.

Continuous Integration (CI) provides constant testing of code as well as automatic deployment. With Github, it is possible to enforce that tests pass before a branch can be merged into `dev`, enforce style checks, automatically test and deploy with every push to `dev`, and a number of other improvements to the development workflow. Setting one of the various CI platforms shouldn't be too hard, but there are some issues with the way unit tests are currently written that makes them a lot less effective than they should be.

[8] https://github.com/PyGithub/PyGithub

```python
#!/usr/bin/env python3

import os
import subprocess

from github import Github

from githubAuth import GITHUB_TOKEN

g = Github(GITHUB_TOKEN)

def createRepo(name):
    org = g.get_organization('ASSISTments')
    org.create_repo(name,
                    private=True,
                    has_downloads=False,
                    has_issues=False,
                    has_wiki=False,
                    has_projects=False)

# Use with caution!
def deleteRepo(name):
    org = g.get_organization('ASSISTments')
    org.get_repo(name).delete()

for repo in os.listdir('repos'):
    repoPath = os.path.join('repos', repo)
    if os.path.isdir(repoPath):
        print(repo)
        try:
            createRepo(repo)
        except:
            print("failed to create repo")

        gitBaseCommand = ["git",
                          "-C", repoPath,
                          "push", "github:ASSISTments/" + repo + ".git"]
        subprocess.call(gitBaseCommand + ["--all"])
        subprocess.call(gitBaseCommand + ["--tags"])
```

Listing 20: Repository Creation Script `SVN-Git-Migration/createRepos.py`

```python
#!/usr/bin/env python3

import os
import subprocess

from github import Github

from githubAuth import GITHUB_TOKEN

g = Github(GITHUB_TOKEN)

ignoredRepos = ['SVN-Git-Migration',
                'XmlBuilder',
                'Operations',
                'Documents',
                'Assistments-Mobile']

def fixBranches(org, repo):
    admins = org.get_team(3338516)
    admins.add_to_repos(repo)
    admins.set_repo_permission(repo, 'admin')

    master = repo.get_branch('master')

    # if there is no dev branch
    try:
        dev = repo.get_branch('dev')
    except:
        # create dev branch from master
        repo.create_git_ref(ref='refs/heads/' + 'dev',
                            sha=master.commit.sha)
        dev = repo.get_branch('dev')

    dev.edit_protection(strict=True,
                        dismiss_stale_reviews=True,
                        required_approving_review_count=1,
                        team_push_restrictions=[admins.slug])

    master = repo.get_branch('master')
    master.edit_protection(strict=True,
                           team_push_restrictions=[admins.slug])

    repo.edit(default_branch="dev")

    dev = repo.get_branch('dev')

org = g.get_organization('ASSISTments')
for repo in org.get_repos():
    if not (repo.archived or repo.name in ignoredRepos):
        print(repo)
        fixBranches(org, repo)
```

Listing 21: Branch Protection Script `SVN-Git-Migration/createControlledBranches.py`

5 References

[1] OWASP Foundation. Cross-site scripting (xss). https://www.owasp.org/index.php/Cross-site_Scripting_(XSS), Jun 2018. Accessed 2019-06-25.

[2] Mozilla. Using the application cache. https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache, Mar 2019. Accessed 2019-08-08.

[3] O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb 2011.

Appendices

A Teacher Assist `inject.js`

```
window.tainjected = true;

function handleSubmit(event) {
  // get all data attributes and add to request
  const prefix = "data-";
  let data_attrs = [...event.target.parentElement.parentElement.attributes]
    .filter(a => a.name.startsWith(prefix))
    .reduce((obj, a) => {
      obj[a.name.substring(prefix.length)] = a.value;
      return obj;
    }, {});

  fetch("/PeerAssistService/get_peer_tutoring/submit_usefulness_report", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "assistments-auth": 'partner="PeerAssistService"'
    },
    body: JSON.stringify({
      did_help: event.target.value,
      ...data_attrs
    })
  });

  let thanks = document.createElement("div");
  thanks.textContent = "Thanks for your feedback!";
  event.target.parentElement.replaceWith(thanks);
}

function makeButton(text, value) {
  let button = document.createElement("button");
  button.className = "gwt-Button";
  button.textContent = text;
  button.setAttribute("value", value);
  button.addEventListener("click", handleSubmit);
  button.style["margin-right"] = "1em";
  return button;
}

function tryReplace() {
  // should never have more than one at a time, but just to be sure
  [...document.querySelectorAll(".teacherassist-inject")].map(async el => {
    // remove XSS img tag and CSS class
    el.removeChild(el.firstChild);
    el.className = "";

    // TODO: could create this once and clone it
```

```
    let buttonDiv = document.createElement("div");
    buttonDiv.append(makeButton("No", "No"), makeButton("Yes", "Yes"));
    buttonDiv.style["font-size"] = "10px";
    el.style["margin-top"] = "2em";
    el.style["text-align"] = "center";

    el.append(
      document.createElement("hr"),
      el.getAttribute("data-prompt"),
      buttonDiv
    );
  });
}

// replace the already existing element (ie the one that loaded this)
tryReplace();

// TODO: narrow the scope of observation for better performance
new MutationObserver(tryReplace).observe(document, {
  // watch the whole page
  childList: true,
  subtree: true
});
```

B Service Worker Linker

From `tutor/client/src/main/java/org/assistments/gwt/student/tutor/linkers/ServiceWorkerL`

```
/*
 * Based on SimpleAppCacheLinker from gwt
 *                                                                                ↩
 ↪  https://code.google.com/p/google-web-toolkit/source/browse/trunk/dev/core/src/com/google/gwt/core/
 */
package org.assistments.gwt.student.tutor.linkers;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.google.gwt.core.ext.Linker;
import com.google.gwt.core.ext.LinkerContext;
import com.google.gwt.core.ext.TreeLogger;
import com.google.gwt.core.ext.UnableToCompleteException;
```

```
21 import com.google.gwt.core.ext.linker.AbstractLinker;
22 import com.google.gwt.core.ext.linker.Artifact;
23 import com.google.gwt.core.ext.linker.ArtifactSet;
24 import com.google.gwt.core.ext.linker.EmittedArtifact;
25 import com.google.gwt.core.ext.linker.EmittedArtifact.Visibility;
26 import com.google.gwt.core.ext.linker.LinkerOrder;
27 import com.google.gwt.core.ext.linker.LinkerOrder.Order;
28 import com.google.gwt.core.ext.linker.Shardable;
29 import com.google.gwt.core.ext.linker.impl.SelectionInformation;
30 import com.google.gwt.core.ext.linker.Transferable;
31
32 @Shardable
33 @LinkerOrder(Order.POST)
34 public class ServiceWorkerLinker extends AbstractLinker
35 {
36   @Override
37   public String getDescription()
38   {
39     return "ServiceWorkerLinker";
40   }
41
42   @Transferable
43   private class PermutationArtifact extends Artifact<PermutationArtifact>
44   {
45     private static final long serialVersionUID = -2097933260935878782L;
46     private final Set<String> permutationFiles;
47     private final String permutationName;
48
49     public PermutationArtifact(Class<? extends Linker> linker,
50                                String permutationName,
51                                Set<String> permutationFiles)
52     {
53       super(linker);
54       this.permutationName = permutationName;
55       this.permutationFiles = permutationFiles;
56     }
57
58     @Override
59     public int hashCode()
60     {
61       return permutationFiles.hashCode();
62     }
63
64     @Override
65     protected int compareToComparableArtifact(PermutationArtifact o)
66     {
67       return permutationName.compareTo(o.permutationName);
68     }
69
70     @Override
71     protected Class<PermutationArtifact> getComparableArtifactType()
72     {
73       return PermutationArtifact.class;
```

```
74      }
75
76      public Set<String> getPermutationFiles()
77      {
78        return permutationFiles;
79      }
80
81      public String getPermutationName()
82      {
83        return permutationName;
84      }
85    }
86
87    // Add any other files not handled by GWT here
88    // ex. cross domain resources
89    static protected String[] staticCachedFiles = {};
90
91    @Override
92    public ArtifactSet link(TreeLogger logger,
93                            LinkerContext context,
94                            ArtifactSet artifacts,
95                            boolean onePermutation) throws UnableToCompleteException
96    {
97      ArtifactSet toReturn = new ArtifactSet(artifacts);
98
99      if (artifacts.find(SelectionInformation.class).isEmpty())
100     {
101       // hosted mode
102       return toReturn;
103     }
104
105     if (onePermutation)
106     {
107       toReturn.add(
108         new PermutationArtifact(
109           ServiceWorkerLinker.class,
110           artifacts.find(SelectionInformation.class).first().getStrongName(),
111           getOutputFiles(artifacts)));
112
113       return toReturn;
114     }
115
116     // get all permutation-specific artifacts
117     Set<PermutationArtifact> permutationArtifacts =
118       artifacts.find(PermutationArtifact.class);
119
120     // get all permutation-specific files
121     Set<String> allPermutationFiles = permutationArtifacts.stream()
122                                       .flatMap(artifact ->                          ↩
                                           ↪  artifact.getPermutationFiles().stream())
123                                       .collect(Collectors.toSet());
124
125     // remove permutation-specific files from all artifacts to get common files
```

```
126     Set<String> commonFiles = getOutputFiles(artifacts);
127     commonFiles.removeAll(allPermutationFiles);
128
129     // make a map of permutation names to permutation-specific files
130     Map<String, Set<String>> permutationFilesMap = permutationArtifacts.stream()
131         .collect(Collectors.toMap(PermutationArtifact::getPermutationName,
132                                   PermutationArtifact::getPermutationFiles));
133
134     InputStream templateStream = getClass().getClassLoader()
135                                  .getResourceAsStream("service-worker.template.js");
136
137     try
138     {
139       ObjectMapper objectMapper = new ObjectMapper();
140
141       String out = Stream.concat(
142                       Stream.of(
143                         "const STATIC_FILES = " +                                    ↩
                           ↪  objectMapper.writeValueAsString(staticCachedFiles) +    ↩
                           ↪  ";",
144                         "const COMMON_ARTIFACTS = " +                                ↩
                           ↪  objectMapper.writeValueAsString(commonFiles) + ";",
145                         "const PERMUTATION_ARTIFACTS = " +                           ↩
                           ↪  objectMapper.writeValueAsString(permutationFilesMap) +  ↩
                           ↪  ";"),
146                       new BufferedReader(new InputStreamReader(templateStream)).lines())
147                     .collect(Collectors.joining("\n"));
148
149       toReturn.add(emitString(logger, out, "service-worker.js"));
150     }
151     catch (JsonProcessingException e)
152     {
153       logger.log(TreeLogger.ERROR, "Failed to create JSON for Service Worker", e);
154       throw new UnableToCompleteException();
155     }
156
157     return toReturn;
158   }
159
160   protected Set<String> getOutputFiles(ArtifactSet artifacts)
161   {
162     return artifacts.find(EmittedArtifact.class).stream()
163             .filter(artifact -> artifact.getVisibility() == Visibility.Public)
164             .map(EmittedArtifact::getPartialPath)
165             .filter(artifact -> shouldArtifactBeInManifest(artifact))
166             .collect(Collectors.toSet());
167   }
168
169   protected boolean shouldArtifactBeInManifest(String pathName)
170   {
171     return !(pathName.endsWith("symbolMap")
172              || pathName.endsWith(".xml.gz")
173              || pathName.endsWith("rpc.log")
```

```
174                 || pathName.endsWith("gwt.rpc")
175                 || pathName.endsWith("manifest.txt")
176                 // || pathName.endsWith("manifest.html")
177                 || pathName.endsWith("cssmap")
178                 || pathName.contains("tiny_mce_wiris")
179                 || pathName.contains("soycReport")
180                 || pathName.startsWith("rpcPolicyManifest")
181                 || pathName.equals("compilation-mappings.txt")
182                 || pathName.endsWith(".devmode.js")
183                 || pathName.endsWith(".cache.js.gz"));
184   }
185 }
```

C The 'Rules' Section of `svn-all-fast-export/svn2git/README.md`

(I wrote this section, which has now been merged into `svn-all-fast-export/svn2git`)

C.1 `create respository`

```
create repository REPOSITORY NAME
  [PARAMETERS...]
end repository
```

`PARAMETERS` is any number of:

- `repository TARGET REPOSITORY` Creates a forwarding repository , which allows for redirecting to another repository, typically with some `prefix`.
- `prefix PREFIX` prefixes each file with `PREFIX`, allowing for merging repositories.
- `description DESCRIPTION TEXT` writes a `DESCRIPTION TEXT` to the `description` file in the repository

C.2 `match`

```
match REGEX
  [PARAMETERS...]
end match
```

Creates a rule that matches paths by `REGEX` and applies some `PARAMETERS` to them. Matching groups can be created, and the values used in the parameters.

`PARAMETERS` is any number of:

- `repository TARGET REPOSITORY` determines the repository
- `branch BRANCH NAME` determines which branch this path will be placed in. Can also be used to make lightweight tags with `refs/tags/TAG NAME` although note that tags

in SVN are not always a single commit, and will not be created correctly unless they are a single copy from somewhere else, with no further changes. See also `annotate true` to make them annotated tags.

- `[min|max] revision REVISION NUMBER` only match if revision is above/below the specified revision number
- `prefix PREFIX` prefixes each file with `PREFIX`, allowing for merging repositories. Same as when used in a `create repository` stanza.
- Note that this will create a separate commit for each prefix matched, even if they were in the same SVN revision.
- `substitute [repository|branch] s/PATTERN/REPLACEMENT/` performs a regex substitution on the repository or branch name. Useful when eliminating characters not supported in git branch names.
- `action ACTION` determines the action to take, from the below three:
- `export` I have no idea what this does
- `ignore` ignores this path
- `recurse` tells svn2git to ignore this path and continue searching it's children.
- `annotate true` creates annotated tags instead of lightweight tags

C.3 `include FILENAME`

Include the contents of another rules file

C.4 `declare VAR=VALUE`

Define variables that can be referenced later. `${VAR}` in any line will be replaced by `VALUE`.

D SVN-Git-Migration/rules/yamlParser.py

```
1  #!/usr/bin/env python3
2
3  import os
4  import yaml
5
6  # move into rules directory
7  os.chdir(os.path.dirname(os.path.realpath(__file__)))
8  OUTPUT_DIR='../compiledRules/'
9  os.makedirs(OUTPUT_DIR, exist_ok=True)
10
11 RULE_PATH_PREFIX='/assistments/theNextGeneration'
```

```
12
13  def format_props(props):
14      if props:
15          return [f'  {k} {v}' for k, v in props.items()]
16      else:
17          return []
18
19  # convert each yaml into a svn2git rules file
20  for yaml_filename in [y for y in os.listdir() if y.endswith('.yaml')]:
21      with open(yaml_filename) as input_f, \
22           open(OUTPUT_DIR + yaml_filename[:-5] + '.rules', 'w') as output_f:
23          y = yaml.safe_load(input_f)
24          if y['repositories']:
25              for repo, props in y['repositories'].items():
26                  output_f.write(
27                      '\n'.join(['create repository ' + repo,
28                                 *format_props(props),
29                                 'end repository\n\n']))
30
31          for rule, props in y['rules'].items():
32              if not rule.startswith('/assistments/'):
33                  rule = RULE_PATH_PREFIX + rule
34              output_f.write(
35                  '\n'.join(['match ' + rule,
36                             *format_props(props),
37                             'end match\n\n']))
```

E SVN-Git-Migration/README.md

E.1 ASSISTments SVN -> Git Migration

This is a set of rules and helper scripts for migrating the ASSISTments SVN monorepo into a set of separate git repositories.

E.1.1 Requirements

- KDE's svn2git[9]
- Python 3
- ZSH (although this could be pretty easily avoided, it made writing the script easier)

Optional, but recommended (Artistic Style)[http://astyle.sourceforge.net/astyle.html]
Optional, but wow would that be slow [gnu parallel](https://www.gnu.org/software/parallel/)

E.1.2 The Import Process

1. We need a local server repo (ie not a normal SVN working copy) to operate on, which we can make via:

[9] <https://github.com/svn-all-fast-export/svn2git>

```
svnadmin create assistments-svn-repo
rsvndump --username=<your username> https://fusion.wpi.edu/svn/assistments-rep/ |
 ↪  svnadmin load assistments-svn-repo
```

As other people commited to this repository, this local copy becomes out of date, and can be brought back up to date with:

```
svnrdump dump --username=<your username>
 ↪  https://fusion.wpi.edu/svn/assistments-rep/ --incremental -r <old revision +
 ↪  1>:HEAD | svnadmin load assistments-svn-repo
```

2. Next, run `doImport.sh` to perform the actual import. It will pass arguments into the `svn-all-fast-export` command, so for instance `-stats` is prettyhelpful. The other part of the following command just jams the stdout/stderr into separate files as well as printing them, for easier debugging later.

```
./doImport.sh --stats > >(tee import.log) 2> >(tee import-stderr.log >&2)
```

You could also just run `svn-all-fast-export` manually, in which case you will have to convert the rules from YAML to the expected rules format with `python3 ../rules/yamlParser.py`. The rules were written in YAML instead because it is more compact and easier to work with, and allowed me to prefix all the rules with `/assistments/theNextGeneration/` automatically, which had been getting tedious.

3. Next, there are some cleanup scripts. `cleanup.sh` is intended to be run via `git filter-branch` on each repo. It preforms a few cleanup operations, mostly tidying up whitespace and applying a style with Artistic Style. It is intended to be run like this:

```
git filter-branch --prune-empty --tree-filter /path/to/cleanup.sh -- --all
```

Just running `git filter-branch -prune-empty` by itself would be suggested even if you don't want the whitespace/style cleanup, as it will remove empty commits (primarily empty directory creation in SVN) from history.

In either case, git filter-branch is rather slow. Since there are a lot of repositories to work with, I would recommend using GNU parallel to run a lot of these at once.

```
parallel "cd {}; git filter-branch --prune-empty -f --tree-filter $PWD/cleanup.sh
 ↪  -- --all" ::: repos/*/
```

4. There is a further cleanup script to be run on each repo that deletes merged branches and backup refs from svn2git called `repoCleanup.sh`. It can be run with or without GNU parallel:

```
/path/to/repoCleanup.sh
# OR
parallel "cd {}; $PWD/repoCleanup.sh" ::: repos/*/
```

5. Next, repack the repos to make them smaller/more efficient. Again, GNU parallel can be used, but since repacking already does its own parallel processing, it is not as helpful.

```
git gc; git repack -adf --window 250 --depth 250
# OR
parallel -j4 "cd {}; git gc; git repack -adf --window 250 --depth 250" ::: repos/*/
```

I ran this with with only 4 jobs (`-j4`), since it seems to spike around 1000% CPU usage on rambo (ie 10 cores) and it has 32 total cores, so this should mostly help rather than hinder.

6. Finally, upload the repos to Github. I've written a script to assist with creating and uploading repositories to Github called `createRepos.py`. To use this script, you will need PyGithub, and a Github personal access token[10] with at least the `repo` permission. Set your token in a file called `githubAuth.py`, containing the following:

```
GITHUB_TOKEN = '<your token here>'
```

Then, you can run the repo creation script:

```
python3 createRepos.py
```

E.1.3 Writing more rules

Mostly, my best advice would be to go read the docs[11]. They are, however, somewhat lacking.

TODO: finish this section

Helpful tools

svn

I've got some aliases for svn commands that I used frequently in the `aliases` file. They mostly just handle prefixing the path argument with the full `file://` path to the repo, plus the `/assistments/theNextGeneration/` prefix if needed. You should just be able to run `source aliases` in a bash-like shell.

- slogstop is `svn log -stop-on-copy`, which will stop when it finds a copy operation. It is good for finding where something (like a branch) came from.
- slogone is `svn log -l 1`, which gets exactly one commit. (The most recent one, unless you use `<@revision number>`)
- sls is `svn ls`

[10] <https://github.com/settings/tokens>

[11] <https://techbase.kde.org/Projects/MoveToGit/UsingSvn2Git#How_rulesets_work>

svneverever

svneverever[12] is an incredibly useful tool which creates a list of all the directories in the repo throughout history, along with the revision they were created and deleted. I mostly used it to create this list once and then pruned it down, but it has more features that I didn't really use that much.

```
svneverever --tags --branches --flatten assistments-svn-repo > alldirs.txt
```

`svn-all-fast-export` itself:

When run with only a "catchall" ruleset, svn2git will provide a nice list of all copy operations. These sometimes correspond to branch/merge operations, but frequently not. Mostly it can be faster than trying to read through the `svn log` output.

```
# catchall.rules
match /*/
end match
```

```
svn-all-fast-export --rules catchall.rules $PWD/assistments-svn-repo 2>&1 | grep "was copied" > copies.txt
```

That catchall rules file is also pretty helpful for running one rules file at a time, as it is much faster.

[12] <https://github.com/hartwork/svneverever>