VERSION CONTROL GRAPHICAL INTERFACE FOR OPEN ONDEMAND

by

HUAN CHEN

Submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

AUGUST 2018

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

Huan Chen

candidate for the degree of Master of Science

Committee Chair

Chris Fietkiewicz

Committee Member

Christian Zorman

Committee Member

Roger Bielefeld

Date of Defense

June 27, 2018

TABLE OF CONTENTS

List of Figures

ACKNOWLEDGMENT

This project would not have been possible without the support of many people. Many thanks to my adviser, Chris Fietkiewicz, who helped me throughout the project. Also, this work made use of the High-Performance Computing Resource in the Core Facility for Advanced Research Computing at Case Western Reserve University. We thank Hadrian Djohari, Emily Dragowsky, and the other HPC staff for their assistance.

Version Control Graphical Interface for Open OnDemand

Abstract

by

HUAN CHEN

For researchers who are not familiar with the necessary tools such as the Linux operating system and job management software, the use of a high-performance computing (HPC) cluster is challenging. Of those who are using HPC systems, the majority are not trained specifically in computer programming. The Ohio Super Computer Center has addressed these issues through the development of an open source, web-based GUI called Open OnDemand. In order to improve the development experience for researchers, we sought to: (1) install and configure Open OnDemand on a private cluster at Case Western Reserve University in order to evaluate the difficulty level of deployment; and (2) develop a custom web-based interface for use of the popular Git version control system. Our module successfully integrates with the existing Open OnDemand interface and provides common version control operations that can be used during typical HPC workflows.

CHAPTER 1: INTRODUCTION

The use of high-performance computing (HPC) clusters in various fields of research involves a user experience that is drastically different from what many researchers are accustomed to with personal computers. The current era of desktop computing and graphical user interfaces (GUIs) has allowed researchers to avoid the limitations that were imposed in the previous era of mainframe computing.

The increased use of cloud-based resources has, perhaps, increased researchers' familiarity with several important concepts in remote computing, such as servers, clients, and shared file systems. However, HPC systems typically are not accessible through the types of GUIs that are common with cloud-based tools. By looking at allocations for the eXtreme Science and Engineering Discovery Environment (XSEDE), we observed that nearly 70% are in primary fields other than computer science and computing science. As a result, most users likely have no previous experience with typical HPC interface tools such as Linux scripting and command line terminals.

Only a few products have addressed this issue. Globus is a cloud-based GUI that provides functions to transfer files to any online nodes, provide data to users and co-workers and access data in a reliable, secure and high-performance way. The Globus service helps researchers to work remotely and provides an easy way for researchers to sync progress with teammates [1]. The eXtreme Science and Engineering Discovery Environment (XSEDE) is a single virtual system that enables scientists to share data, expertise and compute

resources [2]. XSEDE provides a simple GUI for users to check information, use secure shell (SSH), manage user accounts, find real-time information about XSEDE services, and access various services [3], such as file browsing and editing, job submission, and VNC application access [4]. By using this system, users can get access to supercomputers, collections of data and expert support [5]. The FUJITSU software HPC gateway [6] makes it possible to turn a web browser into a desktop, which can make it familiar for those users without experience with command line interfaces. Users can run scripts, submit jobs and browse and edit files in a fully visualized way. There are also pre-built applications that can simplify the work.

Open OnDemand is an open source software project, developed at the Ohio Supercomputer Center (OSC), based on the “OSC Open OnDemand” Platform [5]. Open OnDemand (OOD) provides a web-based GUI that contains basic functions, including file browsers, file editors, a job manager, remote shell and more. As open source software, administrators are free to deploy OOD on their own systems, and developers are free to enhance it. We had previous experience using an initial offering of OOD as users of the Ohio Supercomputing Center, and it proved to be researcher-friendly, especially for those who lack experience with HPC. So we sought to evaluate the use of Open OnDemand on an HPC cluster at Case Western Reserve University.

Our first goal is to evaluate the process of deploying the Open OnDemand on an existing HPC system. We choose the SLURM as the job management software to be used because it is the primary job management system currently

in use on the Case HPC system. We rely mainly on the documentation that Open OnDemand provides [8].

Our second goal is to evaluate the extensibility of Open OnDemand. We develop an integrated GUI application to test the developer mode that Open OnDemand supports and the feasibility of integrating the application. The application of the integrated GUI was chosen to be a version control manager for Git. Version control is a software development tool that tracks file changes and provides controlled access to previous versions of the files in a project [18]. It is widely used in software companies, open source projects and researchers' code development process. The advantages of using version control in research computing and HPC include file documentation, reproducibility, and coordination between multiple researchers. Git is one of the most popular version control products that is open source and provides an easy-to-use version control structure. For those researchers who lack coding experience and software development knowledge, they may have limited knowledge of version control practices. We chose version control as an application in order to promote it to more researchers with easier usage through web-based GUI. However, version control software has never been directly integrated into HPC management software, including Open OnDemand.

CHAPTER 2: METHODS

2.1 Installation for Environments and Open OnDemand

Pre-installation is needed before developing applications for Open OnDemand. First, a cluster management and job scheduling system is needed. Here we choose the SLURM as it is widely used and is the main system used on Case Western Reserve University HPC system. Additionally, it is specifically supported by Open OnDemand. Following SLURM installation, Open OnDemand needs to be installed, and development mode needs to be enabled. All installation, developments, and tests are done in CentOS 7 and RHEL 7.4.

2.1.1 Install SLURM

2.1.1.1 Create User

Two new users need to be created in all nodes for easier installation of SLURM. The user ID (UID) and group ID (GID) should be consistent in every node. The two users defined here are “slurm” and “munge”.

First add a user named “munge”:

```
export MUNGEUSER=1001
groupadd -g $MUNGEUSER munge
useradd  -m -c "MUNGE Uid 'N' Gid Emporium" -d /var/lib/munge -u \
```

```
        $MUNGEUSER -g munge  -s /sbin/nologin munge
```

Then add a user named “slurm”:

```
export SLURMUSER=1002

groupadd -g $SLURMUSER slurm

useradd  -m -c "SLURM workload manager" -d /var/lib/slurm -u \

        $SLURMUSER -g slurm  -s /bin/bash slurm
```

2.1.1.2 Install and Configure Munge

Munge is an authentication service that can help create and validate credentials and is specifically designed to work with HPC systems [19]. It is used for SLURM to communicate among nodes safely.

First, install munge with yum:

```
sudo yum install munge munge-libs munge-devel -y
```

Then create the secret key:

```
dd if=/dev/urandom bs=1 count=1024 > /etc/munge/munge.key
```

Then send it to all other nodes.

Now add proper permission in every nodes:

```
chown -R munge: /etc/munge/ /var/log/munge/

chmod 0700 /etc/munge/ /var/log/munge/
```

Lastly, set munge to start on boot and start it:

```
systemctl enable munge

systemctl start munge
```

2.1.1.3 Install and Configure SLURM

First, download the SLURM source file from the official download page [20]. Then build it with the RPMBUILD function. The “version” in the command should be set using the version number of the downloaded SLURM:

```
rpmbuild -ta slurm-version.tar.bz2
```

Now go to the built files directory:

```
cd ~/rpmbuild/RPMS/x86_64
```

Install all built files:

```
sudo yum --nogpgcheck localinstall slurm*
```

Then go to the “Slurm Configuration Tool – Easy Version” page [21]. In this page, control machines and compute machines should be specified, and everything else can remain unchanged for now.

After submitting the browser based configuration form, the page will show the content of the configuration file, create a file named slurm.conf in /etc/slurm and copy the content in the webpage into the new file. Every node that has SLURM installed should have the same copy of the file.

A Cgroup.conf file should be set in the /etc/slurm directory. We use the official example file here which can be found in the same directory.

Then the permissions should be set for log files.

Now the SLURM can be set to start on boot and be started on every node:

On compute nodes:

```
sudo systemctl enable slurmd
sudo systemctl start slurmd
```

On the control node:

```
sudo systemctl enable slurmctld
sudo systemctl start slurmctld
```

Now SLURM should be able to manage the nodes and batch the jobs.

2.1.1.4 Enable Accounting

SLURM accounting is used for recording all jobs and job steps executed [22]. To use Open OnDemand with SLURM, it is necessary to add the accounting function to SLURM. Here we set the accounting to save job information in the database.

First, MariaDB needs to be installed:

```
yum install mariadb-server mariadb-devel -y
```

Then enable and start the MariaDB server:

```
sudo systemctl enable mariadb
sudo systemctl start mariadb
```

Then configure the MariaDB after running the command below, where the root password should be set, and all other options can be set to default:

```
mysql_secure_installation
```

Then MariaDB can be logged into with the password:

```
mysql -u root -p
```

Then add a database user named “slurm” and grant access to the corresponding database. Detailed settings are provided in the SLURM Accounting and Resource Limits page [22].

Now set the configuration files for SLURM and SLURMDBD to start accounting.

For SLURM, the /etc/slurm/slurm.conf file needs to be changed. The value of JobAcctGatherType needs to be set to jobacct_gather/linux, and the value of the AccountingStorageType needs to be set to accounting_storage/slurmdbd. Note that, after changes, this file should be sent to all nodes. An accounting host may be set if needed.

For SLURMDBD, first copy the /etc/slurm/slurmdbd.conf.example and rename it to slurmdbd.conf. Change the DbdAddr and DbdHost to have the necessary values. Make sure the slurm user and storage user are both slurm,

and change the storage password to the corresponding password. Change the value of StorageType to accounting_storage/mysql.

For both SLURM and SLURMDBD, all used ports should be enabled. After setting both SLURM and SLURMDBD should be restarted.

Finally, add the SLURM cluster to SLURMDBD:

```
sacctmgr add cluster cluster_name
```

2.1.2 Install Open OnDemand

Installation of Open OnDemand is detailed in the Open OnDemand documentation [8]. We initiated our project using Open OnDemand version 1.1. Now the Open OnDemand version is 1.3, and the installation method has simplified significantly since then. Here we describe the process with regard to 1.3, the current version.

There are some dependencies needed by Open OnDemand: Apache HTTP Server 2.4, NGINX 1.6, Phusion Passenger 4.0, Ruby 2.2, Nodejs 0.10, and Git 1.9. Apache HTTP Server is an open source, secure and efficient HTTP server for systems like Windows and Linux [9]. NGINX is an open source HTTP web server, and it is a useful tool for reverse proxy and load balancing for HTTP traffic [10]. Phusion Passenger is an open source web application server that can help handle HTTP request, manage resources and processes [11]. Ruby is an open source programming language [12] that is used in Open OnDemand.

Nodejs is a JavaScript runtime environment that brings JavaScript support to Open OnDemand [13].

To install Open OnDemand version 1.3, first download the RPM package to add Open OnDemand Repository:

sudo yum install https://yum.osc.edu/ondemand/1.3/ondemand-release-web-1.3-1.el7.noarch.rpm

Then install Open OnDemand using YUM:

sudo yum install ondemand

Then disable SELinux by editing /etc/selinux/config and setting the SELINUX value to disabled. Restart the server node.

Now open the ports 80 and 443 using firewalld.

Set a cluster configuration file in /etc/ood/config/clusters.d/, the setting should be consistent with the SLURM and SLURMDBD configuration file.

Finally, start Apache and the Open OnDemand should be usable. Note that there are additional settings for security and extra functions, which we did not install, as they are not necessary for Open OnDemand to work and provide basic functions.

2.2 Git Version Control for Open OnDemand

Our web-based Git version control app can be used as a standalone version control application. However, it is intended to be useable in combination with existing Open OnDemand apps. We modified the existing Job Composer app in Open OnDemand to include features for a Git commit action and quick access to Git branch management. The following provides details regarding methods used to develop of the app.

2.2.1 Ruby on Rails

Ruby on Rails is an open source framework for building web sites [14]. It is written in the Ruby programming language and provides an environment for a mixed usage of Ruby, JavaScript, HTML, and CSS. The part that is written in Ruby is for building the server side which contains the function that handles all incoming requests. JavaScript is used for data handling and some simple user side requests. HTML is used for building the webpage, and CSS is mostly used to control the web page style. Many well-known websites are built in Ruby, such as GitHub, Hulu, Airbnb and previous versions of Twitter.

2.2.1.1 Ruby on Rails Environments

When a Ruby on Rails application is created, three environments are created: development, test, and production [15]. The development environment is designed for rapid development. By using it, it reloads all classes upon refresh, and it reduces the need to restart the web server to see the changes take effect.

Note that caching is turned off in this mode [15]. Also, in this environment, the web page will provide a detailed description for the reason why the server failed.

The test environment is for testing only. The database is set to a one-time-data model. After the test, the data saved will all be destroyed [15]. The production environment is used for an application to run on a live server [15]. Caching will be turned on in this environment. Ruby on Rails also provides the ability to quickly switch between the three environments, which can be done using a simple command.

2.2.1.2 Model, View, and Controller

Model, view, and controller are the most important three elements in the Ruby on Rails application development process. The relationship between them is shown below in Figure 1:

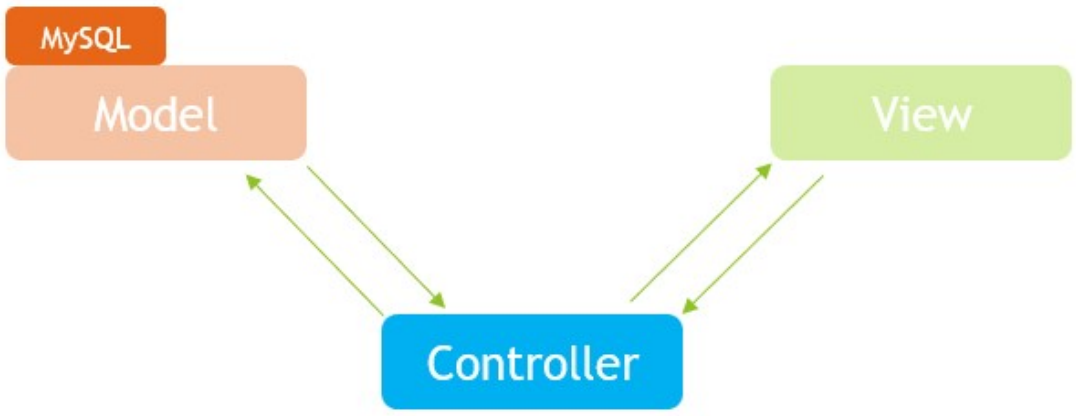

Figure 1. Model, View, and Controller architecture for web app design.

Model is a class that contains the information about the application and functions for handling the information [16]. Ruby on Rails has built in support for SQLite [17], and this is the most common way for Ruby on Rails application to save and load data. In Ruby on Rails, it is common to link it with a database table, so normally every model has a corresponding database table.

Views control the user interface in a Ruby on Rails application [16]. Views are usually written in HTML. The API includes “helpers”, including built in and custom helpers, which can be written in an easier and shorter format and can generate the corresponding full HTML code.

Controllers, as can be seen in figure 1, are the median between models and views [16]. The controllers respond to the requests that come from web browsers, get data from models, and send the data to views.

2.2.2 Function Design

2.2.2.1 Process Git Command

Open OnDemand requires that all clusters can be accessed and controlled from the Open OnDemand server node, and all job files are saved on the Open OnDemand server node, so that the web server can access Git locally. To access Git locally, there are a few ways to choose from:

1. Use a Gem file named ruby-git to access Git and can do most common Git actions. The problem with this method is that it cannot handle the Git command failure, and the web page will fail. Additionally, the error information it can provide is limited.

2. Use Ruby's SSH session Gem. The advantage of this method is that it can access Git even in a remote node. A disadvantage is that it can be slow.

3. Use the Open3 package. The Open3 Gem file provides a way to run shell commands and get the result without risking the stability of the webpage. Additionally, the result can be easily customized.

After comparing all three ways, we decided to use the Open3 package. It is a GEM package that is also used in standard Open OnDemand apps. To make the programming easier, shell scripts are built in to help process Git commands.

To store the gathered information and track the user's actions, five models are created:

1. Git session model: Git session model will save the username and current Git repository path in the database where the username is the key.
2. Git project model: This is the main model for gathering information. For one Git repository path, a Git project model will be created. One Git project model instance includes many branch model instances.

3. Branch model: One instance of a branch model represents one branch in the Git project. One instance contains the branch name and many commit model instances.
4. Commit model: One instance of commit model represents one commit in one branch. The instance contains the time of submission, author name, author email, commit ID and commit message.
5. Branches Compare model: This model uses the results of a comparison between two branches. One instance of this model contains one line of data and a string representing the color to be displayed in the visualization of the branch comparison.

2.2.2.2 Session Tracking

We use the term “session” to refer to the time period during which a user has the app open. As mentioned above, a model is built for tracking the user session. All functions will read the current session information to get the current Git repository path. This is needed on the server end when the function finished, at which time the controller will be destroyed, and all unsaved data will be lost. To solve this problem, data that is needed for future use needs to be sent to the user end and read back when needed, or saved into the database and read when needed. Here the second approach to using the database is chosen in order to reduce the data transfer amount.

For the Git Manager's initializer, the use of the model is different from that of others. When there is no repository path, the Git manager will try to read the saved session information. When there is path data coming from other web pages, the Git Manager will create or update the Git session instance value.

2.2.2.3 Path Input Function

A form with a text field and a submit button are used for path input. And the main algorithm is shown below.

Algorithm 1

1. The submit button is clicked.
2. Controller is activated and will start to check the input.
3. If the input is legal and the corresponding path exists and accessible, it will send the user to a new page with the main Git functions. If not, it will send the user back to the webpage and give an alarm.
4. When a new page is opened, the corresponding controller will start to check if the path is a Git repository. If the path does not contain a valid repository, an alarm will be sent.
5. The controller will start to collect all branch names and commits in current branch, send all information, and redirect to the frontend webpage.

2.2.2.4 Commit Function

The main algorithm is shown below:

Algorithm_2

1. The corresponding function in the controller gets the intent, and redirects to another page.
2. A new page has a form with a text field for message input and one submit button.
3. When the submit button is clicked, the corresponding function will be activated, the message will be checked. If the message is empty, the browser will be directed back with an alert, or else a shell script will be started.
4. The shell script will check if the directory is a valid Git repository. If not, it will execute the Git initialization command. Then all files in the current directory will be committed to the current branch.
5. When the shell script has finished, the webpage will be redirected to the Git manager, and the information will be refreshed. The new commit will be shown at the top of the table if committed successfully.

2.2.2.5 Delete Commit Function

The main algorithm is shown below.

Algorithm_3

1. The corresponding function in the controller will get the intent, and pass the command and start shell script.
2. The shell script will run the delete current commit function.
3. The browser will be directed back to the Git manager, and information will be refreshed.

2.2.2.6 Checkout Commit Function

The checkout commit view is a form that contains a text field and the submit button. The main algorithm is shown below.

Algorithm 4

1. The corresponding function in the controller will get the intent, and read the input commit ID from a text field.
2. Pass the command and start shell script.
3. The shell script will run the checkout commit command. If successful, the old commit will be checked out to be the current commit, and the Git repository will enter detached mode.
4. The browser will be directed back to the Git manager, and information will be refreshed.

2.2.2.7 Change Branch Function

The main algorithm is shown below.

Algorithm 5

1. The corresponding function in the controller will get the intent, and read the current dropdown menu selection value.
2. Pass the command and start shell script.
3. The shell script will run the checkout branch command.
4. The browser will be directed back to the Git manager, and information will be refreshed.

Routing between apps is not as simple as routing in one app. To achieve this, a rule for URL generation is needed. Fortunately, Open OnDemand provides a Gem file for this task. Most of the Open OnDemand apps have a built in rule to work with, except the Job Composer app and Active Jobs app. As we need to open the Job Composer app from the Git Manager and open the Git Manager from the Job Composer app, two new rules need to be added to the Gem file.

2.2.2.8 Links to Other App

Two new classes were created for the Job Composer app and Git Version Control app. The classes are mainly to combine the base URL with an additional value to be passed. The configuration file needs to be changed, and the base URLs for Job Composer app and Git Version Control app need to be added. A base URL is a relative path that the Open OnDemand can use to find the app.

2.2.2.9 Branches Compare and Merge Function

These two functions are built with the same form with two drop-down menus and two submit buttons. The main algorithm is shown below.

Algorithm 6

1. The corresponding function in the controller will get the intent.
2. The function will read the two drop-down menus' values and check the submit button value to determine the appropriate action to be made.
3. If the action should be merging the branches, an auto-merge will be started.
4. If the action should be comparing the branches, the function will get the output from a shell script which compares two branches.
5. Line-by-line results of the branch compare will be displayed using a color coding scheme, as is done in some other version control GUIs.
6. Color information for every line will be calculated. If a data line starts with '+', which represents adding text to a file in the repository, the color will be set to green. If it starts with '-', which represents deleting text from a file in a repository, the color will be set to pink. Otherwise, the color will be set to white.
7. The output lines and color information will be sent to the frontend, where a new page will be created. The compare results will be shown

in the new page, line by line. The background color will be set to the corresponding value generated by the function in the controller.

2.2.2.10 Create New Branch Function

The main algorithm is shown below.

Algorithm_7

1. The corresponding function in the controller will get the intent and redirect to another page.
2. A new page has a form with a text field for message input and one submit button.
3. When the submit button is clicked, the corresponding function will be activated, and the new branch name will be checked. If the name field is empty, the user will be sent back with an alert, or else a shell script will be started.
4. When the shell script is finished, the web browser will be redirected to the branch manager, and the information will be refreshed.

2.2.2.11 The Job Composer App Commit Function

The main algorithm of the Commit function is similar to the algorithm 2.2.2.4. The differences are:

1. There needs to be a selected element in the workflow table before this button can work.
2. The commit button will read the currently selected job's path in the database.

2.2.4 Apps Installation

To use the full functionality of the Git Version Control app, both the Git Version Control app and the modified Job Composer app need to be installed. The installation instructions are similar to those of the official app installation instructions [23], with a few more steps to replace the modified GEM file.

First, delete the old installation of the Job Composer app.

In the Git Version Control app directory, run:

```
RAILS_ENV=production scl enable git19 rh-ruby22 nodejs010 -- bin/setup
sudo mkdir -p /var/www/ood/apps/sys/gits
sudo cp -r . /var/www/ood/apps/sys/gits
```

In the modified Job Composer app directory, run:

```
RAILS_ENV=production scl enable git19 rh-ruby22 nodejs010 -- bin/setup
sudo mkdir -p /var/www/ood/apps/sys/myjobs
sudo cp -r . /var/www/ood/apps/sys/myjobs
```

Then copy the modified OOD-appkit-1.03 directory, and replace the directory with the same name in both apps.

In version 1.3 of Open OnDemand, the GEM file may not be installed. In this case, the applications are still usable, but it will be more difficult to replace the modified GEM file. In this case, run the following command in the installed directory, and replace the GEM file:

```
sudo RAILS_ENV=production scl enable git19 rh-ruby22 nodejs010 -- bin/setup
```

CHAPTER 3: RESULTS

3.1 Deployment of Open OnDemand

Our first objective was to evaluate the process of deploying Open OnDemand. For versions 1.1 and 1.2, a significant number of separate installations were required for third-party software, including Apache, NGINX, Phusion Passenger, Ruby, Node.js, and Git. However, version 1.3 is installed through an RPM package. The documentation also describes numerous configuration tasks, following the software installation, regarding security, launching the web server, and configuration files for use with job management software [8].

We make several observations regarding the process and considerations for other users. The results show that the instructions are thorough and accurate. The documentation also includes detailed specifications for CentOS, which is important for those who require an open source OS. Although the Open OnDemand installation is highly streamlined, the installation of job management software requires separate expertise. Still, the configuration file examples for supported job management software are well documented.

3.2 Git Version Control app

Our Git Version Control app exchanges information with the underlying Git version control software. Our app requires Open OnDemand in order to operate,

but users are not restricted to using our app exclusively because Git itself is functional without Open OnDemand. Our intent is to make common tasks convenient by having Git controls that exist side-by-side with standard Open OnDemand controls.

When using the Git Version Control app in stand-alone mode, the user can use the link in the Open OnDemand dashboard. The index page of the Git Version Control app will ask the user to input a legal path for the Git repository before opening the Git Manager (see Figure 2). If the path is empty or incorrect, the user will be asked to re-enter a path (see Figure 3).

Figure 2. Path Input View

Figure 3. Alert for Empty Path or Incorrect Path

Figure 4 shows an example of the Git Manager view which is intended to provide controls typically found in other version control GUIs. This view includes a reverse chronological table showing each commit with the date, unique ID,

author, email address, and associated message. Above the table are various buttons for Git and Open OnDemand operations. The Commit button will redirect the users to a new page for a text message before performing a commit action. The delete button will delete the last commit. The Shell button, Edit button, and Jobs button will open other Open OnDemand apps. The Shell button will open the web-based shell app using the current Git repository path. The Edit button will open the File Explorer app in current Git repository path. The Jobs button will open the modified Job Composer app. The Branch Manager button will open our branch manager view. The Check button, located next to a text area, allows the user to enter a partial ID of any commit in order to roll back files to that state. Note that this will cause the current files to enter a state that is called a “detached head”, where new modifications are not associated with a specific branch. The Change Branch button and associated drop-down menu allow the user to switch between different branches. Note that, in a detached head state, if the user changes back to the head of a specific branch, any uncommitted file changes will be lost. However, if there are any unsaved changes in the current branch, an automatic commit will be made. As shown in Figure 4 and Figure 5, the Git Manager view uses the same style as the original Job Composer app that is standard in Open OnDemand.

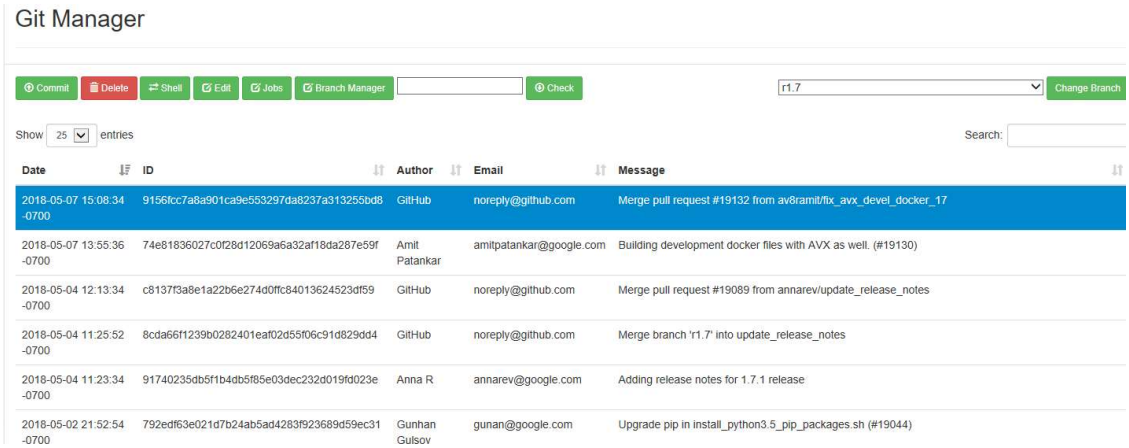

Figure 4. Git Manager view

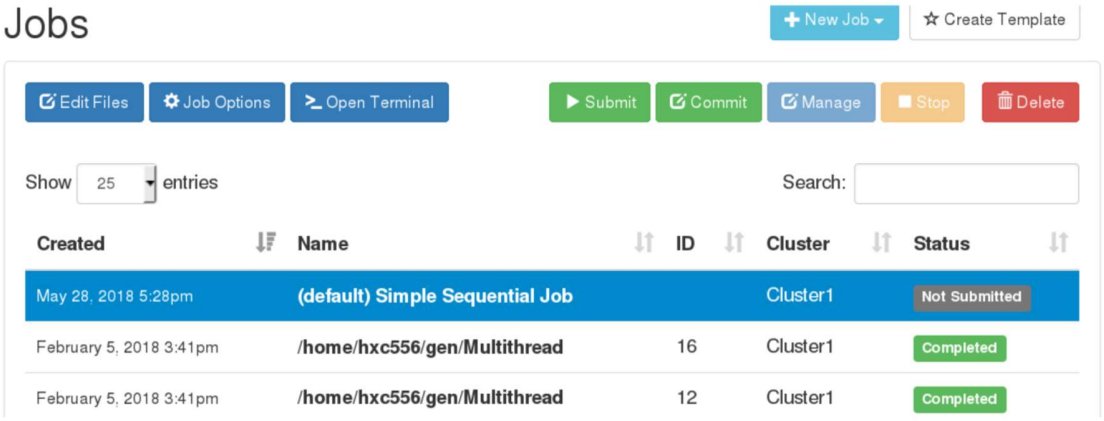

Figure 5. Modified Job Composer app

The modified Job Composer app is almost identical to the app that is installed with the standard Open OnDemand, and it retains the same features. In our new app, there are two new, significant features: a Commit button and a Manage button. The Commit button redirects to a new page for a text message before committing the files for the currently selected job. We consider this to be a very convenient feature because the execution of a batch job is often a moment

at which a user has either made a significant revision or has obtained significant results. The Manage button redirects to the Git Manager app in selected job path.

The Branch Manager view provides features that are common for users who work with multiple branches. Figure 6 shows an example of the Branch Manager view. On the left side, two drop-down menus allow the user to select two branches by name. The Compare button produces a new page containing a comparison between the branches, as described below. The Merge button attempts to merge modifications from the second branch into the first branch, assuming there are no conflicts. In the case of a merge conflict, the Git error message is displayed, and no other action is taken. Note that merging branches is an advanced version control technique that may require a standard Git interface. Lastly, Create New Branch creates a new branch originating from the current branch.

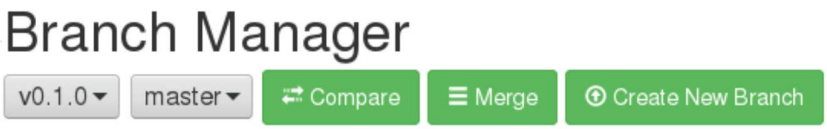

Figure 6. Branch Manager view main controls

As stated above, the Branch Manager view includes a Compare button that uses the branch comparison feature in Git. Figure 7 shows an example of a view that might appear when comparing two branches. In this example comparison, differences in every file are displayed, along with additional information that indicates where a particular branch either added or deleted text

within the given file. The difference is color coded according to their category, such as green for text additions and pink for text deletions.

Figure 7. Branch Comparison view

Finally, we describe how the branch comparison feature is an advanced tool that can be useful to HPC researchers for different reasons. At a basic level, version control is valuable without the use of separate branches, simply because it allows a researcher to efficiently manage changes to their software. For an independent researcher, however, the use of branches offers the ability to maintain multiple versions of a project that involve a common code base but may differ with regard to particular experiments or analyses. In this context, a researcher can utilize our branch comparison feature to organize and identify important differences between various experiments, analyses, etc. The use of branches may also be of interest to research teams where multiple researchers need to work concurrently. The branch comparison feature allows a development team to manage coordinated modifications. In this context, it may be desirable to

merge branches. It is generally desirable to first perform a branch comparison in order to identify and resolve conflicts prior to the attempted merge.

Chapter 4: DISCUSSION

Having been personally trained on the use of HPC systems through traditional methods, and having subsequently trained others for the same methods, we feel Open OnDemand represents an important tool in opening HPC to a wider community. In order for Open OnDemand and GUI development, in general, to be successful, open source tools must be tested and extended by the community. The present study has contributed to both tasks. It is noteworthy that the researchers in this study did not have experience as system administrators, particularly with regard to a job management system on an HPC cluster. Our lack of experience highlights how Open OnDemand has been developed to include a streamlined installation process and excellent documentation.

Our custom version control app also served as a means of testing the Open OnDemand framework with regard to extensibility. Not surprisingly, the development of the app depended significantly on our own previous experience with web app development, including experience with Ruby on Rails. However, many excellent resources exist for researchers to learn web development skills. Additionally, Open OnDemand includes tools, examples, and documentation to assist developers. Also, the developer mode is well made and easy to use. Although most of our development work did not utilize it, the developer mode's My Sandbox feature provides a real environment for testing with the installed standard apps, which is very important for testing the redirection between apps. For developers who want to develop an Open OnDemand application, the My

Sandbox function can be more than useful because directly testing software in a real environment is hard and may need some extra development, especially when developing a feature that works with more than one application.

In choosing version control to be the focus of our app development exercise, we hoped to address a lack of awareness among researchers regarding this important software development tool. As HPC methods evolve to be more accessible, opportunities exist to promote good practices such as version control. In conclusion, our tests and development show that the Open OnDemand system is both administrator friendly and developer friendly. By adding a Git application to Open OnDemand, we provided a useful tool that can help researchers in the future.

APPENDIX 1: List of Important Files in Git APP

gits/db/schema.rb: Contains all data table information.

gits/db/test.sqlite3: Database for test

gits/db/development.sqlite3: Database for development

gits/data/production.sqlite3: Database for production

gits/db/migrate/20180527033617_create_git_sessions.rb: Save user ID and Git repository path to the database.

gits/git_scripts/message_commit.sh: Commit files with a commit message, if the directory is not a Git directory, it will be initialized automatically.

gits/git_scripts/safe_checkout.sh: Checkout to a new branch, if current branch got files that need to be committed, an automatic commit will be made.

gits/git_scripts/get_diff.sh: Run Git Command to get differences of two branches.

gits/git_scripts/get_commits.sh: Get the commit information in current branch.

gits/git_scripts/commit_delete.sh: Delete current commit.

gits/git_scripts/get_head.sh: Return current commit ID.

gits/git_scripts/get_merge.sh: Run the auto-merge command.

gits/manifest.yml: Define the application name, icon and description here.

gits/Gemfile: All GEM file that was used should be written here.

gits/bin/bundle, gits/bin/rails, gits/bin/setup, gits/bin/rake, gits/bin/setup-production: Files used for setup and installation.

gits/config.ru: Used by a rack-based server to start the application.

gits/config/database.yml: Set database for three environments: development, test, production.

gits/config/application.rb: Settings for the application.

gits/config/secrets.yml: Secret key for database access.

gits/config/locales/en.yml

gits/config/routes.rb: Set the routes inside the application. Used for finding the routes to controllers.

gits/app/assets/javascripts/jquery.cookie.js: Used for cookie usage.

gits/app/assets/javascripts/datatables.js.coffee: Used to set the default value for data table in Git Manager.

gits/app/assets/javascripts/application.js: The main JavaScript file that responds to views.

gits/app/models/filesystem.rb: The model that used to create other applications URL, so the Git app can redirect to them.

gits/app/models/commit.rb: The model that record commit information, include commit time, commit id, commit author, author email address, and commit messages.

gits/app/models/branch.rb: The model that gather all commits in a branch, and record the branch name.

gits/app/models/git_project.rb: The model which gather all branch information in a Git repository.

gits/app/models/diff.rb: Record two branches' compare the result with two string value, content, and color.

gits/app/models/git_session.rb: Record the username and current Git repository path for future usage.

gits/app/views/gits/branch_compare.html.erb: The page that shows the two branches' compare result in color.

gits/app/views/gits/commit.html.erb: The page that asks for a commit message.

gits/app/views/gits/branch_manager.erb: The page that provides basic functions for branch operation.

gits/app/views/gits/git_manager.html.erb: The page that provides commits information and basic Git operation.

gits/app/views/gits/new_branch.html.erb: The page that asks for the new branch name.

gits/app/views/gits/git_main.html.erb: The page that asks the user to input a valid path for Git operation.

gits/app/views/layouts/application.html.erb: The file that set the basic layout for other pages to use.

gits/app/controllers/gits_controller.rb: The file that all functions that respond to request from Git related pages in a web browser.

gits/app/controllers/application_controller.rb: The controller responds to application.html.erb

APPENDIX 2: List of Partial Codes in Git APP

application.js

Used the same way as the Job Composer app handle the data. The selected content will be set to active and show as selected.

```
if ($('#git-list-table').length) {
    git_table = $('#git-list-table').DataTable();
    $('#git-list-table tbody').on('click', 'tr', function () {
        if ($(this).hasClass('active')) {
            $(this).removeClass('active');
        }
        else {
            git_table.$('tr.active').removeClass('active');
            $(this).addClass('active');
        }
    });
}
if (git_table) {
    if (git_table.$('#' + selected_id).length > 0) {
        git_table.$('#' + selected_id).click();
    } else {
        git_table.$('tr:first').click();
    }
    console.log("success")
};
```

git_controller.rb

```
# The main class of the git controller
class GitsController < ApplicationController
```

```
# The activity for the git_main page, nothing need to be done here.
  def git_main
  end
#The activity respond to Git Manager's page.
  def git_manager
    cur_user,s = Open3.capture2 "whoami"
    path_to_git=params["git_path"]
    if path_to_git.nil?
      if GitSession.exists?(:user =>cur_user)
        @git_session = GitSession.find_by(:user =>cur_user)
        path_to_git = @git_session.path_to_git
      else
        flash[:alert] = "Please Input Path"
        redirect_to root_path
        return
      end
    else
      if GitSession.exists?(:user =>cur_user)
        @git_session = GitSession.find_by(:user => cur_user)
      else
        @git_session = GitSession.new
        @git_session.user = cur_user
      end
      @git_session.path_to_git = path_to_git
      @git_session.save
    end

    if path_to_git[-1]!='/'
      path_to_git=path_to_git+'/'
    end
```

```
    if !File.exist? (path_to_git + '.git')
      flash.now[:alert] = 'This is not a git repository, please check the path or make
an initial commit before using any other function'
      @git_project_case = GitProject.new
      @branch_case = Branch.new
    else
      @git_project_case = GitProject.get_current_porject(path_to_git)
      @git_project_case.path_to_git=path_to_git
      @branch_case = Branch.get_branch_info(path_to_git)
    end
  end
  def show
  end
  def check
    path_to_git=params["git_repo_path"]
    if path_to_git.nil? || (!File.exist? path_to_git)
      flash[:alert] = "Path Does Not Exist"
      redirect_to root_path
    elsif !File.directory? path_to_git
      flash[:alert] = "Path is not a directory"
      redirect_to root_path
    else
      redirect_to git_manager_path(:params => {:git_path => path_to_git})
    end
  end
  def checkout_branch
    cur_user,s = Open3.capture2 "whoami"
    @git_session = GitSession.find_by(:user =>cur_user)
    Open3.capture2 "./git_scripts/safe_checkout.sh
#{Shellwords.escape(@git_session.path_to_git)}
#{Shellwords.escape(params['branch_id'])}"
```

```
    redirect_to git_manager_path
  end
  def commit
  end
  def commit_message
    cur_user,s = Open3.capture2 "whoami"
    @git_session = GitSession.find_by(:user =>cur_user)
    message_for_commit = params['git_commit_message']
    if message_for_commit.size == 0
      flash[:alert] = "Please enter a message"
      redirect_to git_commit_path
    else
      o,s = Open3.capture2 "./git_scripts/message_commit.sh
#{Shellwords.escape(@git_session.path_to_git)}
'#{Shellwords.escape(message_for_commit)}'"
      redirect_to git_manager_path
    end
  end
  def file_system
    cur_user,s = Open3.capture2 "whoami"
    @git_session = GitSession.find_by(:user =>cur_user)
    redirect_to Filesystem.new.fs(@git_session.path_to_git)
  end
  def job_composer
    redirect_to Filesystem.new.myjobs
  end
  def delete
    cur_user,s = Open3.capture2 "whoami"
    @git_session = GitSession.find_by(:user =>cur_user)
    Open3.capture2 "./git_scripts/commit_delete.sh
#{Shellwords.escape(@git_session.path_to_git)}"
```

```
    redirect_to git_manager_path
  end
  def to_branch_manager
    redirect_to git_branch_manager_path
  end
  def branch_manager
    cur_user,s = Open3.capture2 "whoami"
    @git_session = GitSession.find_by(:user =>cur_user)
    path_to_git = @git_session.path_to_git
    if path_to_git[-1]!='/'
      path_to_git=path_to_git+'/'
    end
    @git_project_case = GitProject.get_current_porject(path_to_git)
    @branch_case = Branch.get_branch_info(path_to_git)
  end
  def branch_control
    branch_1 = params["branch_id_1"]
    branch_2 = params["branch_id_2"]
    if params["submit"]
      cur_user,s = Open3.capture2 "whoami"
      @git_session = GitSession.find_by(:user =>cur_user)
      o,s = Open3.capture2 "./git_scripts/get_merge.sh
#{Shellwords.escape(@git_session.path_to_git)}
#{Shellwords.escape(branch_1)} #{Shellwords.escape(branch_2)}"
      if s[-6,-1]!=”exit 0”
          flash[:alert] = “Auto Merge Failed”
      redirect_to git_branch_manager_path
    else
      redirect_to git_branch_compare_path(:params => {:branch_id_1 =>
branch_1, :branch_id_2 => branch_2})
    end
```

```
end
def to_new_branch
  redirect_to git_new_branch_path
end
def new_branch
end
def new_branch_creator
  cur_user,s = Open3.capture2 "whoami"
  @git_session = GitSession.find_by(:user =>cur_user)
  g = Git.open(@git_session.path_to_git, :log => Logger.new(STDOUT))
  new_branch_name = params["git_branch_name"]
  test = g.branch(new_branch_name).checkout
  redirect_to git_branch_manager_path
end
def to_git_manager
  redirect_to git_manager_path
end
def to_branch_compare
  redirect_to git_branch_compare_path
end
def branch_compare
  @branch_id_1 = params[:branch_id_1]
  @branch_id_2 = params[:branch_id_2]
  if @branch_id_1.nil? || @branch_id_2.nil?
    return
  end
  cur_user,s = Open3.capture2 "whoami"
  @git_session = GitSession.find_by(:user =>cur_user)
```

```
    o,s = Open3.capture2 "./git_scripts/get_diff.sh
#{Shellwords.escape(@git_session.path_to_git)}
#{Shellwords.escape(@branch_id_1)} #{Shellwords.escape(@branch_id_2)}"
    diff_data = o.split("\n")
    @diffs = []
    for diff in diff_data do
      diff_case = Diff.new
      diff_case.content = diff
      if diff[0].eql? "+"
        diff_case.color= "greenyellow"
      elsif diff[0].eql? "-"
        diff_case.color="hotpink"
      else
        diff_case.color="white"
      end
      @diffs << diff_case
    end
  end
  def shell
    cur_user,s = Open3.capture2 "whoami"
    @git_session = GitSession.find_by(:user =>cur_user)
    redirect_to Filesystem.new.shell(@git_session.path_to_git)
  end
end
```

commit.rb

```
class Commit < ActiveRecord::Base
  belongs_to :branch
attr_accessor :commit_id, :commit_author_name, :commit_author_email, :commi
t_time, :commit_message
```

```
  def self.new_commit(new_id, new_author_name, new_author_email, new_time,
new_message)
    commit = Commit.new
    commit.commit_id = new_id
    commit.commit_author_name = new_author_name
    commit.commit_author_email = new_author_email
    commit.commit_time = new_time
    commit.commit_message = new_message
    commit
  end
end
```

branch.rb

```
class Branch < ActiveRecord::Base
  belongs_to :git_project
  has_many :commits, dependent: :destroy
  attr_accessor :branch_name
  def self.new_branch(new_name)
    @branch_case = Branch.new
    @branch_case.branch_name=new_name
    @branch_case
  end
  def self.get_branch_info(path_to_git)
    branch_case = Branch.new
    g = Git.open(path_to_git, :log => Logger.new(STDOUT))
    new_name = g.describe('HEAD', {:all => true, :tags => true})
    if new_name[0..5].eql?'heads/'
      branch_case.branch_name = new_name[6..-1]
    else
```

```
      branch_case.branch_name = new_name
    end
    o, s = Open3.capture2 "./git_scripts/get_commits.sh
#{Shellwords.escape(path_to_git)} #{Shellwords.escape("-100")}"
    commits = o.split('|~|!!')
    last_idx=-2
    if commits.size<3
      last_idx=-1
    end
    for commit in  commits[0..last_idx] do
      commit_info = commit.split('||~')
      branch_case.commits <<
Commit.new_commit(commit_info[0],commit_info[7],commit_info[8],commit_info[
7],commit_info[8])
    end
    branch_case
  end
end
```

git_project.rb

```
class GitProject < ActiveRecord::Base
  has_many :branches, dependent: :destroy
  attr_accessor :path_to_git
  def self.get_current_porject(path_to_git)
    git_project = GitProject.new
    if File.exist? (path_to_git + '.git')
      g = Git.open(path_to_git, :log => Logger.new(STDOUT))
      git_project.path_to_git = path_to_git
      for branch_name in g.branches do
        name = branch_name.to_s
```

```
      git_project.branches << Branch.new_branch(branch_name)
    end
  end
  git_project
 end
end
```

filesystem.rb

This file is mostly the same as the one in Job Composer App, the myjobs function is the only function added. To use this function the Ood-Appkit GEM file need to be modified.

```
 def myjobs
  OodAppkit.myjobs.url.to_s
 end
```

git_session.rb

Nothing need to be write in the class, all data will be directly read and saved.

```
class GitSession < ActiveRecord::Base
end
```

gits/db/migrate/20180527033617_create_git_sessions.rb, corresponding to git_session.rb

```
class CreateGitSessions < ActiveRecord::Migration
 def change
  create_table :git_sessions do |t|
   t.string :user
   t.string :path_to_git
   t.timestamps null: false
  end
 end
```

```
end
```

diff.rb

```
class Diff < ActiveRecord::Base
  attr_accessor :content, :color
end
```

routes.rb

```
Rails.application.routes.draw do
  resources :gits do
    collection do
      post :check
      get :checkout_branch
      get :commit_message
      get :file_system
      get :job_composer
      get :delete
      get :to_branch_manager
      get :to_new_branch
      get :new_branch_creator
      get :to_git_manager
      get :branch_control
      get :to_branch_compare
      get :shell
    end
  end
  root "gits#git_main"
  get "git_manager" => "gits#git_manager"
```

```
  get "git_commit" => "gits#commit"
  get "git_branch_manager" => "gits#branch_manager"
  get "git_new_branch" => "gits#new_branch"
  get "git_branch_compare" => "gits#branch_compare"
```

git_main.html.erb

Some other files are similar with this one and will be skipped.

```
<div class="page-header" xmlns="http://www.w3.org/1999/html">
  <h2>Git</h2>
</div>
<div class="panel panel-default">
  <div class="panel-body">
    <div class="field">
      <h4>Path</h4>
      <%= form_tag(:controller => "gits",:action => "check", method: "get") do %>
        <input id="git_repo_path" name="git_repo_path" type="text" rows="1"
style="width: 100%"/>
        <%= button_tag(:type => 'submit', :class => "btn btn-success") do %>
          <span class="glyphicon glyphicon-play" aria-hidden="true"></span> <span
class="hidden-xs hidden-md"> Check </span>
        <% end %>
      <% end %>
    </div>
  </div>
</div>
```

Git_manager.html.erb

The page used the similar main struct with Job Composer App, but all items, i.e. buttons, drop-down menu and so on, are newly made.

```
<div class="page-header" xmlns="http://www.w3.org/1999/html">
  <h2>Git Manager</h2>
</div>
<div class="row">
  <div class="col">
    <div id="git-list-view" class="panel panel-default">
      <div class="panel-body">
        <a class="btn btn-success btn-sm" id="commit_all_button" data-
toggle="tooltip" title="Commit Your Build" role="button" data-method="get" href =
"git_commit">
          <span class="glyphicon glyphicon-upload" aria-
hidden="true"></span><span class="hidden-xs"> Commit </span>
        </a>
        <a class="btn btn-danger btn-sm btn-primary" id="delete_commit_button"
data-toggle="tooltip" title="Delete Last Commit" role="button" data-method="get"
data-confirm="Are You Sure?" href ="gits/delete" >
          <span class="glyphicon glyphicon-trash" aria-hidden="true"></span><span
class="hidden-xs"> Delete </span>
        </a>
        <a class="btn btn-success btn-sm" id="compare_button" data-
toggle="tooltip" title="Compare Two Commits" role="button" data-method="get"
href = "gits/shell" target="_blank">
          <span class="glyphicon glyphicon-transfer" aria-
hidden="true"></span><span class="hidden-xs hidden-md"> Shell </span>
        </a>
        <a class="btn btn-success btn-sm" id="file_system_button" data-
toggle="tooltip" title="Edit Files In File Browser" role="button" data-method="get"
href="gits/file_system" target="_blank">
          <span class="glyphicon glyphicon-edit" aria-hidden="true"></span><span
class="hidden-xs hidden-md"> Edit </span>
        </a>
        <a class="btn btn-success btn-sm" id="job_composer_button" data-
toggle="tooltip" title="Manage Batched Jobs" role="button" data-method="get"
href="gits/job_composer" target="_blank">
```

```
        <span class="glyphicon glyphicon-edit" aria-hidden="true"></span><span
class="hidden-xs hidden-md"> Jobs </span>

      </a>

      <a class="btn btn-success btn-sm" id="manage_branch_button" data-
toggle="tooltip" title="Manage The Branch" role="button" data-method="get"
href="gits/to_branch_manager">

        <span class="glyphicon glyphicon-edit" aria-hidden="true"></span><span
class="hidden-xs hidden-md"> Branch Manager </span>

      </a>

      <form accept-charset="UTF-8" action="gits/checkout_branch" method="get"
style="display: inline">

        <%= text_field_tag(:branch_id)%>

        <%= button_tag(:type => 'submit', :class => "btn btn-sm btn-success", :title
=> "Checkout A Commit") do %>

          <span class="glyphicon glyphicon-download" aria-hidden="true"></span>
<span class="hidden-xs hidden-md"> Check </span>

        <% end %>

      </form>

      <form accept-charset="UTF-8" action="gits/checkout_branch" method="get"
style="display: inline" class="pull-right">

        <input name="utf8" type="hidden" value="&#x2713;" />

        <%= select_tag 'branch_id',
options_for_select(@git_project_case.branches.collect {|branch|
branch.branch_name},@branch_case.branch_name)%>

        <input name="commit" type="submit" value="Change Branch" class="btn
btn-success btn-sm"/>

      </form>

    </div>

    <div class="panel-body">

      <table id="git-list-table" class="table data-table" style="width:100%" >

        <thead>

        <tr>

          <th>Date</th>

          <th>ID</th>
```

```
          <th>Author</th>
          <th>Email</th>
          <th>Message</th>
        </tr>
        </thead>
        <tbody>
        <% @branch_case.commits.each do |commit| %>
          <tr class="git-row" id="<%= commit.commit_id %>">
            <td><%= commit.commit_time %></td>
            <td><%= commit.commit_id %></td>
            <td><%= commit.commit_author_name %></td>
            <td><%= commit.commit_author_email %></td>
            <td><%= commit.commit_message %></td>
          </tr>
        <% end %>
        </tbody>
      </table>
    </div>
  </div>
 </div>
</div>
```

branch_manager.erb

```
<div class="page-header" xmlns="http://www.w3.org/1999/html">
  <h2>Branch Manager</h2>
</div>
<div class="row">
  <div class="col">
    <div id="git-list-view" class="panel panel-default">
```

```
    <div class="panel-body">
      <form accept-charset="UTF-8" action="gits/branch_control" method="get" style="display: inline" target="_blank">
        <input name="utf8" type="hidden" value="&#x2713;" />
        <%= select_tag 'branch_id_1', options_for_select(@git_project_case.branches.collect {|branch| branch.branch_name},@branch_case.branch_name)%>
        <%= select_tag 'branch_id_2', options_for_select(@git_project_case.branches.collect {|branch| branch.branch_name},"master")%>
        <%= button_tag(:type => 'submit', :class => "btn btn-sm btn-success", :title => "Compare Two Branch") do %>
          <span class="glyphicon glyphicon-transfer" aria-hidden="true"></span> <span class="hidden-xs hidden-md"> Compare </span>
        <% end %>
        <%= button_tag(:name => "submit", :type => 'submit', :class => "btn btn-sm btn-success", :title => "Merge Two Branch", :value => "Merge") do %>
          <span class="glyphicon glyphicon-menu-hamburger" aria-hidden="true"></span> <span class="hidden-xs hidden-md"> Merge </span>
        <% end %>
      </form>
      <a class="btn btn-success btn-sm" id="new_branch_button" data-toggle="tooltip" title="Create New Branch" role="button" data-method="get" href = "gits/to_new_branch">
        <span class="glyphicon glyphicon-upload" aria-hidden="true"></span><span class="hidden-xs"> Create New Branch </span>
      </a>
      <a class="btn btn-success btn-sm pull-right" id="commit_all_button" data-toggle="tooltip" title="Back To Git Manager" role="button" data-method="get" href = "gits/to_git_manager">
        <span class="glyphicon glyphicon-backward" aria-hidden="true"></span><span class="hidden-xs"> Back </span>
      </a>
    </div>
  </div>
```

```
  </div>
</div>
```

Branch_compare.html.erb

```
<div class="page-header" xmlns="http://www.w3.org/1999/html">
  <h2>Version Compare</h2>
</div>
<div class="row center-block col-md-12">
  <div class="col">
    <div id="git-list-view" class="panel panel-default">
      <div class="panel-body">
        <H3><%= @branch_id_1 %> VS <%= @branch_id_2 %> </H3>
        <% @diffs.each do |diff| %>
          <div style="background-color: <%=diff.color %>"><%=
diff.content %></div>
        <% end %>
      </div>
    </div>
  </div>
</div>
```

message_commit.sh

```
cd $1
[ -d ".git"]  || (git init && git add .)
git commit -a -m "$2"
```

safe_checkout.sh

```
cd $1
git commit -a -m "Auto Commit Before Change Branch"
git checkout $2
```

get_dff.sh

```
cd $1
git diff $2 $3
```

get_commits.sh

```
cd $1
git log --pretty='%H||~%cn||~%ce||~%ci||~%s|~|!!' $2
```

commit_delete.sh

```
cd $1
git reset --hard HEAD^
```

get_head.sh

```
cd $1
git rev-parse --abbrev-ref HEAD
```

get_merge.sh

```
cd $1
git merge $2 $3
```

Reference

[1] Globus Docs: https://docs.globus.org/

[2] XSEDE Home: https://www.xsede.org/

[3] What is the XSEDE User Portal: https://kb.iu.edu/d/bazu

[4] Diwan, P. D. (2013). A Software Product Line Engineering Approach to Building A Modeling and Simulation as a Service (M&SaaS) Application Store (Doctoral dissertation, The Ohio State University).

[5] About XSEDE: http://www.ncsa.illinois.edu/enabling/xsede

[6] White paper FUJITSU Software HPC Gateway: https://sp.ts.fujitsu.com/dmsp/Publications/public/wp-HPC-Gateway-ce-en.pdf

[7] Open OnDemand GitHub: https://github.com/OSC/Open-OnDemand

[8] Open OnDemand Documentation: https://osc.github.io/ood-documentation/master/.

[9] Apache HTTP Server Project: https://httpd.apache.org/

[10] Welcome to the NGINX and NGINX Plus documentation: https://docs.nginx.com/nginx/

[11] Quickstart: Ruby + Phusion Passenger: https://www.phusionpassenger.com/library/walkthroughs/start/ruby.html

[12] Ruby: https://www.ruby-lang.org/en/

[13] Nodejs: https://nodejs.org/en/

[14] Ruby on Rails: http://guides.rubyonrails.org/getting_started.html

[15] Use of Ruby on Rails environments: http://teotti.com/use-of-rails-environments/

[16] The MVC architecture: http://guides.rubyonrails.org/v2.3/getting_started.html#configuring-a-database

[17] Ruby on Rails Configuring A Database: http://guides.rubyonrails.org/v2.3/getting_started.html#configuring-a-database

[18] Git About Version Control: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[19] MUNGE: https://dun.github.io/munge/

[20] SLURM Download Page: https://www.schedmd.com/downloads.php.

[21] SLURM Version 17.11 Configuration Tool - Easy Version: https://slurm.schedmd.com/configurator.easy.html

[22] SLURM Accounting and Resource Limits https://slurm.schedmd.com/accounting.html

[23] Open OnDemand Job Composer GitHub https://github.com/OSC/ood-myjobs