Evo Developer Bootcamp

Lab Guide

Juniper Engineering Training

Revision History:

Revision edb-017

April 24, 2024

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Comments** | **Modified by** |
| 1 | 10/18/2017 | Created the first draft | Shaughnessy (patricks) |
| 2 | 7/17/2018 | Validated/updated till end of Chapter 4 | Ranjith |
| 3 | 8/22/2018 | Validated/updated all chapters | Ranjith & Namitha |
| 4 | 4/17/2020 | First version of Evo Developer Bootcamp | McCormack |
| 5 | 6/12/2020 | Added BaaS, verified and updated exercises, reformatted chapters | McCormack |
| 6 | 7/27/2021 | Added Fixed Info section to GNATS. Added Lab Devices chapter. Added cset workaround (Ex. 2, step 2) – in Code Review chapter. | McCormack |
| 7 | 10/26/2021 | Removed Junos scope creation exercise from PRWF chapter. | McCormack |
| 8 | 3/24/2022 | Updated PRWF chapter. Added screenshots and instructions in support of GUI change. | McCormack |
| 9 | 3/31/2022 | Removed instructions on committing via Gerrit. Moved PCT section to the final chapter. | McCormack |
| 10 | 12/9/2022 | Updated Classify settings in PRWF chapter to reflect tool changes. In Sandbox chapter, removed “explore your sandbox” exercises and placed them in Build chapter. Moved Build chapter up one spot to immediately follow the Sandbox chapter. | McCormack |
| 11 | 3/30/2022 | Removed psg-evo-build alias.  Added exercise at end of guide showing how to close out an Evo PR. | McCormack |
| 12 | 3/30/2022 | Minor fixes – updated code coverage build command. | McCormack |
| 13 | 6/27/2023 | Replaced Lab 2 (“Set Up”) with text from Junos Bootcamp. Updated Lab 2 to work with Git etc. | McCormack |
| 14 | 8/18/2023 | Added Lab 9 “Lab 9: Test Code Using Static Analysis (Coverity). | McCormack |
| 15 | 9/5/2023 | Updated Sandbox chapter by replacing the target of the sparse sandbox exercise. Pio replaces radix2. | McCormack |
| 16 | 9/15/2023 | Added vPTX Lab. | McCormack |
| 17 | 4/24/2024 | Added Code Coverage lab 10. | McCormack |

Copyright © 2024, Juniper Networks— All rights reserved.

Revision date: April 24, 2024 (Revision edb-017)

The contents of this document may not be copied or duplicated in any form, in whole or in part, without the prior written permission of Juniper Networks.

The information in this document is subject to change without notice. Juniper Networks shall not be liable for any damages resulting from technical errors or omissions which may be present in this document, or from use of this document.

This document is an unpublished work protected by the United States copyright laws and is proprietary to Juniper Networks Disclosure, copying, reproduction, merger, translation, modification, enhancement, or use of this document by anyone other than authorized employees, authorized users, or licensees of Juniper Networks without the prior written consent of Juniper Networks is prohibited.

Juniper Networks and the Juniper Networks logo are trademarks or registered trademarks of Juniper Networks.

All other trademarked names used herein are the properties of their respective owners and are used for identification purposes only.

**Table of Contents**

[Lab 1: Create an Evo PR 1](#_Toc164877705)

[Lab 2: Verifying Your Setup and Checking Your Access to UNIX, Git, SVN, and BaaS 6](#_Toc164877706)

[Lab 3: Create a Sandbox 16](#_Toc164877707)

[Lab 4: Create a Build 19](#_Toc164877708)

[Lab 5: Navigate Git Repositories 25](#_Toc164877710)

[Lab 6: Working with Lab Devices 36](#_Toc164877711)

[Lab 7: Creating a Virtual PTX and MX 46](#_Toc164877712)

[Lab 8: Searching Source Code Repository 54](#_Toc164877713)

[Lab 9: Test Code Using Static Analysis (Coverity) 56](#_Toc164877715)

[Lab 10: Code Coverage 64](#_Toc164877716)

[Lab 11: Code Review with Gerrit 68](#_Toc164877717)

[Lab 12: Commit Code Using PCT 72](#_Toc164877718)

# Lab 1: Create an Evo PR

1

Introduction

This module shows you how to create Evo Problem Reports (PRs) in the PR Workflow Management System.

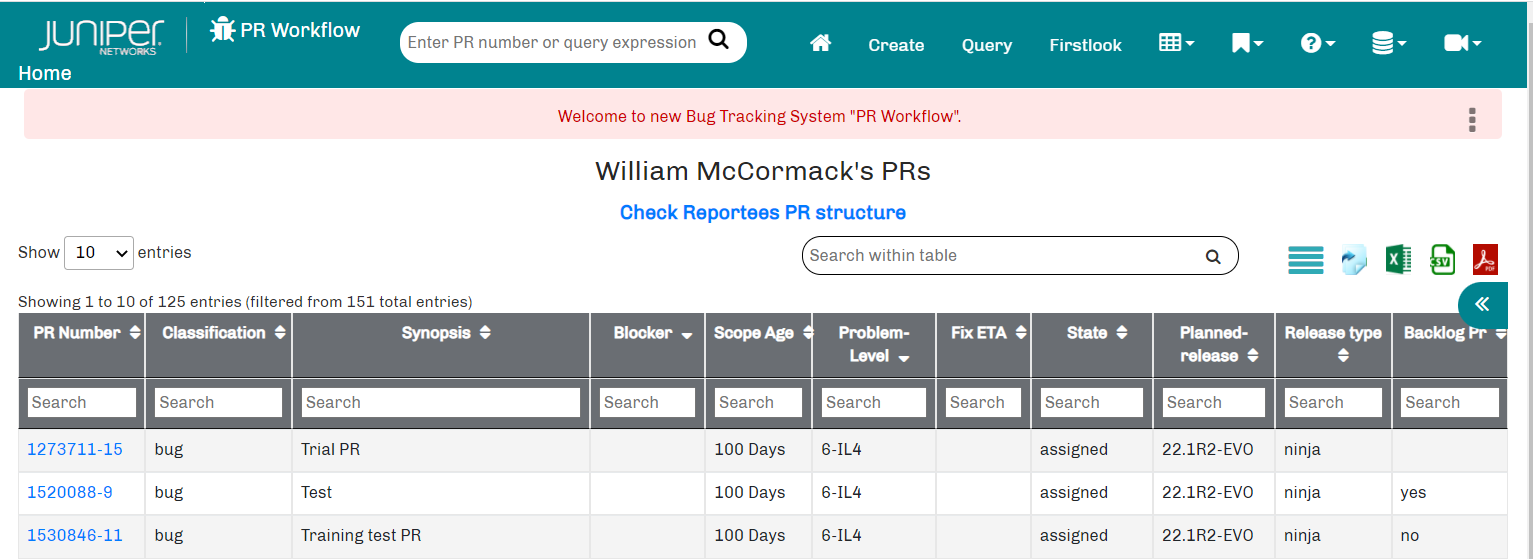
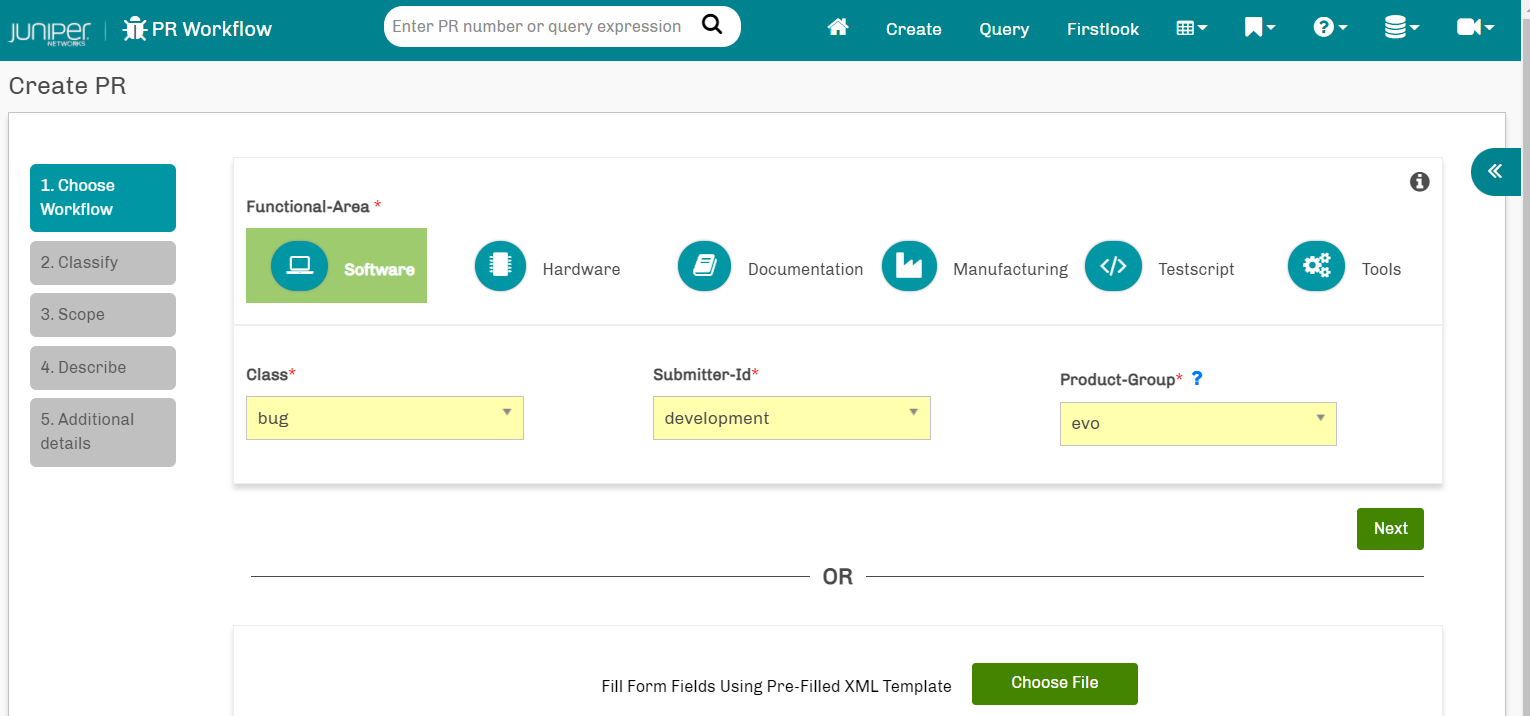
Module Objectives

After completing this module, you will be able to:

* Create an Evo PR
* Create a PR with two scopes: one for Evo and one for Junos
* Exercise 1 – Open an Evo PR with One Scope

Evo uses the **PR Workflow Management System** (PRWF) for bug tracking.

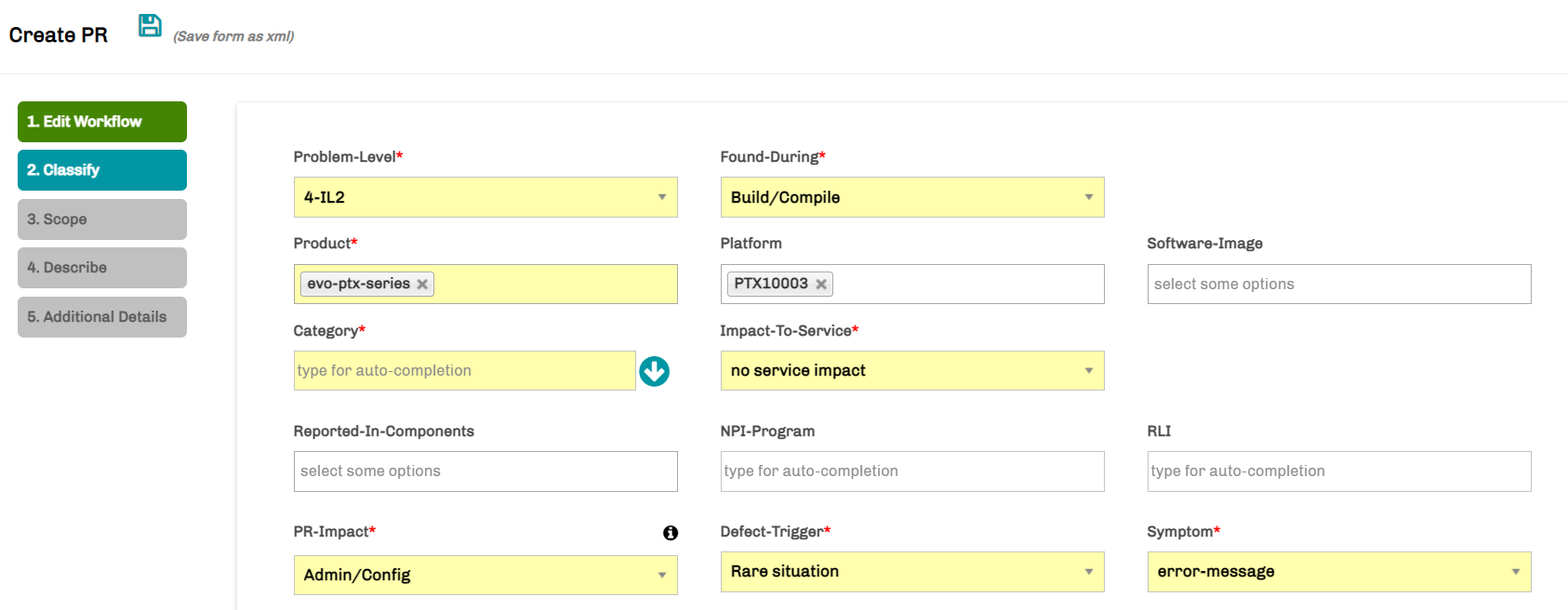
In this exercise, you create an Evo Problem Report (PR) using a category called **evo-dummy-pr.**

1. Open a browser to: <https://gnats.juniper.net>
2. Log in with your Windows ID and password. The Home page is displayed.
3. In the top of your screen, click **create**.
4.   
   The **Create PR** page appears.  
     
   
5. Click the Software icon, then open the menus below and select the following information:

|  |  |
| --- | --- |
| **Field** | **Value** |
| Class | Bug |
| Submitter-ID | Development |
| Product Group | Evo |

1. Click the **Next** button. The **Classify** page opens.
2. Fill in the following information:

|  |  |
| --- | --- |
| **Field** | **Value** |
| Problem Level | 4-IL2 |
| Product | evo-ptx-series |
| Category | evo-dummy-pr |
| Found-During | Build/Compile |
| Platform | PTX 10003 |
| Impact-to-Service | No service impact |
| PR-Impact | Admin/Config |
| Defect-Trigger | Rare situation |
| Symptom | error-message |

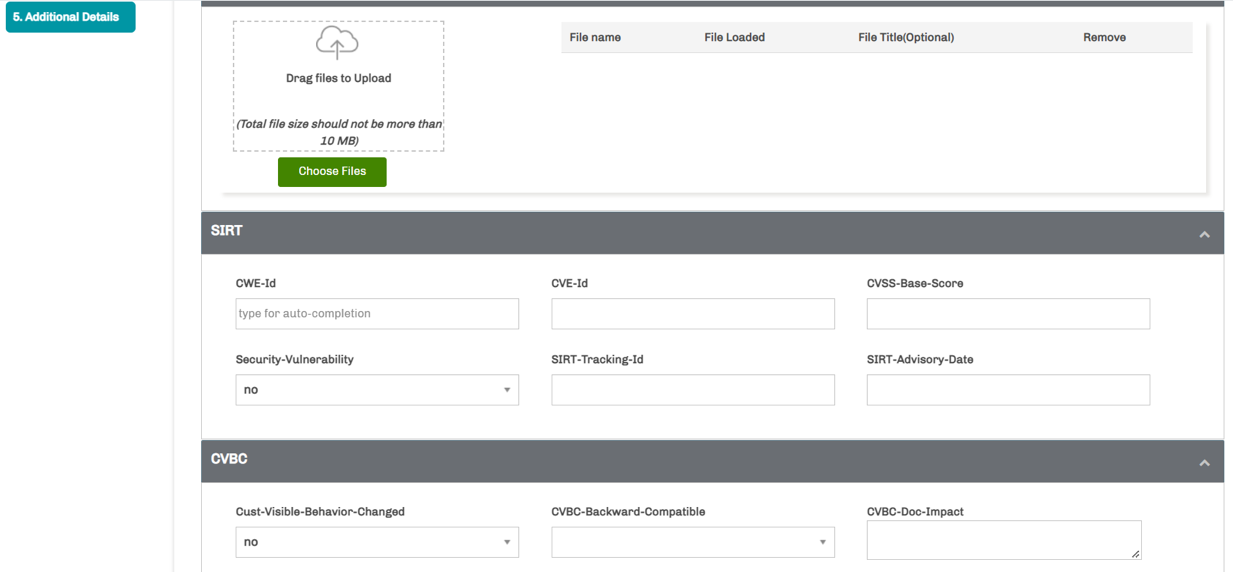


* Fill in the Scope Page

1. Click the “**Next**” button.
2. In the Scope page, enter the following information:

|  |  |
| --- | --- |
| Planned-Release | evo-training-prs |
| Reported-In | evo-training-prs |

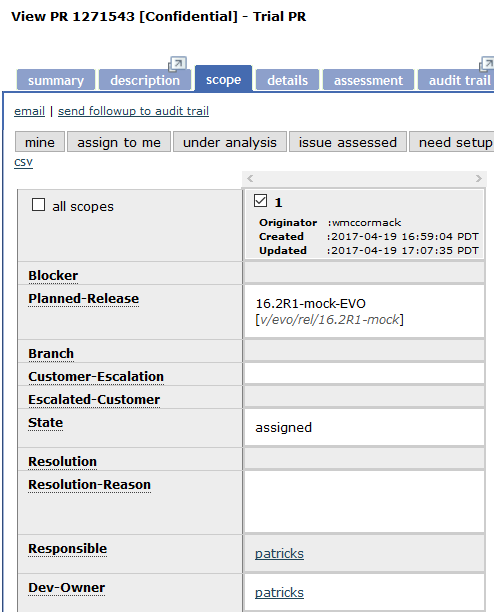
* Fill in the Additional Details Page

1. Click the “**Next**” button.
2. In the **Synopsis** field, type in some text such as “**Test PR**.”
3. In the **Description** field, enter some text such as “**Test Software Bug PR**.”
4. Click the “**Next**” button.
5. In the **SIRT** section, select “**No**.” This section asks if the bug represents a security vulnerability.
6. In the **CVBC** section, select “**No**” This section is asking if the fix will cause product changes (in the CLI, default values, behavior etc.) that will impact customers and require documentation or test suite updates.  
   
7. Click **Submit**.

The wizard disappears, and the summary page appears. 

* Edit the PR and Assign a Developer to the Bug

Assign yourself as the developer for this bug.

1. Open the **Scope** tab.
2. Click the **Edit** button at the top right of the screen. 
3. In the **Dev-Owner** field, enter your username.  
   
4. In the **State** field, select **Under Analysis**.
5. Click the **Submit** checkmark at the top right of the page. The **Dev-Owner** field should show your name and the state should be **Under Analysis**.

* Add Information about the Root Cause of your PR to Fix-Info Tab

1. Open the **Fix-Info** tab. This tab is where you put information about the problem you are fixing and your solution for fixing it. It is required for code review.
2. Click **Edit**.
3. In the **Root Cause** section, add a brief description. Failure to add information here will later cause your code review request to be blocked.
4. Click **Submit**. The PR is submitted.

2

# Lab 2: Verifying Your Setup and Checking Your Access to UNIX, Git, SVN, and BaaS

Introduction

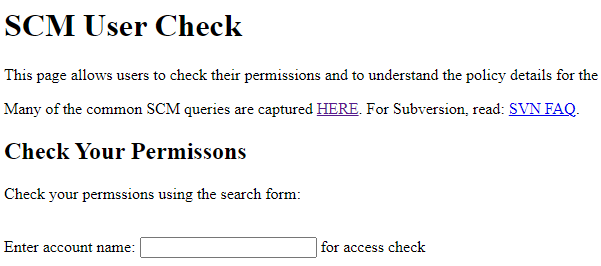
In this lab, you verify you have the proper access and setup to use the various tools for software development.

Module Objectives

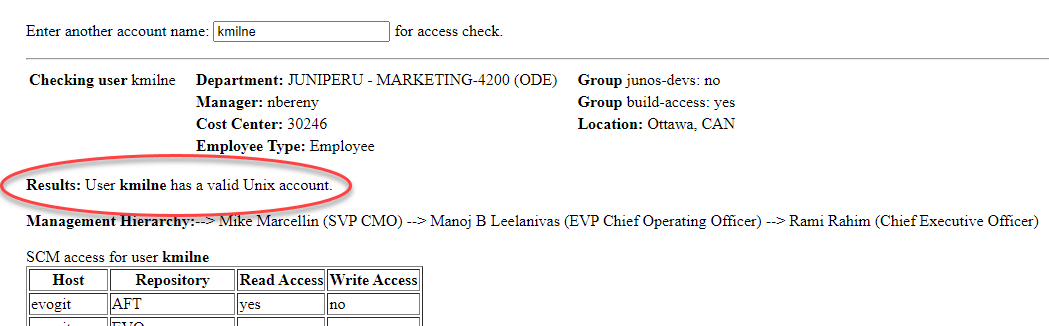
* Verify you have access to access to various systems
* Log into a BaaS build server and verify setup details
* Exercise 1 – Verify You Have a UNIX Account

In this exercise, you will verify your UNIX account is active.

1. Open a browser and go to **https://scm-access.juniper.net/cgi-bin/user\_check**.
2. Type your user ID in the text box, and press **<Enter>**.

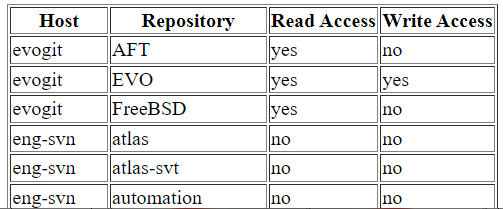


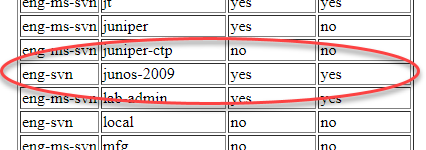
1. The results page should indicate you have a valid Unix account.



* Exercise 2 - Check Your Junos OS Source Code Repository Access Status

In this exercise, you will verify your access to the Subversion (SVN) repository.

1. On the results page above, scroll down to the table and find the repositories **EVO** and **junos-2009**. Your read/write access for each should show **yes/yes**.  
   



* Exercise 2: Subscribe to Evo Security Group and Email Group

1. Open <https://mygroups.juniper.net> .
2. Subscribe to **psg-evo-eng**. This is an alias that will give you access to Evo related discussions.

* Exercise 3 - Connect to the BaaS Build Environment and Verify You Have a Home Directory

In this exercise, you will log into your BaaS build server and verify you are in your home directory. Currently there are two build server environments, located in Bangalore and Quincy, Washington.

1. SSH into one of the build servers (using your corp credentials):   
   bng-baas-shell.juniper.net (Bangalore)qnc-baas-shell.juniper.net (Quincy, Washington)
2. Type pwd to verify you have a home directory. The output should show /homes/<your\_user\_ID>.

* Exercise 4 - Verify Your UNIX Security Groups

In this exercise, you will verify your account is configured with the required UNIX security groups.

1. Type id <userID> and review the output. The output should include the following items:

400883(scm\_access\_junos\_rw)

950(build-access)

499658(baas-users)

238917(psg-evo-eng) This will appear after your subscription to it takes effect.

* Exercise 5 – Add Build Tool Paths and an SVNREPO alias to the .profile File

In this exercise, you will add paths for the various build tools to the .profile file, and also setup an SVN alias to the source code repository.

1. Open the **.profile** file in your home directory with a text editor (this lab guide uses vi).

$ **vi .profile**

**i** (to insert text in vi)

1. Add the following (**all on one line**):

export PATH=$PATH:/volume/baas\_devops/bin:/volume/buildtools/bin:/volume/labtools/bin:/volume/labtools/bin/ccc

1. If you will be working on SRX or QFX, add these additional paths onto the end of the line above:

/volume/ssd-tools/bin

/volume/fsgtools/build-tools/yocto-build

1. Separate from the path statement, on a new line define an SVN alias to the source code repository by adding the following:

SVNREPO=svn+ssh://svl-svn.juniper.net/junos-2009; export SVNREPO

**NOTE**: The command above assumes you are located in Americas. If you are located in EMEA/APAC, change svl (Sunnyvale) to bng (Bangalore).

1. Save and close the .profile file.

**<Esc>**, then **:wq**

1. Type source .profile toimport the changes into your current shell session.
2. Verify the path and environment variable changes. You should see the paths you added above.  
   $ **echo $PATH**  
   /sbin:/bin:/usr/sbin:/usr/bin:...:**/volume/baas\_devops/bin:/volume/buildtools/bin:/volume/labtools/bin:/volume/labtools/bin/ccc:/volume/ssd-tools/bin:/volume/fsgtools/build-tools/yocto-build**:...
3. To further verify the paths are setup correctly, type which baas, then which mksb, then which res. Each command should return the tool’s full path.

$ **which baas**

/volume/baas\_devops/bin/baas

$ **which mksb**

/volume/buildtools/bin/mksb

$ **which res**

/volume/labtools/bin/res

1. To verify the SVN alias is setup correctly, type echo $SVNREPO. The command should return the path you specified above.

$ **echo $SVNREPO**

svn+ssh://svl-svn.juniper.net/junos-2009

* Exercise 6 – Generate SSH Keys

In this exercise, you will generate SSH keys (or verify you already have them). SSH keys are needed to interact with some build tools.

1. View the ~/.ssh directory and look for two SSH key files, such as id\_rsa and id\_rsa.pub.
2. If you have these files you’re all set, move to the next exercise; if you don’t have these files (or even the .ssh directory), create a .ssh directory and then change directories into it and run the command ssh–keygen. This command will create create two security keys.
3. When prompted for a filename for key(s), *do not* provide a name, just press Enter. This will use the default names (id\_rsa and id\_rsa.pub).
4. Enter a passphrase when prompted.

**NOTE**: You must enter an actual passphrase here. If you just press enter (no passphrase), the system will auto-delete your keys within 24 hours.

1. The output should then confirm the keys have been generated.

$ **ssh-keygen**

Generating public/private rsa key pair.

Enter file in which to save the key (/homes/kmilne/.ssh/id\_rsa): **<Enter>**

Enter passphrase: **<Passphrase>**

Enter same passphrase again: **<Passphrase>**

Your identification has been saved in id\_rsa.

Your public key has been saved in id\_rsa.pub.

The key fingerprint is:

64:79:de:c1:13:5a:29:e1:c5:a5:d8:cf:d0:6c:cf:a3 User@qnbaasshell2

The key's randomart image is:

+--[ RSA 2048]----+

| .o+o. |

| o.Bo= |

| + =.B + |

| o o . B o |

| S . . o.o|

| . .|

| E |

| |

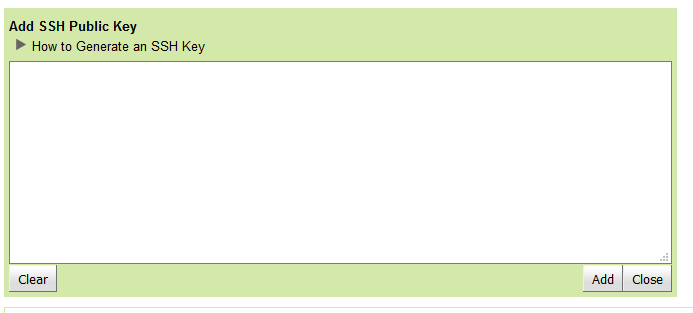
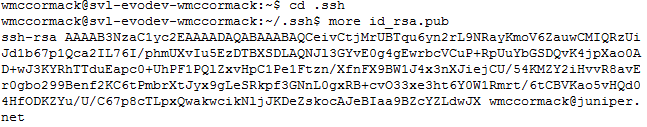
| |

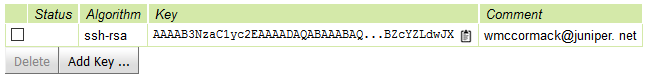
+-----------------+

1. View the ~/.ssh directory again to verify the two key files, names, id\_rsa and id\_rsa.pub, were created.

* Exercise 5: Upload the SSH Public Key to Gerrit

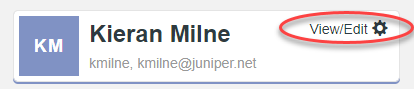
The ssh-keygen command shown above generates a public key called **<key name>.pub** which you will now upload to a Gerrit server. Uploading the public key gives you access to Evo Git repos.

1. Go to the URL <https://svl-evogit-01.juniper.net/#/settings/ssh-keys> .
2. Sign in using your corporate password.
3. Click the **Add** button. You will be prompted for the public key you use for Git access.  
   
4. To get the public SSH key, open your SSH window.
5. cd to **~/.ssh**
6. Type **cat id\_rsa.pub** This will display the public key.  
     
   
7. Press **control-c** to copy the text.
8. Return to your browser and paste the key text into the window. When you paste the key, make sure there are no new line characters (\n) in the key.

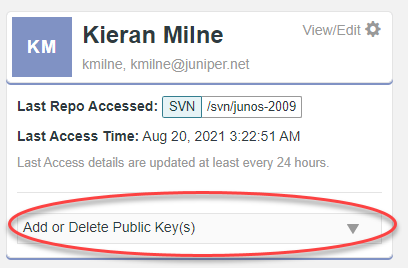
* Exercise 7 - Publish Your Public SSH Key to Subversion

In this exercise, you publish your public key to Subversion (or verify it is already published there).

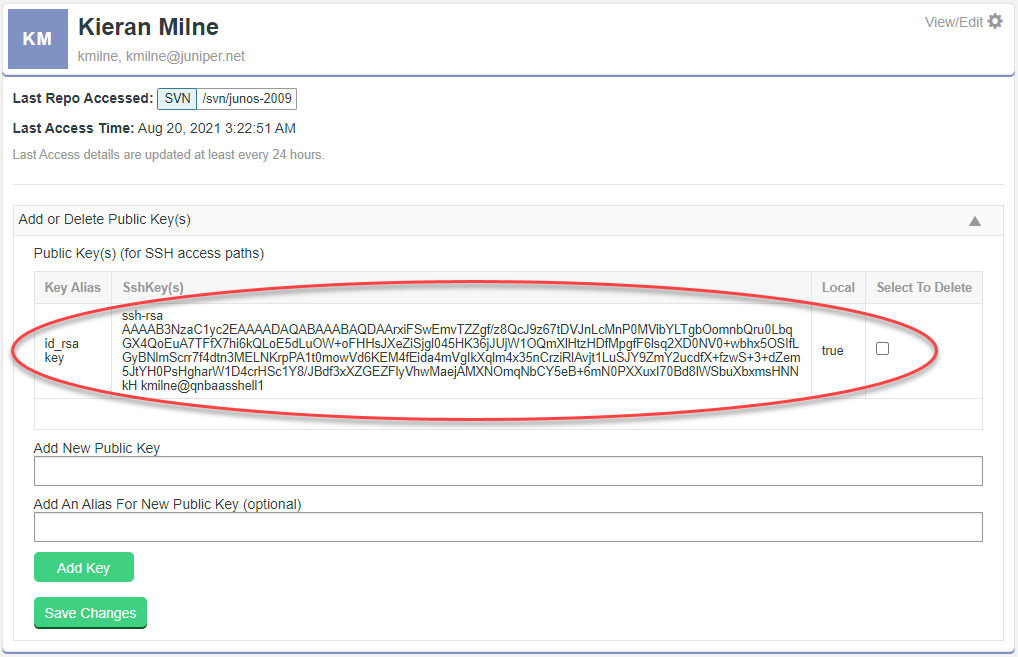
1. Open a browser and go to **https://eng-svn.juniper.net:8084/ui/#/selfservice**. You may need to login.
2. On the landing page, click **View/Edit** next to your name.



1. In the expanded view, click **Add or Delete Public Key(s)**.



1. If your public key is already added you’re all set, move to the next exercise; if your key is not added, go to the next step.



1. In the BaaS server CLI, show (cat) your public key file and copy the *entire* output.
2. Return to the browser, paste the public key string into the Add New Public Key field.
3. Click **Add Key**, then **Save Changes**.
4. You have now added your public key into SVN, which will enable you to interact with this system later. Note it may take up to 15 minutes for the change to take effect.

* Exercise 8 – Create an SSH Config File

In this exercise, you will create an SSH config file (or verify the existing file is setup correctly). The correct settings are needed to interact with some build tools.

1. Open the file ~/.ssh/config and add your home directory and ssh keyname.
2. Open (or create) the config file and add the lines above. Make sure you indent as shown below.

$ **vi ~/.ssh/config**

**i** (to insert text in vi)

**For SVL users**, make sure the following entries exist in your **~/.ssh/config** file exactly as below.

Host evogit

< == Gerrit server

Hostname svl-gerrit.juniper.net

Port 29418

IdentityFile ~/.ssh/id\_rsa

Host evogit-push

Hostname qnc-gerrit.juniper.net

Port 29418

IdentityFile ~/.ssh/id\_rsa

Host emake\_cm

Hostname svl-evodev-review

**For BNG users**, make sure the following entries exist in your ~/.ssh/config file exactly as below.

Host evogit

< == Gerrit server

Hostname bng-gerrit.juniper.net

Port 29418

IdentityFile ~/.ssh/id\_rsa

Host evogit-push

Hostname qnc-gerrit.juniper.net

Port 29418

IdentityFile ~/.ssh/id\_rsa

**For QNC users**, make sure the following entries exist in your ~/.ssh/config file exactly as below.

Host evogit

Hostname qnc-gerrit.juniper.net

Port 29418

IdentityFile /homes/<yourid>/.ssh/id\_rsa

Host evogit-push

Hostname qnc-gerrit.juniper.net

Port 29418

IdentityFile /homes/<yourid>/.ssh/id\_rsa

Host linuxgit-push

Hostname qnc-gerritln.juniper.net

Port 29418

IdentityFile ~/.ssh/id\_rsa

1. Save and close the config file.

**<Esc>**, then **:wq**

* Exercise 9 – Verify your SSH Key Setup

In this exercise, you will load your private SSH key and verify that your setup is working correctly.

1. Type b to start the utility.
2. Type ssh-add (if no arg, it assumes id\_rsa), and enter your passphrase when prompted.
3. Type ssh-add -l to verify your key is loaded correctly.

$ **ssh-add -l**

2048 SHA256:NVaWX6LAtrztxR0gZ00O4LQgWOwT++zWB0mrMxCcZxM /homes/kmilne/.ssh/id\_rsa (RSA)

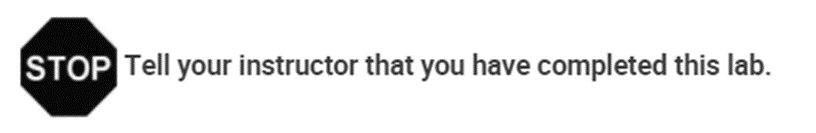
1. Enter one of the following commands, based on your location:

$ **svn info svn+ssh://svn@bng-svn.juniper.net/junos-2009** (Bangalore)

$ **svn info svn+ssh://svn@qnc-svn.juniper.net/junos-2009** (Sunnyvale/US)

1. Were you prompted for a passphrase or password?

* Yes – something is wrong with your setup. Review the previous sections and ask your instructor for help to correct the issue.
* No – your setup is working correctly, you’re all set!



# Lab 3: Create a Sandbox

3

Introduction

In this lab, you create two volumes and two sandboxes.

Module Objectives

When you complete this lab, you will be able to:

* Log into a BaaS server
* Create two volumes
* Create two sandboxes
* Exercise 1 - Connect to the BaaS Build Environment

If you are still logged into your build server, skip to Exercise 2.

1. Open an SSH session if you are on a Mac or an SSH tool such as **PUTTY** if you are on a PC:
2. Select SSH and enter one of the following addresses:

**bng-baas-shell.juniper.net (=Bangalore) OR**

**qnc-baas-shell.juniper.net (= Quincy, Washington)**

1. Provide your corporate credentials:

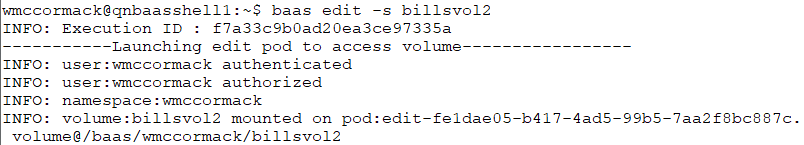
* Login: <your corporate userid>
* Password: <your UNIX password>
* Exercise 2 – Create a Volume

In this exercise, you create a 1500 GB volume.

1. Type **baas ls** to see a list of your volumes. You will have no volumes at this point.
2. Type **baas createvol -s <volumename> -z 1500Gi** to create a 1500 GB volume.

* Exercise 3 – Edit Your Volume

In this exercise, edit your volume and create a sandbox.

1. Type **pwd** to see where you are. You should be in your home directory **/homes/<username>**.
2. Type **baas edit -s <volumename>** 
3. Type **pwd** to see where you are (or just read your prompt.) You should be in **/baas/<username>/<volumename>.**
4. Make sure your ssh agent is running and has your passphrase. Type **ssh-add** followed by your passphrase when prompted. If not prompted, run the ssh agent by typing **eval `ssh-agent.sh`** Then run **ssh-add** to add your passphrase.

* Exercise 4 - Create a Full Sandbox Based on a Private Branch

Create a volume and a sandbox that we will use at the end of the class for code review and commit.

1. Open a new SSH session and connect to one of the following addresses:

**bng-baas-shell.juniper.net (=Bangalore) OR**

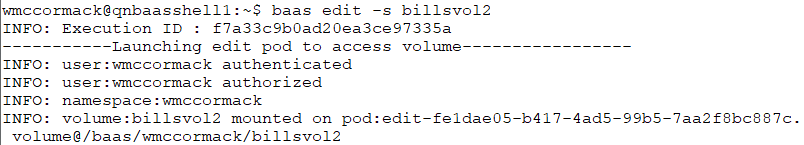
**qnc-baas-shell.juniper.net (= Quincy, Washington)**

1. Create a new volume: **baas createvol -s <name> -z 1500Gi**
2. Edit the volume with the command: **baas edit -s <name>**
3. Create a sandbox with the following command:  
   **sb create -n <sbname> -p evo-dev-training -L**

Where **-n** = “name of my sandbox” **-p** = project name (in this case, “evo-dev-training”) **-L** means use pre-compiled object files. Example: **sb create –n BillsSandbox –p evo-dev-training -L**  
At this point, the sandbox will be created. The process takes approximately 10+ minutes.

* Exercise 5 – Create and Edit a Volume

We are now going to create a second volume and sandbox. This will be a sparse sandbox that we will build in the next chapter.

1. Type **baas ls** to see a list of your volumes. You will have one volume at this point.
2. Type **baas createvol -s <volumename> -z 1500Gi** to create a 1500 GB volume.
3. Type **pwd** to see where you are. You should be in your home directory **/homes/<username>**.
4. Edit your volume with the command: **baas edit -s <volumename>** 
5. Type **pwd** to see where you are (or just read your prompt.) You should be in **/baas/<username>/<volumename>.**

* Exercise 6 - Create a Sparse Sandbox

You will now create a sparse sandbox based on the *pio* component. Pio stands for “Platform IO,” and is a PFE-related component.

1. Make sure you are in **/baas/<username>/<volumename>**
2. Create a sandbox with the following command: **sb create -n <sbname> -p evo pio**

At this point, you should see your sandbox being created.

# Lab 4: Create a Build

4

Introduction

In this module, you explore the directories in your sandbox and then execute a build using the ***sb make*** command. Once the build is running, we will examine builds created by the automated build system.

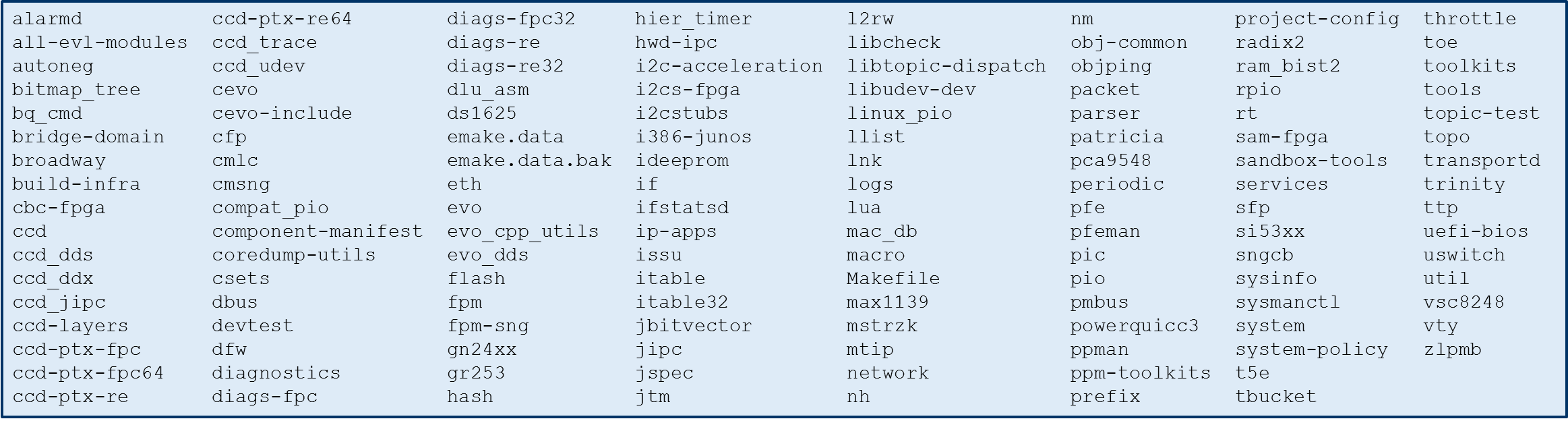
Module Objectives

After successfully completing this module, you will be able to:

* Understand the contents of your sandbox
* Create a build
* Examine the automated build environment
* Exercise 1 – Enter your sandbox

In this exercise, you will examine a sandbox.

1. Once your first sandbox has been created (based on **evo-training-prs**), move into it by typing: **cd <sandbox>**
2. View the contents by typing **ls**

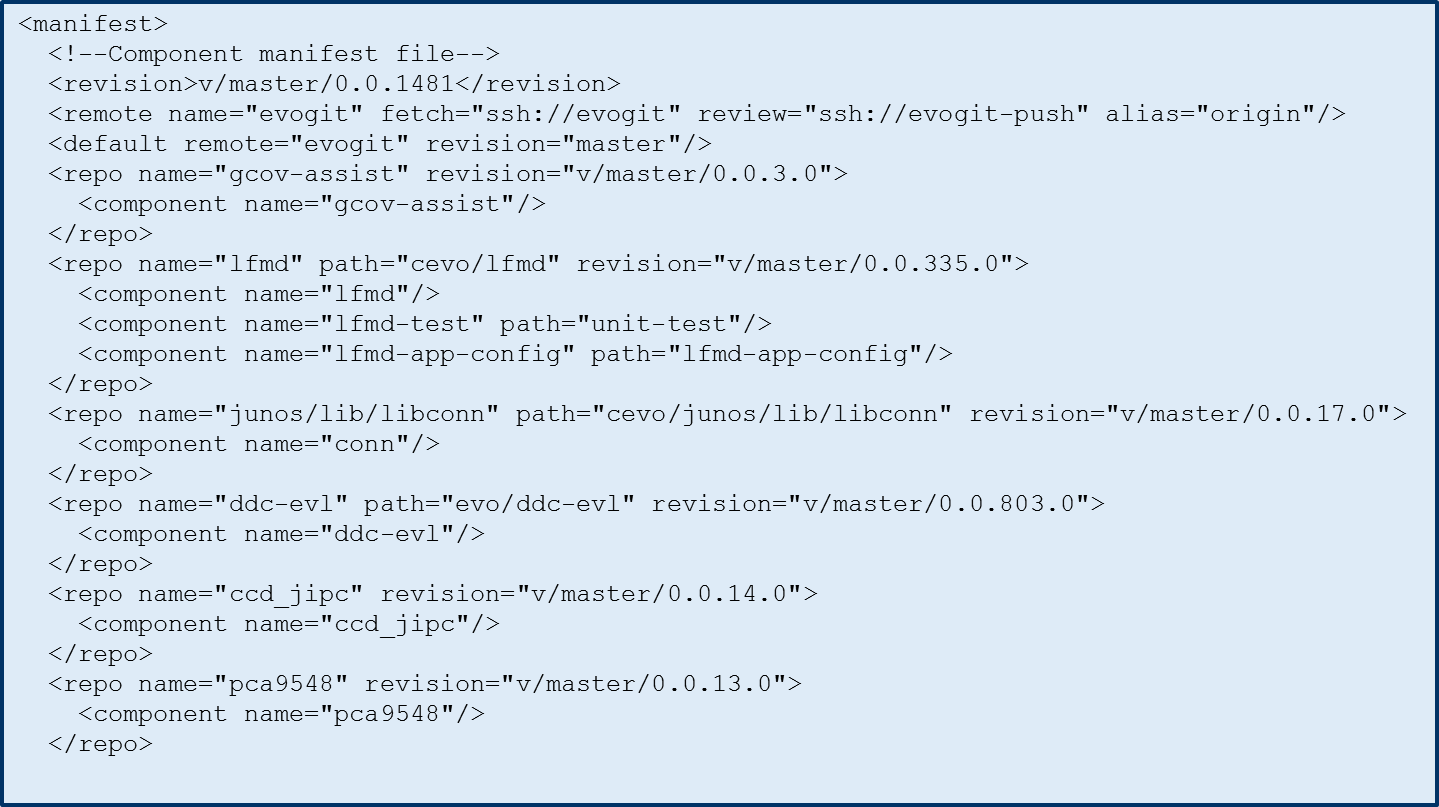


* Exercise 2 – Examine the component manifest

The component manifest file is an XML-based mapping file that identifies the repositories in your checkout. The component manifest is named **component-manifest.xml**.

1. In your ***<sandbox>***, let’s move into the component manifest directory. To do this, type:   
   **cd component-manifest**

To examine the contents of the XML file, type: $ **more component-manifest.xml**



* Exercise 3 – Examine a typical component

Components are the smallest units of code that will build. A component is a unit of work that is small enough you can test on its own. Here are three examples:

* rpd
* alarmd
* chassisd

We will go view a typical component and the directory structure.

1. In your ***<sandbox>***, move back to the top of your sandbox: **cd ..**
2. Let’s move into the directory for the component alarmd. To do this, type: $ **cd alarmd**

Type **ls** to examine the directory structure:



* **/src** directory for the source of the component.
* **/common** directory for the common code of the component

1. To **view** the structure of the **/lib**, let’s move into it by typing: $ **cd lib**

Type **ls** **to** examine the directory structure. Do the same for **/src** and **/common**. **/src** has the source files for **alarmd**.

1. Let’s go to the root of your sandbox by typing: **$ cd ..**

* Exercise 4 - Examine the “cevo” Directory

The cevo directory holds some of the typical daemons in Junos. We will search for some of these.

1. Let’s move into the ***cevo*** directory. To do this, type: $ **cd cevo.** You should be in **<sb>/cevo**.
2. We will look for the routing protocols daemon (rpd) by typing: $ **find . -name rpd**



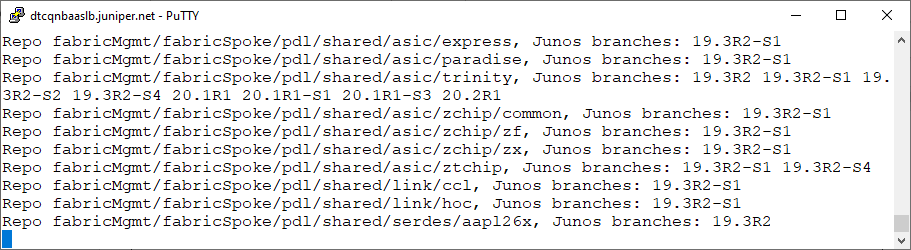
Sample Output

1. Examine the contents of these files and directories to become familiar with them.

* Exercise 5 – View the existing SubGit Repositories

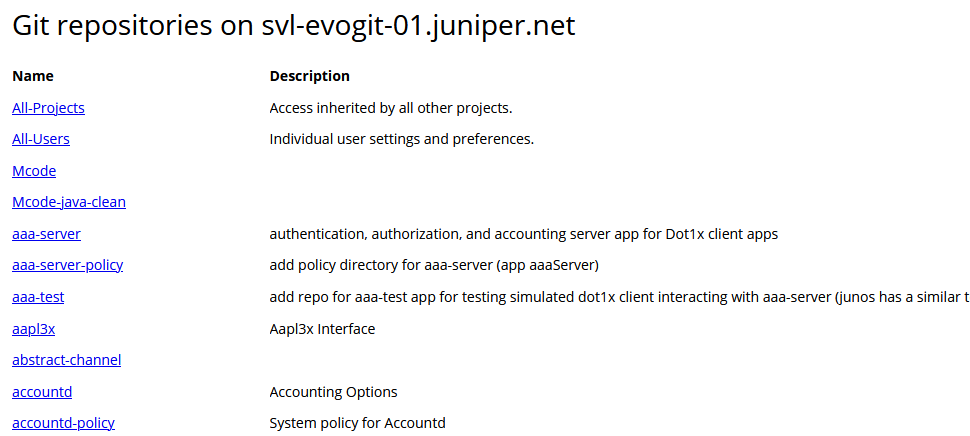
To understand better some of the single sourced repositories, you can view the SubGit repositories.

1. Inside your sandbox, type the following: **$ sb show-subgit-repos**

Sample Output

* Exercise 6 – View the existing Evo Git Repositories

To understand better some of the sandbox directories, you can view the Evo Git repositories with a brief description and owner.

1. Open a browser and enter the URL type: <https://svl-evogit-01.juniper.net/admin/repos>
2. You will need to log in. Click the sign-in link on the top right of your screen.  
   

Sample Output

1. Scroll and view the existing repositories which will have a very brief description of the repository and who the owner of that repository.

* Exercise 7 – Build your Sparse Sandbox

In this module, you will build your **sparse sandbox,** based on the **pio** component**,** using the ***sb make*** command.

1. Type **cd <volumename>/<your sparse sandbox>**

You should now be in the **<sandbox>** directory.

1. Start your build by typing: **baas build -b "sb make UT\_FAIL\_IGNORE=1" -x** This command will start the build process. Your build should take about 30 to 60 minutes to run.

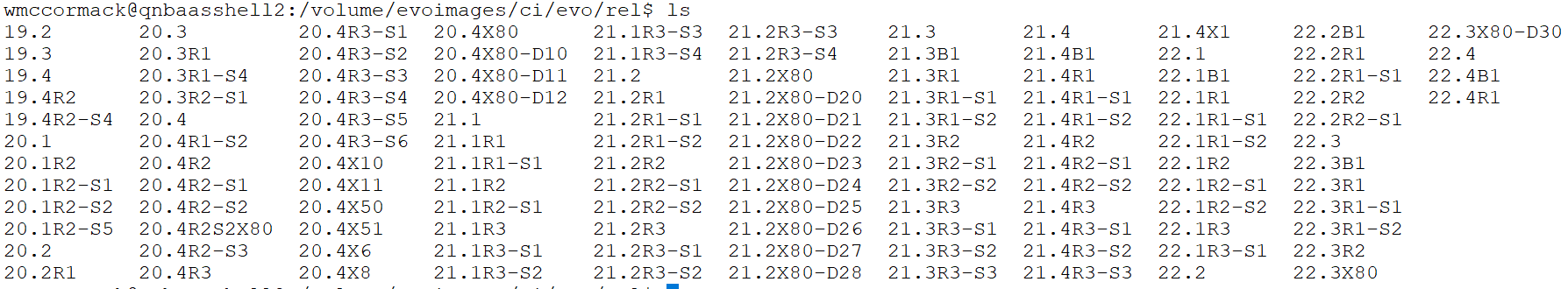
* Exercise 8 - Examine Releases Created by Automated Build System

While your build is running, we can look at builds created by the automated build system.

1. Change directories into **/volume/evoimages/release/evo/rel**. This directory contains published releases for Evo.
2. Take a few minutes to examine the contents of this directory.

* Exercise 9 – Examine CI Images Created by the Automated Build System

“Continuous Integration” (CI) directories contain images produced by the automated CD/CI build servers. In this exercise, you examine CI images directories.

1. Change directories to **/volume/evoimages/ci/evo/rel**
2. Type **ls**
3. Move into these directories (cd into them)

* **Any release ending in** R1 These are the major releases.
* **Any release ending in R2** **or R3** These are the maintenance releases.

1. Examine the contents of these files and directories to become familiar with them.

# 

5

# Lab 5: Navigate Git Repositories

Introduction

In this lab, you will use Git commands to view source code contained in multiple branches.

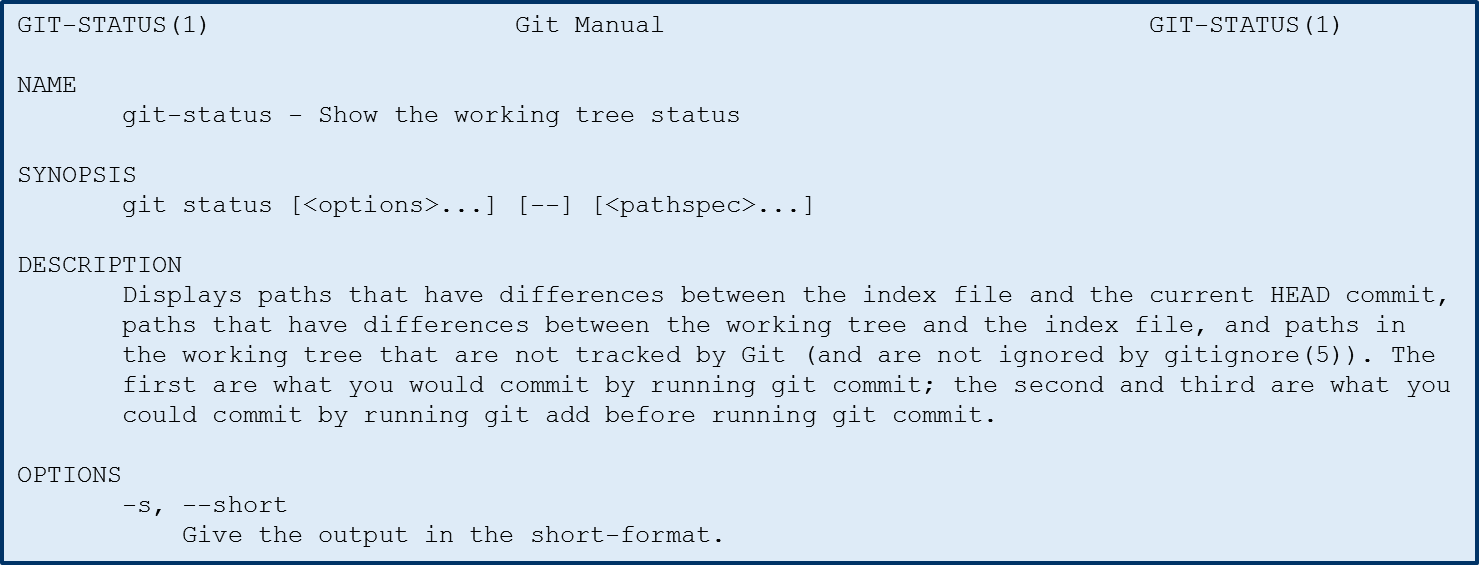
Module Objectives

* View and create branches in a Git repository
* Locate and view pointers in a Git repository
* Modify and add files to a Git repository
* Track changes to a Git repository
* Rebase a Git repository
* Exercise 1 – Git Help

In this exercise, you will use Git help to understand commands.

1. To show all available commands in Git, type the following:  
   $ **git help -a**
2. To view help for a specific commands, type the following:  
   $ **git help <command>**

For example, **git help status**. This command will give you help on “status.”



Sample Output

* Exercise 2 – Create a Git Repository in Your Home Directory

In this exercise, you will and create a directory in your home directory and then run the ***git init*** command to transform this directory into a Git repository, i.e., a .git directory capable of doing versioning.

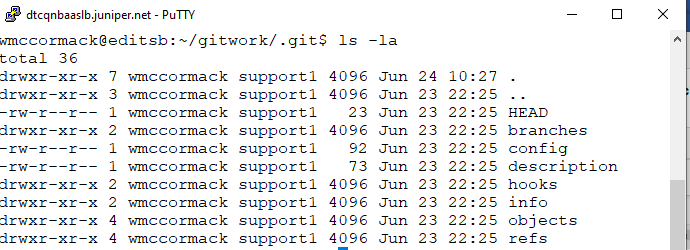
1. We will create a directory in your **/home** directory, type:  
   $ **cd**

This will ensure you are at your **/homes/<userid>** directory. To validate, type: **pwd**

1. To create a directory, type:  
   $ **mkdir gitwork**
2. Let’s move into the new directory and create a new git repository by typing:  
   $ **cd gitwork**
3. Next, let’s create a new git repository by   
   **$ git init**



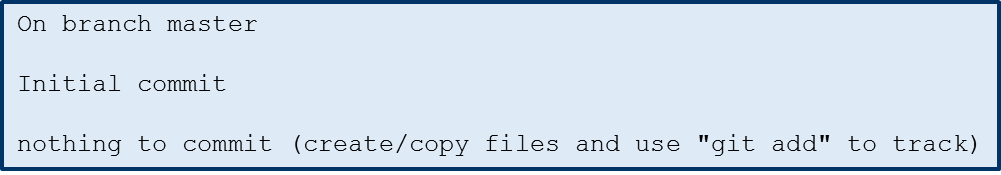
Sample Output

1. Do an **ls -la** to see the files in <sb>/gitwork. You should see a directory called **.git.**
2. Change directories to gitwork/.git.
3. Do an **ls -la** to see the git-related files that were created when you did the **git init** command earlier.  
   
4. Type **cd ..** to move back to <sb>/gitwork.

* Exercise 3 – Add and track a file moving into Git repository

In this exercise, you will create a file, insert some text to the file, and add this file to your Git repository.

1. Let’s initially check the status of our directory with git by typing:  
   $ **git status**

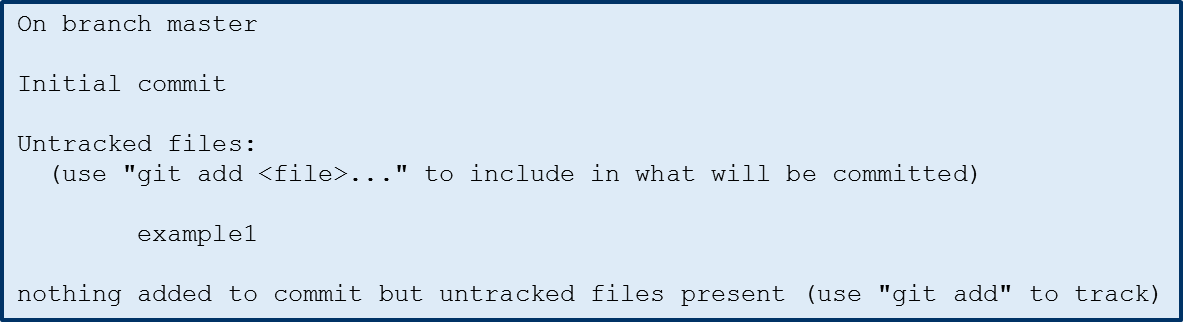


This validates we are pointing to the **master** branch and there is nothing in the staged/indexed area to commit.

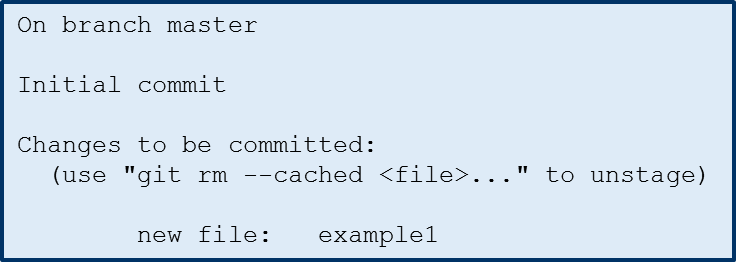
1. Let’s create a file and add a couple of lines to the file like:
2. $ **vi example1**
3. **This is my first line of text**

*Add some text like this.*

1. **Then, I wrote my second line of text**
2. Save and close the file.
3. Let’s see what git thinks after creating example1 file by typing: $ **git status**

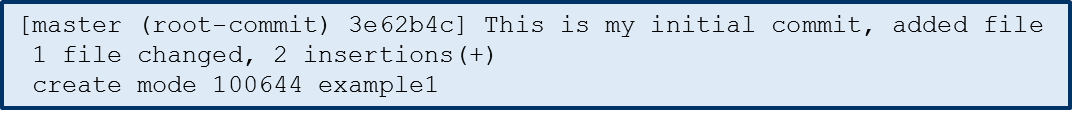


1. This shows example1 is not tracked by git and there is nothing in the staged/indexed area.
2. Let’s add this file to the staged/indexed area by typing:   
   **$ git add example1**
3. **$ git status**



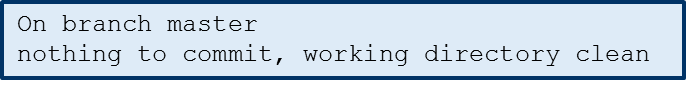
This shows example1 is ready to be committed into the repository.

1. Let’s commit this file into our git repository by typing:   
    **$ git commit -m "This is my initial commit, added file"**



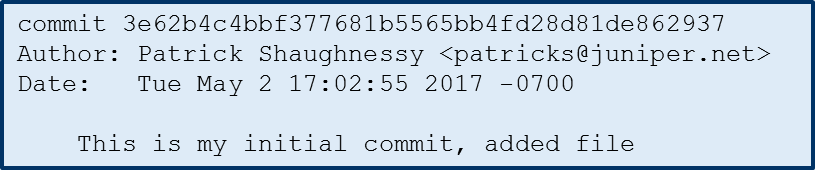
This message shows that 1 file changed with 2 lines with the commit hash [3e62b4c].

1. To verify the commit took place, you can type: **$ git status**



This shows there is now nothing to commit.

1. To validate your commit, type: **$ git log**



1. **$ git show**

* Exercise 4 – Change a Git tracked file and view the changes

In this exercise, you will modify a Git tracked file and view the changes to that file.

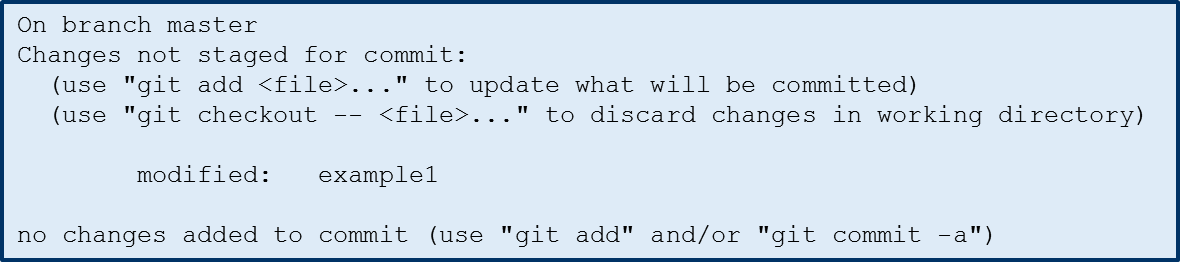
1. Make sure you are in the directory **/homes/<userid>/gitwork** by typing:  
   **$ pwd**
2. Let’s edit example1 since it is now tracked by git. Let’s add and modify a line, type:  
   $ **vi example1**

*This is my* ***1st*** *line of text*

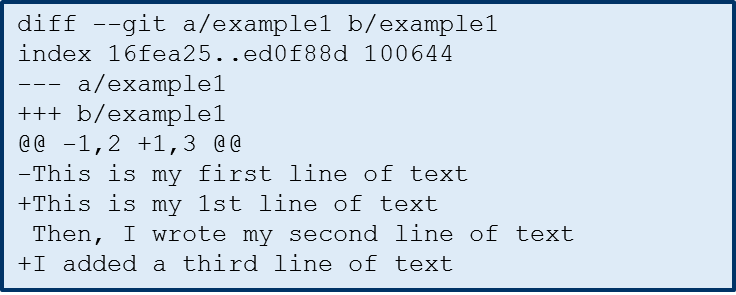
*Then, I wrote my second line of text*

*I added a third line of text*

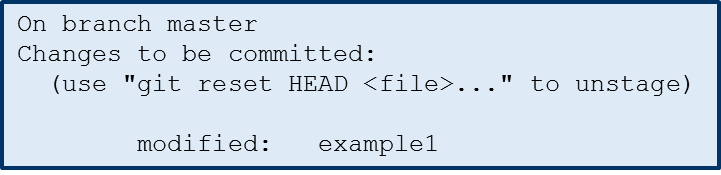
1. Save and close the file.
2. Let’s see what git thinks by typing: $ **git status**



1. Let’s see the changes to example1 by typing: $ **git diff**



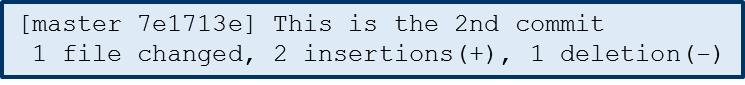
1. We will add these changes and checking the status by typing:   
   $ **git add example1**
2. Then type: **git status**



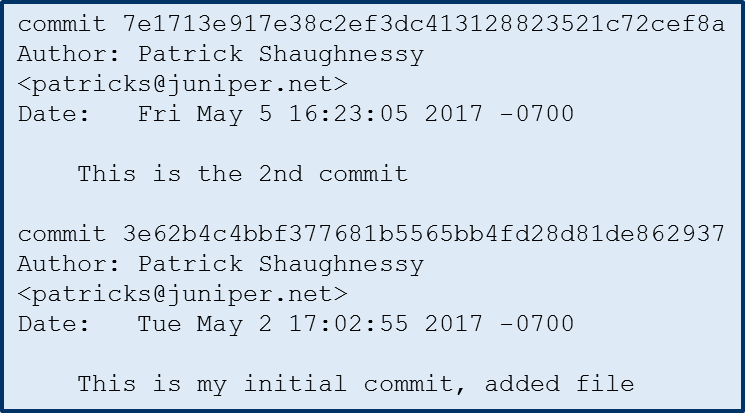
1. Let’s commit these changes by typing: **$ git commit –a**  
   Note this will open an editor.
2. Below the commented lines, add the following commit message: **“This is the 2nd commit”**



1. Save and quit the editor after making the changes.



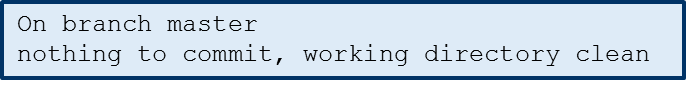
1. To see the existing log messages, type the following: $ **git log**



* Exercise 5 – Using Git Stash command

In this exercise, you will make some changes to a file and see how you can use git stash.

1. Let’s ensure your staged/indexed area is clean by typing: $ **git status**



1. Let’s make some changes to *example1* by typing:  
   $ **vi example1**

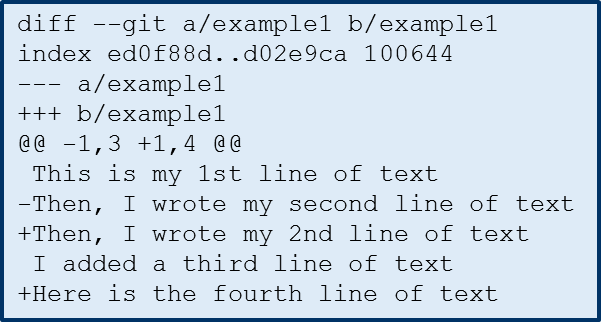
*This is my 1st line of text*

*Then, I wrote my* ***2nd*** *line of text*

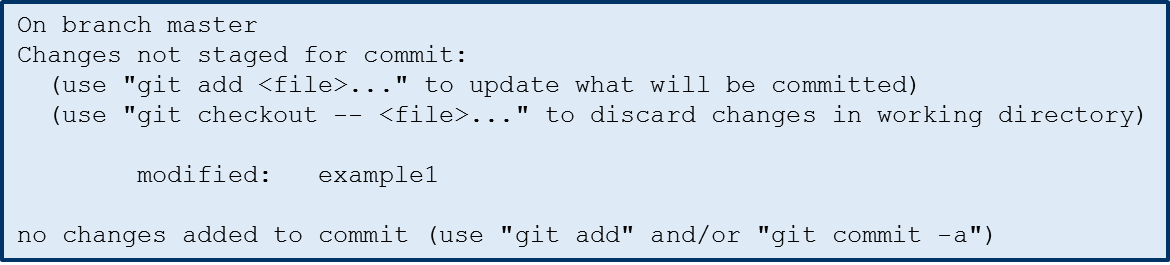
*I added a third line of text*

*Here is a fourth line of text*

1. Save and close the editor.
2. Let’s see the changes by typing: $ **git diff**



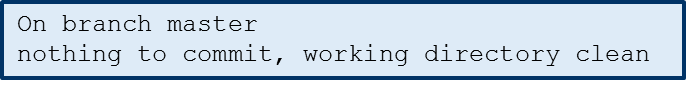
1. Also, you can type:$ **git status**



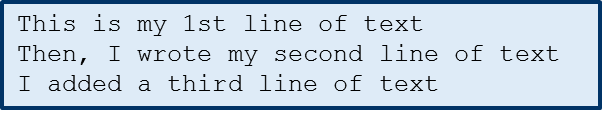
1. Note: Looks like we are ready to commit these changes. However, you may get pulled or called onto something else, so you need to store your changes temporarily.
2. To store these changes, we can use *git stash.* Type the command:   
   **$ git stash save “Chg 2nd and added fourth line of text”**



Check to see your staged/indexed area now by typing: $ **git status**

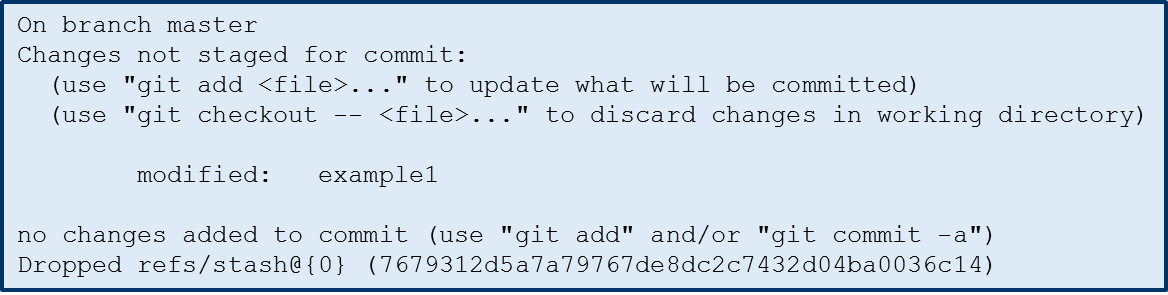
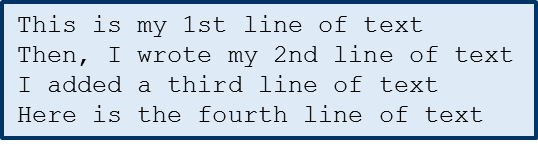
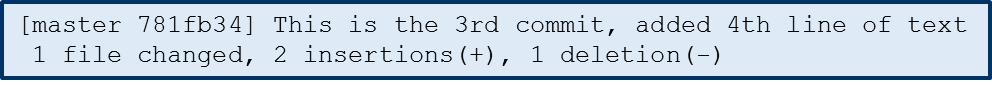
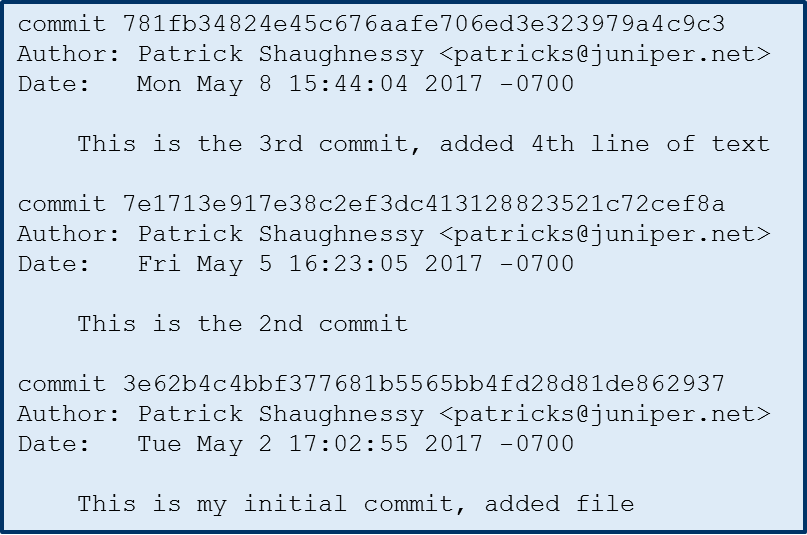


1. Let’s see if I changes are in the file by typing: $ **git diff** *(after the command, what happened?)*
2. Nothing appears. Notice your changes are gone. They are stored in the stash.
3. Let’s see the contents of the file you have been changing by typing: $ **cat example1**



1. Git stash stored your changes in a temporary stack to be recalled later.

Let’s see all of your stashes by typing: $ **git stash list**

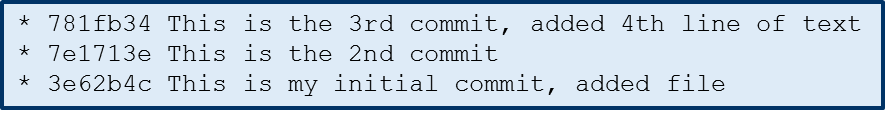
1. To re-apply the changes, I can do that by “popping” the stack by typing: $ **git stash pop**
2. To see if my changes were applied, we can type: $ **cat example1**
3. Note: You can do **git status** and **git diff** to see that all your changes have been restored.
4. Let’s add and commit our changes at the same by typing:
5. $ **git commit -a**
6. In editor, add this text below the comments: "This is the 3rd commit, added 4th line of text"
7. Save the file and close the editor.
8. To see all our commits, we can type: $ **git log**

* Exercise 6 – Using Git rebase and the effects to your branch

In this next exercise, you create a new branch, perform some changes, and rebase this branch back onto the master branch.

1. Let’s see what branches we have by typing: $ **git branch**

Let’s create a separate line of code called a branch by typing: $ **git checkout –b new\_feature**

1. Note: With this checkout, we have created a new branch of code and are pointing to it.
2. To view the available branch we type: $ **git branch**
3. Let’s view a summary of the log messages so far by typing: $ **git log --oneline**
4. To view the contents of the *new\_feature* branch, type: $ **ls**
5. Let’s create a new file *newfeature* by typing $ **vi newfeature**
6. Then enter the following text: *This is my 1st line of text****.*** Then save and close the file.
7. Let’s make some edits to the *example1* file by typing:
8. $ vi example1

*This is my 1st line of text*

*Then, I wrote my 2nd line of text*

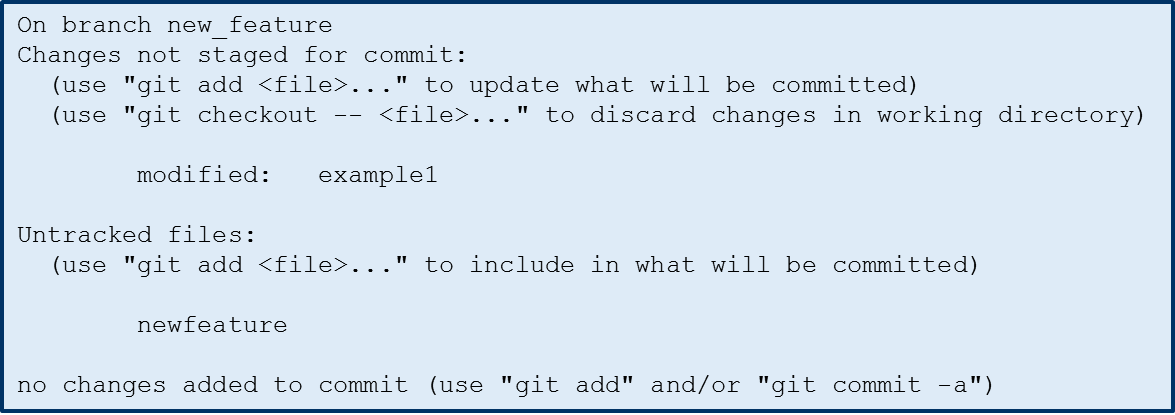
*Changed* ***third*** *to* ***3rd***

*I added a* ***3rd*** *line of text*

*Changed* ***fourth*** *to* ***4th***

*Here is a* ***4th*** *line of text*

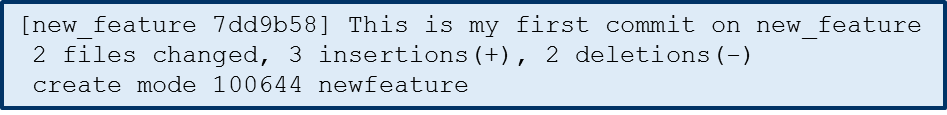
1. Save and close the file.
2. Let’s see what git thinks of the new file by typing: $ **git status**



Let’s add and commit *newfeature* to *new\_feature* branch by typing:

$ git add newfeature

1. **$ git add .** This will stage newfeature for a commit.
2. **$ git commit –m "This is my first commit on new\_feature"**

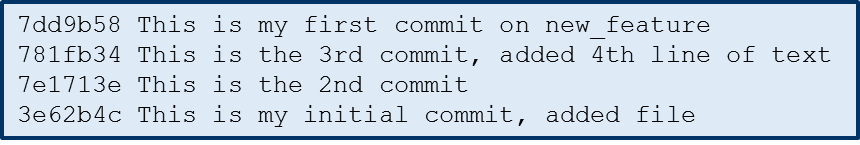


Note: Now we have a change/file, not in the master branch.

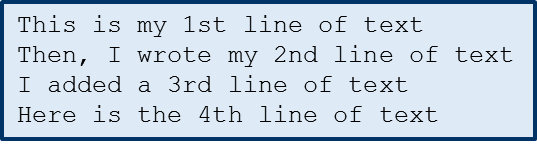
1. Let’s switch to the *master* branch and viewing what is there by typing:
2. $ **git checkout master**
3.  $ **ls**
4. Note: You do not see the newfeature file.
5. To bring changes from another branch into another, you can use git merge or git rebase. In this   
   case, we will use git rebase to bring the changes from new\_feature to the master branch. To do this, we type the following:
6. $ **git rebase new\_feature**  (we will pull the changes from this branch to the one I am pointed to).



1. Let’s see the results of the rebase by typing:
2. $ **ls**
3. 
4. $ git log --oneline



$ cat example1



1. **Note:** git add, git show, git diff, git status, git commit, git log are the most commonly used git commands.

# 

# Lab 6: Working with Lab Devices

6

Introduction

In this lab, you learn how to check out a lab device from a lab using a command line tool called ***res*** and a webtool called ***Lab Resource Manager*** or ***LRM****.* Afterwards, we will learn how to install an image onto a test device.

Module Objectives

Covered here:

* Search for an available lab device
* Checkout a lab device
* Connect to a lab device, move files/builds, and update software
* Check in a lab device
* Exercise 1 - Connect to the Build Environment

If you are still logged into your build server, skip to Exercise 2.

1. Open an SSH tool such as **PUTTY**.
2. Select SSH and enter one of the following addresses:   
   **bng-baas-shell.juniper.net (=Bangalore) OR  
   qnc-baas-shell.juniper.net (= Quincy, Washington)**
3. Provide your corporate credentials:

* Login: <your userid>
* Password: <your UNIX password>

1. Edit your volume:  
   **baas edit -s <volumename>**
2. Move into your sandbox:  
   **cd <sandbox>**You will now be in the directory **<sandbox>.**

Exercise 2 – Search for a Device

In this exercise, you search for lab devices. Most groups at Juniper have a test bed (group of devices in the lab) at their disposal to use. As a suggestion, consult with your mentor and/or manager to find out the test bed available to you.

1. Make sure we have access to the **res** script:

**which res**

**/volume/labtools/bin/res**

If you did not receive the response above, check your PATH and see that **/volume/labtools/bin** is included.

1. To view devices and their status, type the following:

**res –help-cmd sh**

Usage: res sh|show [option] <list of routers>

options:

-a|--attribute show resource attributes

-c|--checkout show checkout resources

-f|--free show currently available resources

-d|--detail detail output

-p|--component show components within a group resource

-h|--help display this usage message

-H|--history display reservation history

-n|--no\_heading display without heading

-N|--no\_truncate do not truncate long output line.

-m|--manufacturer <mfr> mfr: juniper,cisco,intel,juniper-east,...

-o|--output\_fmt <fmt> output format. fmt: same as sprintfx

-P|--period Time period. Work only with -H option

e.g. -P "2003-01-01 - 2003-07-11 23:10"

-r|--regex <regex> regular expression search. regex: any perl

regular expression. Should be used together

with -w option to reduce the scope of search

-s|--sort <sort\_spec> sort output. sort\_spec: field\_name[:asc|:desc]

-w|--where <checkout\_id|bind\_id|group|host|lab|testbed|user|location>

default is host

-q|--query <item> <host|location|testbed> list items

Sample output. This will show you all the available options for the **res sh** command. Take a moment to read and understand the options of this command.

1. Let’s view some devices – devices that are available and are a part of the hardware test bed. Type the following command:

**res sh –f –w testbed hardware**

name model osname testbed

sleater windows windows hardware

sparkys m120 junos hardware

spoon olive Unknown hardware

starbolt mx240 junos hardware

starwars olive junos hardware

Sample output. If you want to see just what is available in a test bed, remove the **–f** from the command above.

1. To view a certain model of hardware, and see what is available, type the following:

**res sh –f –w model mx80**

name model osname testbed

amos mx80 junos calypso-hw

argos mx80 junos wf-wmd

asclepius mx80 junos calypso-hw

award mx80 junos fragrance

azazel mx80 junos calypso-hw

Sample output.

1. To see what machines you have checked out, type the following:

**res sh –w user <user-id>** (for example: res sh –w user ckar)

id bind\_id name model username start finish

340271 340271 gauntlet m10 ckar 2011-01-12 2011-02-11

395009 395009 netbee mx480 ckar 2011-01-17 2011-01-24

383775 383775 nortenos mx480 ckar 2012-02-08 2012-02-08

This will show your outstanding reservations with the start and finish timestamps.

1. To see the availability of a particular machine, type the following:

**res sh <name of device>** (for example: res sh netbee)

id bind\_id name model username start finish

395009 395009 netbee mx480 ckar 2011-01-17 2011-01-24

277708 277708 netbee mx480 gbarrinuevo 2011-01-24 2011-02-01

117868 117868 netbee mx480 gbarrinuevo 2011-02-02 2011-02-11

If there are no outstanding reservations, res will display blank or nothing.

1. Now that you have a flavor for the **res sh** command, try to answer the questions below. What command would you issue to show?

* Devices that were reserved for bind\_id 123456. (Note: 123456 bind\_id does not exist)
* Devices that are checked out for test bed hardware.
* Devices that are free, Cisco devices, located in the systest lab.
* Devices that are free, test bed hardware, and model mx240 (Hint: use regex)
* Exercise 3 – Checking Out a Device

In this exercise, you will checkout a device.

1. To see what options are available for the checkout command, type the following:

**res –help-cmd co**

1. Once you know the particular machine you want to reserve, let’s perform an individual checkout of the device, type the following:

**res co <name of device>**

Date and time to checkout: [2011-01-21 16:26]

Duration(?d ?h ?m): [120m]

Purpose: [scripting] development

Comment: Testing for Training

After completing these four questions, **res** should come back and display your reservation details.

1. To verify your reservation, type the following command:

**res sh <name of device>**

1. To find out more information about a particular device, type the following:

**params-info <name of device>**

[patricks@svl-junos-pool44 ~]$ **params-info fen**

Current information on fen:

Manufacturer..................juniper

Model.........................m7i

Testbed.......................vino

Host fen:re0

Osname......................JunOS

Osver.......................9.2

Ip..........................192.168.183.219/25

Loopip......................10.255.183.219

Loopipv6....................ABCD::10:255:183:219

Conip.......................192.168.153.174

ge-0/0/2 hunan ge-0/0/1

ge-0/0/3 vinegar ge-2/2/3

This gives you interface, IP information, and much more detail about the device.

1. Now that you have checked out the device, you can access the device. You can use telnet, Putty, SSH, or similar application to connect.

**telnet <name of device>** Accessing the mgmt port

**telnet <name of device>-con** Accessing the console port

Most systems should have your user-id and password. If not, you can try the following:

User-id: **regress**, Password: **MaRtInI**  (or) User-id: **root**, Password: **Embe1mpls**

* Exercise 4 – Checking in Lab Devices

In this exercise, you will check in a lab device back into the pool of available resources.

1. On the build server, you will use **res checkin | ci** to check in devices.

**res ci <name of device>**  (or for numerous devices) **res ci –w bind\_id <bind\_id number>**

[patricks@svl-junos-pool44 /c/patricks/pat-main/ship]$ **res ci taj**

Check in by (id, bind\_id, host, test bed, lab): [host]

id bind name from to purpose

398704 398704 taj 2011-01-23 20:43 2011-01-23 22:43 scripting

Check in id 398704? [y] **y**

Reservation Request Completed in 14.642328 seconds.

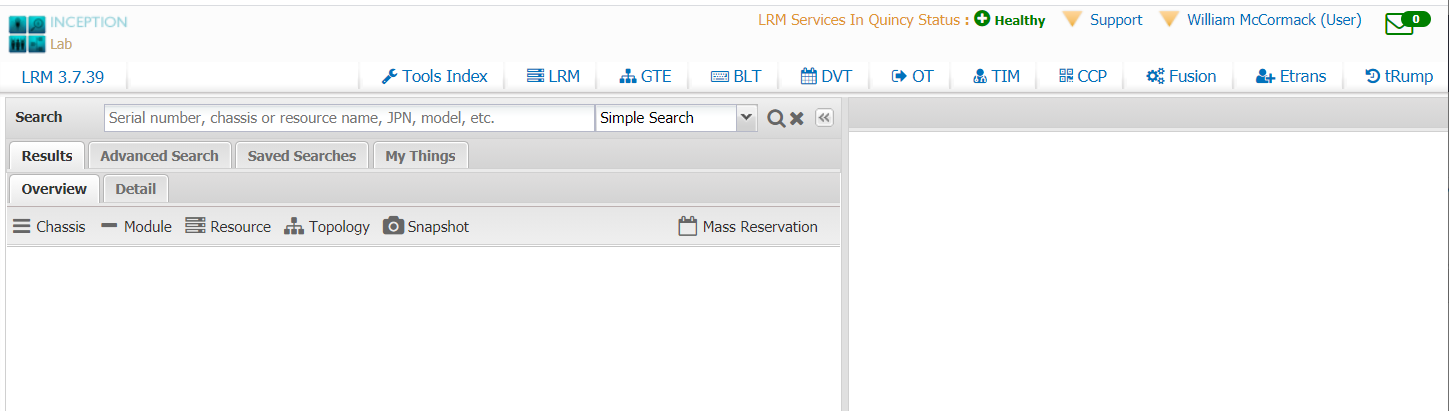
An email will be sent to you making you aware your reservation has been cancelled.

Note: Please ensure lab devices are accessible after usage (both management and console ports). If there is an issue with a lab device, please contact the lab-trolls with the URL provided in class.

* Exercise 5 – Access Lab Resource Manager (LRM) via a Browser

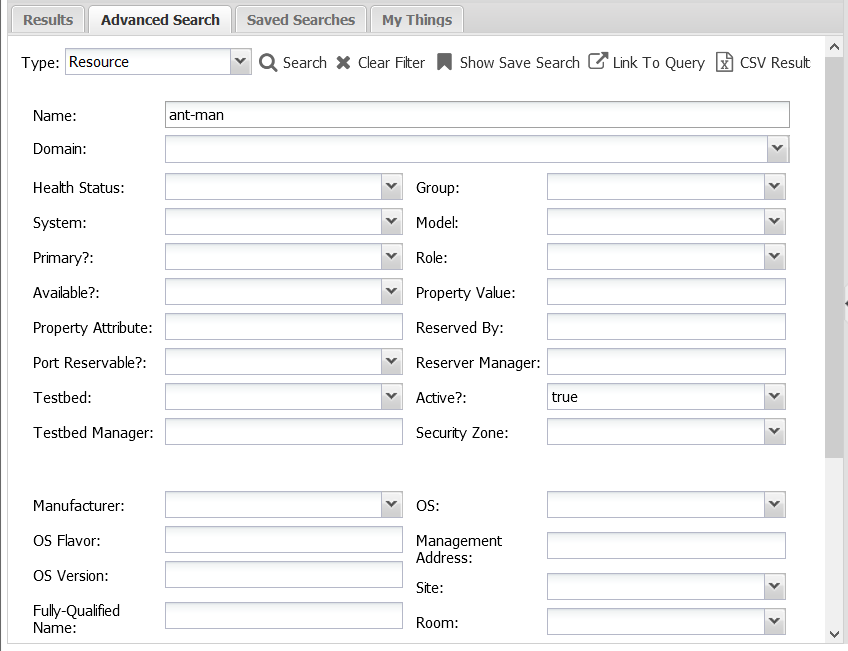
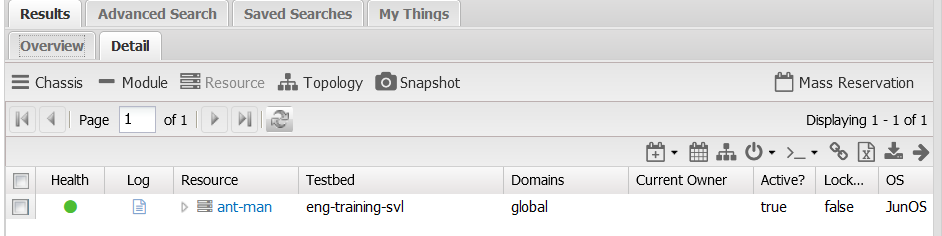
In this exercise, you will use LRM to search and manage lab devices.

1. Open a browser and connect to: <https://inception.juniper.net/lrm>
2. Log in using your Windows login and password.



* Exercise 6 – Use LRM to Search for Devices

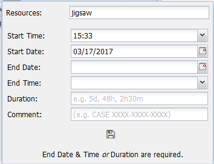
In this exercise, you will use LRM to search for devices.

1. Open the **Advanced Search** tab, then type the name of the router you were assigned by your instructor.  
   
2. Click the *magnify glass icon * on the Search line. After this is done at the bottom of the screen, the router should appear in the Results window.  
   
3. Let’s do another **Advanced Search** query. Click the **Advanced Search** tab and then click the **Clear Filter** button.
4. In the **Testbed** box, enter **eng-training-svl** and then lick the *magnifying glass icon*. A number of devices in the testbed will display.
5. Let’s do one last query in the **Advanced Search** box. Click **Clear Filter**, and then enter **ptx3000** in the **Model** box. Note the box marked **Active**. The value is set to “true” by default.
6. Click the *magnify glass icon * A number of available *ptx3000* devices will display.
7. To find out more details about a device, click on the name of a device in the **Resource** column.

* Exercise 7 – Use LRM to Reserve a Device

In this exercise, you will use LRM to reserve a device.

1. In *Advanced Search*, clear the filter and then type the *name of the router* you were assigned and click the *magnify glass icon *. After this is done at the bottom of the screen, the router should appear in the Results window.
2. To reserve a device, in the Results window, add a check to the box for device name. The icon in the Results window click the down arrow and the below menu will appear.
3. In Duration field, type ***2hr*** and click the *save icon* to make the reservation.



1. To verify on the left side of the screen, you should see ***My Things*** line. Click on double arrow on that line. You should see the device you have reserved in the My Reservations Tab.

* Exercise 8 – Copying Builds/Files to Lab Devices

In this exercise, you will copy builds/files to the lab devices. As with programming, there are a number of ways of accomplishing this task. We will show you a couple of methods.

Decide the file or files you need to send to the lab device.

1. Move **cd** to the specific directory of where these files are located.
2. Type **ls** to ensure the files you want to send are located here.
3. Log into the lab device via **telnet <name of device>**. User-id: ***regress***  Password: ***MaRtInI***
4. Perform a secure copy **scp** command to copy the files from your build location to the lab device.

**scp <from-file-location> <userid>@<to-host>:<to-file-location>**

% **scp jinstall-15.1I20160520\_2103\_patricks-domestic.tgz regress@atom-girl:/var/home/regress/.**

1. Another method would be to File Transfer Protocol (FTP) the files over to the lab device.

* You have moved (**cd**) to the directory to send the files.
* Start FTP with the IP of the lab device using your user-id and password:

**ftp <TCP/IP Address>** (for example: ftp 192.168.48.35)

Connected to 192.168.48.35.

220 taj FTP server (Version 6.00LS) ready.

Name (192.168.48.35:patricks):

331 Password required for patricks.

Password:

230 User remote logged in.

Remote system type is UNIX.

Using binary mode to transfer files.

* Copy the files you want to transfer>

**put <file name>**

ftp> **put jroute-11.1I20110118\_2053\_patricks.tgz**

This will copy files to your home directory on the lab device. If that is not the location you want to copy, simply use the cd command to change the remote directory location before issuing the **put** command. To exit FTP, you issue the **quit** command.

* Exercise 9 – Install the New Package or Build on Lab Device

In this exercise, you will install a package or new build on a lab device.

1. On the lab device, go into the CLI operational mode. If you are in the shell, you need to type: **cli**

--- JUNOS 15.1R3.6 built 2016-03-24 18:39:40 UTC

% cli

regress@ariel>

1. Issue the **request system software add** command to update the lab device software. There are a number of options when updating system software.

patricks@taj> **request system software add ?**

Possible completions:

<package-name> URL or pathname of package

best-effort-load Load succeeds if at least one statement is valid

delay-restart Don't restart processes

force Force addition of package (ignore warnings)

jdocs-11.1I20110118\_2053\_patricks.tgz Size: 3268532, Last Changed

jroute-11.1I20110118\_2053\_patricks.tgz Size: 25544383, Last Changed

no-copy Don't save copies of package files

no-validate Don't check compatibility with current configuration

reboot Reboot system after adding package

For Example:

patricks@taj> **request system software add jroute-11.1I20110118\_2053\_patricks.tgz reboot**

You may request a reboot as well as many more options (use **?**). This upgrading software operation may take **15** minutes to complete.

1. To check to see when it is complete, you can ping the system name or TCP/IP Address.
2. Once the system is back online, telnet back into the device. Notice the banner which should show the new version of Junos you are running.

* Exercise 10 – Extra Credit

In this exercise, you will revert the device to the previous OS.

1. On the lab device, go into the CLI operational mode. We will move to the previous OS version by typing the following:

patricks@taj> **request system software rollback**

After this command, it should ask if you want to move to this OS, you will need to reboot the device.

patricks@taj> **request system reboot**

*After this command, you will be bumped off the system and you can see when it is completed by*

*pinging the system.*

# Lab 7: Creating a Virtual PTX and MX

7

Introduction

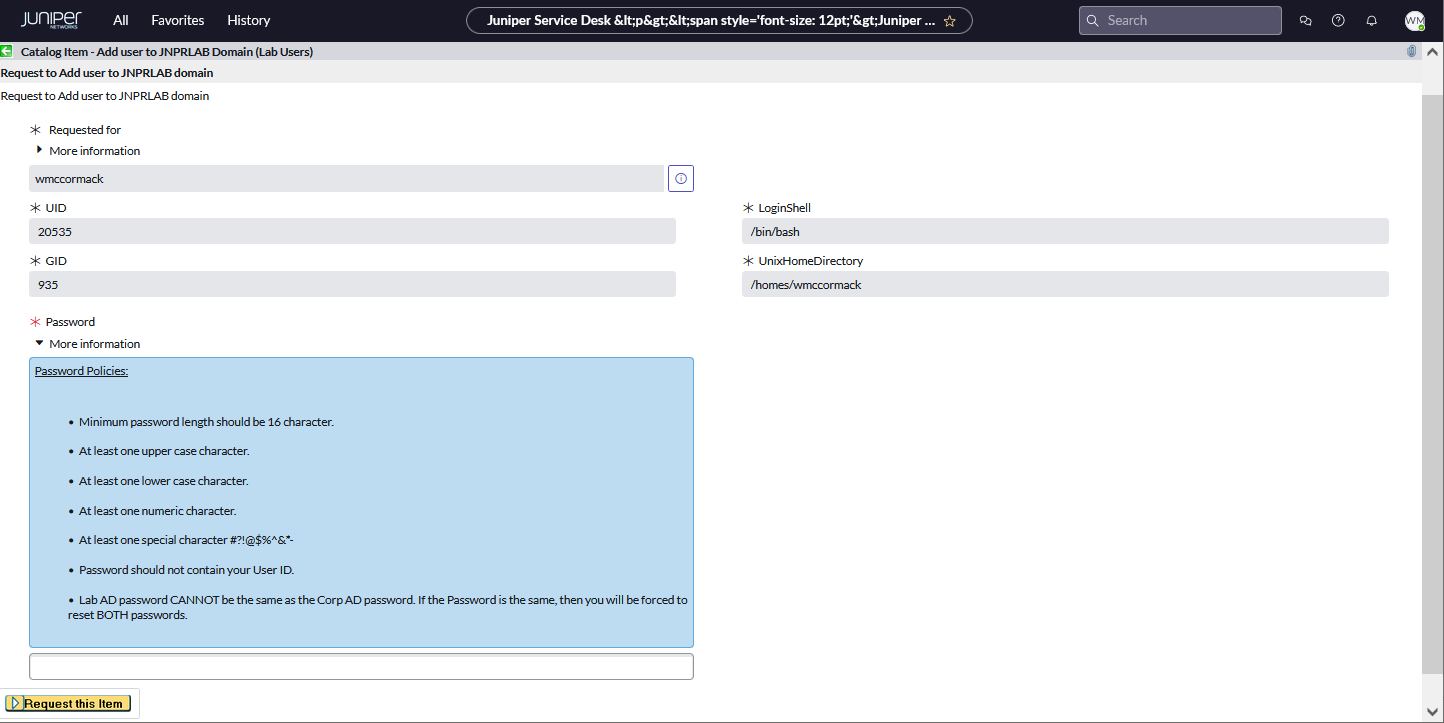
In this lab, you will create a vPTX and vMX from an image published by the build and release team. Once you have created your virtual devices, we will connect to your vPTX and execute some CLI commands to understand better how to view and manage the device and Evo.

Module Objectives

* Create and access a VMM account
* Copy a vPTX image into your VMM space, then create a VMM config file
* Create a vPTX and a vMX
* Watch the virtual devices boot up
* Start a CLI session with your vPTX and test it by issuing CLI commands
* Exercise 1 – Connect to a VMM POD server from any Test Shell Server

In this exercise, you get yourself VMM pod access. Next, you will log into a BaaS shell server, and from there you will SSH into a VMM POD server.

1. Open a Helpdesk ticket and get instant, automated access to the VMM servers. Provide a new password (not your corporate password) and then submit. This password will be used for connecting to the VMM servers. If you already have VMM access, proceed to Step 2.

[**https://junipernetworks.service-now.com/now/nav/ui/classic/params/target/com.glideapp.servicecatalog\_cat\_item\_view.do%3Fv%3D1%26sysparm\_id%3D71e7e8d2db78c1d0ef0793b7f4961984**](https://junipernetworks.service-now.com/now/nav/ui/classic/params/target/com.glideapp.servicecatalog_cat_item_view.do%3Fv%3D1%26sysparm_id%3D71e7e8d2db78c1d0ef0793b7f4961984) ****

Enter a password here and click “Request this Item

Once you have submitted, access will be instantly granted. Make sure you write down your new password.

1. Next, use an SSH tool such as a terminal window, or PuTTY to login to a BaaS server. You will use a BaaS server to SSH to a VMM server.

**bng-baas-shell.juniper.net (=Bangalore) OR**

**qnc-baas-shell.juniper.net (= Quincy, Washington)**

1. From the BaaS server, now access a VMM pod by SSH-ing to one of the following POD servers. Enter a command such as: **ssh wmccormack@q-pod22-vmm**. You will be prompted to provide your new password that you created in Step 1.

|  |  |
| --- | --- |
| **Location** | **Server Name** |
| Quincy | q-pod05-vmm, q-pod08-vmm, q-pod13-vmm, q-pod21-vmm, q-pod22-vmm, q-pod26-vmm |
| Bangalore | elpod1-vmm, enpod3-vmm, enpod4-vmm |
| Sunnyvale | sv8-pod1-vmm, sv8-pod3-vmm |

Exercise 2 – Change Directories to /vmm/data/user\_disks/<username>

You will now move to the directory where you will spin up your virtual devices.

1. Type **pwd**.

You should now be in the **/homes/<username>** directory.

1. **cd**to***/vmm/data/user\_disks/<username>*** This is your space for creating virtual devices.
2. Type **mkdir vPTX** to create a new directory where we will put the image and configuration files.
3. Type **cd vPTX**. Our next step will be to get an image file.

* Exercise 3 - Open a New SSH Session to BaaS

To learn about the images are available to you, browse the image directories from our BaaS shell. We need to use BaaS because your VMM account does not have access to the images.

1. Open a new terminal window, or new PuTTY session to login to a BaaS server.

**bng-baas-shell.juniper.net (=Bangalore) OR**

**qnc-baas-shell.juniper.net (= Quincy, Washington)**

1. Type **cd** **volume/evoimages/release/evo/rel**
2. Type **ls** to view the directories.
3. Change directories to **volume/evoimages/release/evo/rel/23.2R1-S1/rel\_23.2R1-S1.8**.
4. Type **ls** to view **vJunosEvolved-23.2R1-S1.8-EVO.iso**, which is the image we will use to create our virtual device.

* Exercise 4 - Switch Back to Your VMM Shell and Copy an Image File into It

1. Switch back to your VMM shell and verify you are in your **vPTX** directory.
2. Using **SCP**, copy an image file into your vPTX directory:  
   **scp** [**<username>@ttqc-shell002**](mailto:rharwood@ttqc-shell002)**:/volume/evoimages/release/evo/rel/23.2R1-S1/rel\_23.2R1-S1.8/vJunosEvolved-23.2R1-S1.8-EVO.iso .** (Don't forget the final period which means "here.") Because your VMM account does not allow direct access to the images, we use our BaaS account to send the files to our VMM account.
3. After the transfer is completed, type **ls** to see the transferred file.

* Exercise 5 - Create a Configuration File

For this next step, we will create a VMM configuration file. This file specifies what hosts to create. Here we will be creating a vPTX and a vMX.

1. Use an editor to create a file with the name **<name>.conf**. For example, vptx.conf.
2. Copy and paste the following text in your **<name>.conf** file. Additionally, where highlighted below, add your username and vPTX directory name.

#include "/vmm/bin/common.defs"

#define VJUNOSEVOLVED\_PHASE 2

#include "/vmm/data/user\_disks/vmxc/common.vmx.p2.defs"

#include "/vmm/data/user\_disks/vptxc/common.vJunosEvo.ptx.defs"

#include "/vmm/data/user\_disks/vptxc/common.vJunosEvo.defs"

#define EVOVPTX\_DISK1 "/vmm/data/user\_disks**/<your username>/vPTX**/vJunosEvolved-23.2R1-S1.8-EVO.iso"

#define VMX1\_IMG \

"/vmm/data/base\_disks/default\_images/default\_image\_vmx\_phase2.img"

TOPOLOGY\_START(config)

#undef VMX\_CHASSIS\_NAME

#define VMX\_CHASSIS\_NAME vMX1

#undef VMX\_CHASSIS\_I2CID

#define VMX\_CHASSIS\_I2CID 21 //\*\*\*\* MX960 Chassis\_1

#undef VMX\_DISK

#define VMX\_DISK basedisk VMX1\_IMG;

VMX\_CHASSIS\_START()

VMX\_RE\_START(vmx1, 0)

VMX\_RE\_INSTANCE(vmx1, VMX\_DISK, VMX\_RE\_I2CID, 0)

install "/vmm/data/user\_disks/evo\_test/EVOvPTX/pbuilder\_reference/vmx1.conf" "/root/olive.conf";

VMX\_RE\_END

VMX\_MPC\_START(vmx1\_mpc, 0)

VMX\_MPC\_INSTANCE(vmx1\_mpc, VMX\_DISK, VMX\_MPC\_I2CID, 0)

VMX\_CONNECT(GE(0, 0, 0), private10)

VMX\_MPC\_END

VMX\_CHASSIS\_END

#undef PTX\_CHAS\_NAME

#define PTX\_CHAS\_NAME vEVO

#undef CHANNELIZED

#define CHANNELIZED no

EVOVPTX\_CHASSIS\_START\_ (PTX\_CHAS\_NAME)

vJunosEvoRE(PTX\_CHAS\_NAME,EVOVPTX\_DISK1,CHANNELIZED)

EVOVPTX\_CONNECT(IF\_ET (0, 0, 0), private10)

EVOVPTX\_CONNECT(IF\_ET (0, 0, 1), private11)

vJunosEvoRE\_END

EVOVPTX\_CHASSIS\_END\_

#undef VMX\_CHASSIS\_NAME

#define VMX\_CHASSIS\_NAME vMX2

#undef VMX\_CHASSIS\_I2CID

#define VMX\_CHASSIS\_I2CID 21 //\*\*\*\* MX960 Chassis\_1

#undef VMX\_DISK

#define VMX\_DISK basedisk VMX1\_IMG;

VMX\_CHASSIS\_START()

VMX\_RE\_START(vmx2, 0)

VMX\_RE\_INSTANCE(vmx2, VMX\_DISK, VMX\_RE\_I2CID, 0)

install "/vmm/data/user\_disks/evo\_test/EVOvPTX/pbuilder\_reference/vmx1.conf" "/root/olive.conf";

VMX\_RE\_END

VMX\_MPC\_START(vmx2\_mpc, 0)

VMX\_MPC\_INSTANCE(vmx2\_mpc, VMX\_DISK, VMX\_MPC\_I2CID, 0)

VMX\_CONNECT(GE(0, 0, 0), private11)

VMX\_MPC\_END

VMX\_CHASSIS\_END

PRIVATE\_BRIDGES

TOPOLOGY\_END

1. Where highlighted above, add your username and vPTX subdirectory.
2. Save and close the file.

* Exercise 6 – Create Your Virtual Devices

In this exercise, you create a virtual PTX and a virtual MX and watch them boot up. The bootup process takes a while – like 10 to 20 minutes. Be patient.

1. From within your vPTX directory, type the following command:   
   **vmm config <config file> -g vmm-default**
2. The command should take a few seconds to run. After you get a prompt back, type **vmm ls** to see your devices. Note that your vPTX is called vEVO-RE0 and that the devices are not actually running yet.
3. Type **vmm bind** to bind IP addresses to your devices.
4. Type **vmm ip** to learn the IP addresses of your devices.
5. Type **vmm start** to start your devices.
6. Type **ping <IP address>** to see if your device is reachable (it will not be.) Repeat this command in the future to determine the status of the device(s).
7. Type **q** to access the console port of your virtual PTX and watch it boot up.
8. Once boot up completes, you can log into the device(s) and issue some CLI commands. To login, provide the following credentials:

Login: **regress**

Password: **MaRtInI**

* Exercise 7 – Execute some CLI commands on your vPTX

Once your device has booted up, you will be able to log into it and execute some CLI commands.

1. Upon logging in, you will be in the Linux shell. You can verify this by typing **pwd** or any other Linux command. A # prompt indicates that you are in the Linux shell.
2. Move into the CLI, by typing the following:

**$ cli**

1. To see command options, type a ? after part of the command. For example, type:

**> show chassis ?**

regress@vEVO-RE0> show chassis ?

Possible completions:

alarms Show alarm status

beacon Show beacon status

craft-interface Show craft interface status

environment Show component status and temperature, cooling system speeds

ethernet-switch Show Ethernet switch information

fabric Show internal fabric management state

fan Show fan and fan tray information

firmware Show firmware and operating system version for components

fpc Show Flexible PIC Concentrator status

fru Show fru replacement status

hardware Show installed hardware components

Sample output.

1. To view the software version, type the following:

**> show version**

Hostname: vEVO-RE0

Model: ptx10001-36mr

Junos: 23.2R1-S1.8-EVO

Yocto: 3.0.2

Linux Kernel: 5.2.60-yocto-standard-g12d8464

JUNOS-EVO OS 64-bit [junos-evo-install-ptx-fixed-x86-64-23.2R1-S1.8-EVOI20230913034619-evo-builder-1]

Sample output.

1. To see the current active configuration, type the following

**> show configuration**

regress@vEVO-RE0> show configuration

## Last commit: 2023-09-14 11:09:56 PDT by root

version 23.2R1-S1.8-EVO;

groups {

global {

system {

host-name vEVO-RE0;

root-authentication {

encrypted-password "$1$ZUlES4dp$OUwWo1g7cLoV/aMWpHUnC/"; ## SECRET-DATA

}

login {

class wheel {

permissions [ admin clear field floppy interface maintenance network reset routing shell snmp system trace view ];

}

class readonly {

permissions [ interface network routing system trace view ];

}

Sample output showing current configuration.

1. To see system storage information, type the following:

**> show system storage**

regress@vEVO-RE0> show system storage

re0:

--------------------------------------------------------------------------

Filesystem Size Used Avail Capacity Mounted on

/dev/root 33M 33M 0 100% /run/initramfs

/dev/vda2 16G 1.4G 14G 10% /soft

/dev/vda5 3.0G 60M 2.7G 3% /data

/dev/vda7 16G 1.2G 14G 8% /var

/dev/loop0 2.3G 7.1M 2.2G 1% /data/var/external

Sample output.

1. To see system statistics, type the following:   
   > **show system statistics**

regress@vEVO-RE0> show system statistics

Tcp:

0 packets sent

0 window probe packets

0 packets received

0 discarded for bad checksums

0 discarded for bad header offset fields

Sample output.

1. To get help for a command, the question mark (?) will be your friend. Type the following:  
   > **show ?**

Possible completions:

accounting Show accounting profiles and records

agent Show SDN agent information

app-engine Show App-engine information

arp Show system Address Resolution Protocol table entries

as-path Show table of known autonomous system paths

authentication-whitelist Show 802.1X White List MAC addresses

backup-selection Show backup selection policies information

bfd Show Bidirectional Forwarding Detection information

bgp Show Border Gateway Protocol information

captive-portal Show captive portal information

chassis Show chassis information

45632 duplicate acks

0 acks for unsent data/pivot/data/junos-install-ptx-x86-64-16.2I20170417105358\_patricks/dev

/dev/loop2 10M 10M 0 100% /pivot/data/junos-install-ptx-x86-64-16.2I20170417105358\_patricks/perl-5.20.0

/dev/loop3 128K 128K 0 100% /pivot/data/junos-install-ptx-x86-64-16.2I20170417105358\_patricks/evofiller

/dev/loop4 2.2M 2.2M 0 100% /pivot/data/junos-install-ptx-x86-64-16.2I20170417105358\_patricks/modules

/dev/loop5 49M 49M 0 100% /pivot/data/junos-install-ptx-x86-64-16.2I20170417105358\_patricks/python-2.7

/dev/loop6 12M 12M 0 100% /pivot/data/junos-install-ptx-x86-64-16.2I20170417105358\_patricks/python-3.3

/dev/loop7 925M 925M 0 100% /pivot/data/junos-install-ptx-x86-64-16.2I20170417105358\_patricks/ccd-ptx-fpc

/dev/loop8 2.2M 2.2M 0 100% /pivot/data/junos-install-ptx-x86-64-16.2I20170417105358\_patricks/base

Sample output.

1. When done investigating the Evo CLI, Type the following:
2. > **exit**
3. $ **exit**

The vPTX will continually run until the vPTX or POD is brought down. To restart a CLI session, telnet to your device’s IP address. Since DNS service is not enabled, you will not be able to telnet to your device’s host name.

# Lab 8: Searching Source Code Repository

8

Introduction

In this lab, you will use OpenGrok to search for a string within the Evo Repositories.

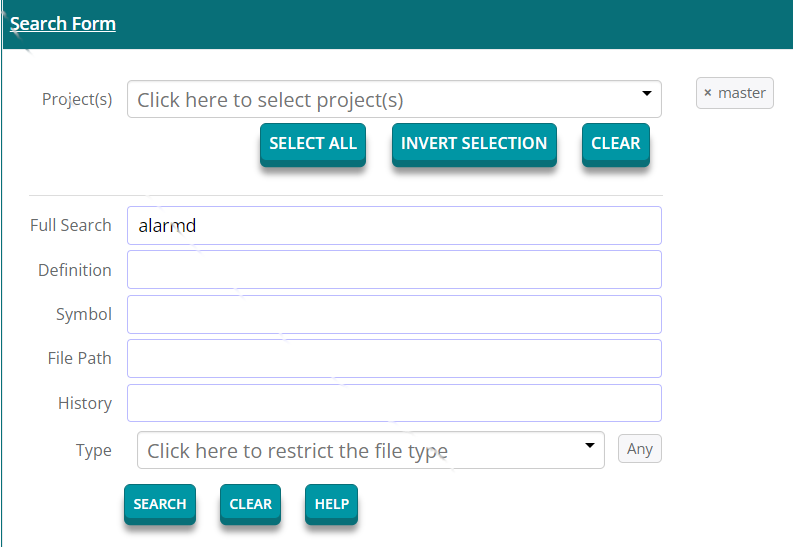
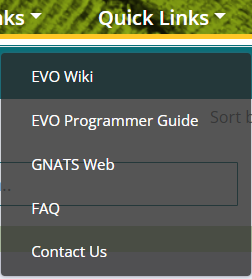
Module Objectives

Covered here:

* You will use OpenGrok to search the Evo repositories
* Install Evo onto a vPTX

Exercise 1 - Search Code in Git Using OPENGROK

**OpenGrok** searches repositories. In this exercise, you search through code in the Git repositories.

1. Open a browser and type the following address:  
   <https://opengrok-evo.juniper.net>
2. Enter your username and password.
3. In the “**In Projects**” menu at the top right of the screen, scroll down and click on: “master.”
4. In the box marked “**Full Search**,” type **alarmd** and then click **Search**.  
   
5. Scroll through the results, open a file of your choice and look at the contents.
6. Click the Quick Links menu option at the top of the page. This option presents some helpful links to other Evo resources.  
   
7. Click the **Evo Wiki** link. This will open a new page in your browser and take you to the Evo Wiki.
8. Take a few minutes to examine the wiki.

# 

9

# Lab 9: Test Code Using Static Analysis (Coverity)

Introduction

In this lab, you learn how to test code using static analysis (Coverity).

Module Objectives

* Run Coverity to perform static analysis
* Use Jenkins and log files to review results
* Analyze and correct detected defects
* Repeat process until “Coverity clean”
* Exercise 1 - Connect to Your Sparse Sandbox

In this exercise, you will run Coverity testing on your **sparse sandbox**.

1. Using an SSH tool (such as a terminal window, or PuTTY), login to one of the build servers (using your corp credentials):   
   bng-baas-shell.juniper.net(Bangalore)qnc-baas-shell.juniper.net(Quincy, Washington)
2. Type baas edit -s <volume-name> to edit your volume.
3. Change directory to your sparse sandbox <SB>.

* Exercise 2 – Edit a Common Daemon

In this exercise, you modify a common daemon to add a “dummy” function.

1. Change to the <SB>/pio/src directory, where the code for our target component is located.
2. Type **vi pio\_mux.c** to open the file in vi.
3. Type :set nu and press Enter to turn on line numbering.
4. Move down to line 47 which has the text **pio\_mux\_dev\_t \*piom\_dev;**
5. Move down one more line to line 48.
6. Type a (add) to add text to the file.
7. Copy the following “dummy” function into line 48.

int \*p1,\*p2,\*ptr;

p1 = malloc(42);

free(p1);

p2 = malloc(42);

free(p2);

ptr = (int \*)malloc(sizeof(int));

\*ptr = 10;

1. Review your work; it should match the bold type shown below.

{

48 pio\_mux\_dev\_t \*piom\_dev;

49 **int \*p1,\*p2,\*ptr;**

50

51 **p1 = malloc(42);**

52 **free(p1);**

53

54 **p2 = malloc(42);**

55 **free(p2);**

56

57 **ptr = (int \*)malloc(sizeof(int));**

58 **\*ptr = 10;**

59

60 for (piom\_dev = pio\_mux\_dev\_head; piom\_dev; piom\_dev = piom\_dev->next) {

61 if (&piom\_dev->mux\_pio\_internal.handle == mux\_pio\_handle) {

62 break;

1. Press <Esc>, then type :wq to save and close the file.

* Exercise 3 – Create a Build to Verify Sandbox Compiles Correctly

In this exercise, you create a build to ensure your sandbox is in good working order and will build successfully. In this case, you build only a single directory.

1. Type baas edit -s <volume> to return to your volume.
2. Change directories to the **<SB>/pio** directory.
3. Type baas build -b "sb make" -l to start the build process. Since you are running the -l option, you will remain in your edit pod and will be able to watch the build progress.
4. Monitor your build’s progress. It should take about 2 to 5 minutes.

**NOTE**: If the build fails, review the **pio\_mux.c** file for errors.

* Exercise 4 – Run Static Analysis Against the Code Changes

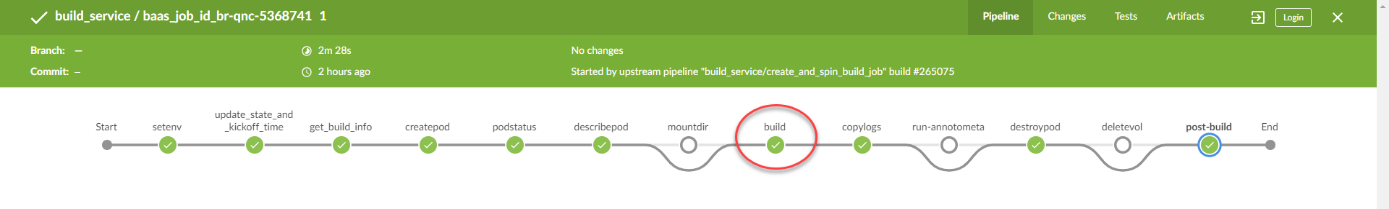
In this exercise, you run static analysis on your code changes using Coverity.

1. Verify that you are in the <SB>/pio directory.
2. Type baas build -b "run-coverity" -x to run static analysis.
3. Go to **https://baas.juniper.net/**, click **My Builds** in the left-side menu, and monitor your build’s progress. The build should take 2 to 5 minutes.

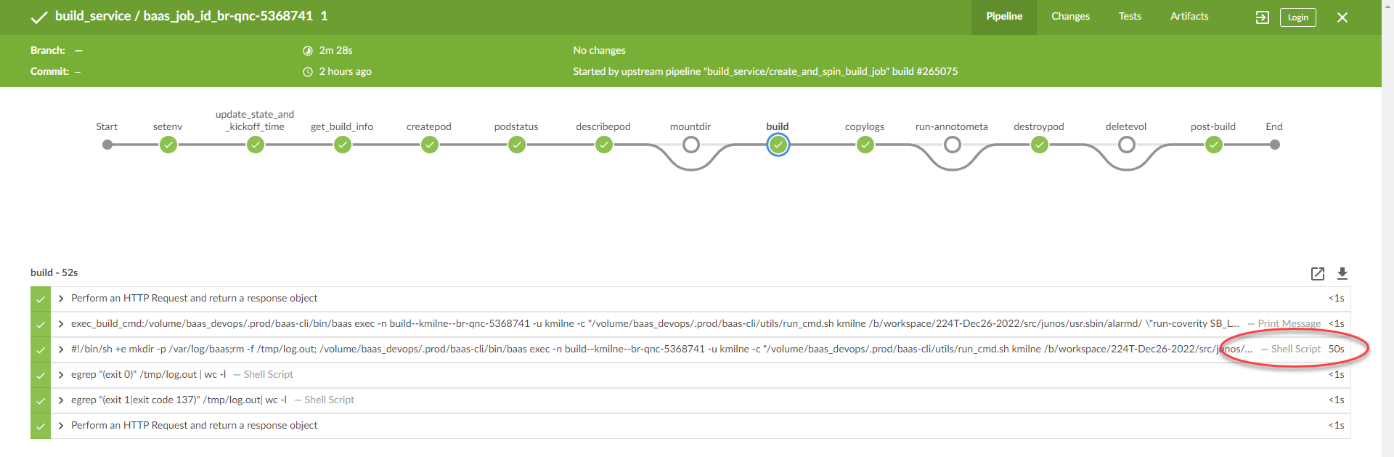
* Exercise 5 – Review Results

In this exercise, you review the static analysis report to learn the results of Coverity testing, and review details about the defects found.

1. When the build completes successfully, return to the My Builds page in baas.juniper.net and refresh the page to show your latest build.
2. Click its Request ID for your latest build.
3. A new tool, Jenkins, opens in a new browser tab. Login using your corporate credentials.
4. In the Jenkins UI, click the **build** stage.



1. In the activity list, click the first **Shell Script** instance.



1. Review the output shown in the drop-down area. Notice there is a Static Analysis Report section indicating there are two defects, as well as a path to a report.

=================================================================

Static Analysis Report

=================================================================

Total of 2 new coverity defects detected

Defects Report :

/b/workspace/<SB>/obj-coverity/cov-logs/pio\_new\_<timestamp>.txt

=================================================================

1. Return to your BaaS shell session and move to the <SB>/obj-coverity/cov-logs directory.
2. Type cat pio\_new\_<timestamp>.txt to view the defects report. Notice that Coverity has detected two types of defects in your **pio\_mux.c** file:

* Wrong size argument (SIZEOF\_MISMATCH).
* Resource leak (RESOURCE\_LEAK).

1. Copy the report contents into a text file for easy reference in the next exercise.

* Exercise 6 – Analyze the Defects

In this exercise, you review your code changes to determine how to address the defects.

1. Change to the <SB>/pio directory.
2. Type vi pio\_mux.c to open the file in vi.
3. Type :set nu and press Enter to turn on line numbering.
4. Move down to line 48, where you added the new code. Review your code changes in relation to the defects report details and:

* Trace through what the code is actually doing (vs. what it’s intended to be doing)
* Attempt to determine whether each defect is a valid bug
* Determine if in fact the defect is a ‘false positive’ (the code is intentionally coded as is) and requires a code annotation.
* Exercise 7 – Fix the Defects

In this exercise, you edit the code to fix the defects.

1. Move down to line 50, a blank line below the line the text . **int \*p1,\*p2,\*ptr;**
2. In vi, type i (insert) to edit the file.
3. For the first defect, in this case the code is intended as is, so has triggered a ‘false positive’. On line 50 (a blank line), add three spaces and then add the following code annotation:

// coverity[suspicious\_sizeof:FALSE]

1. For the second defect, in this case the fix is to add another line of code. On line 59 add a new blank line.
2. On line 59 (now blank), add three spaces and then add the following new line of code:

free(ptr);

1. Review your work; it should match the example shown below.

...

49 int \*p1,\*p2,\*ptr;

50 **// coverity[suspicious\_sizeof:FALSE] //Added this line**

51 p1 = malloc(42);

52 free(p1);

53

54 p2 = malloc(42);

55 free(p2);

56

57 ptr = (int \*)malloc(sizeof(int));

58 \*ptr = 10;

59 **free(ptr); //Inserted this line**

60

...

1. Press <Esc>, then type :wq to save and close the file.

* Exercise 8 – Run Static Analysis Against the Updated Code Changes

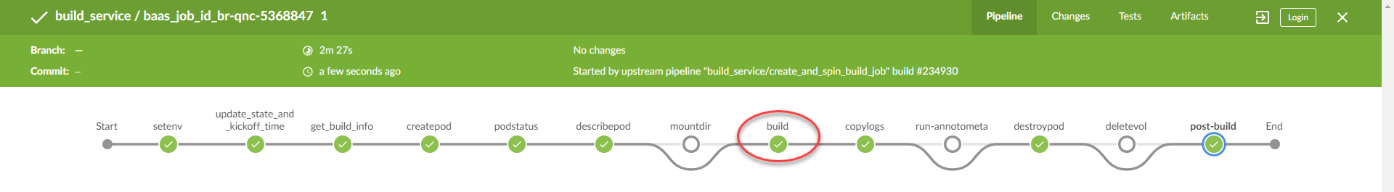
In this exercise, you run static analysis on your code changes using Coverity.

1. Change directories to **<sb>/pio**.
2. Type baas build -b "run-coverity" -x to run static analysis again.
3. Return to **https://baas.juniper.net/**, click **My Builds** in the left-side menu, and monitor your build’s progress. It should take about 5 minutes.

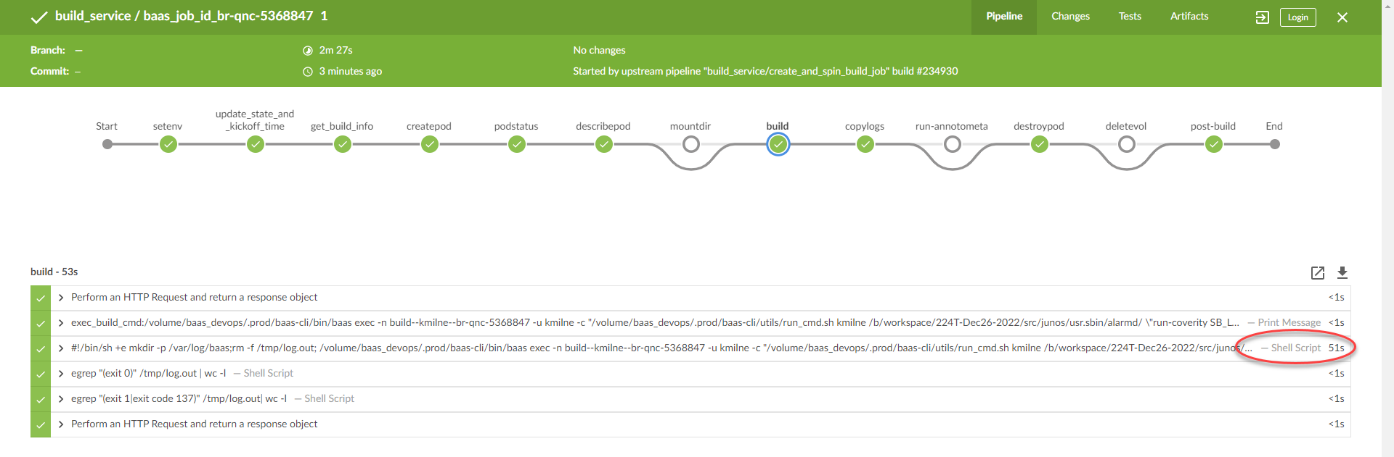
* Exercise 9 – Review Updated Results

In this exercise, you review the second static analysis report to learn the results of Coverity testing, and review details about the defects found.

1. When the build is completed successfully, click its Request ID.
2. If needed, login to Jenkins again using your corporate credentials.
3. In the Jenkins UI, click the build stage.



1. In the activity list, click the first **Shell Script** instance.



1. Review the output shown in the drop-down area. Notice in the section previously called Static Analysis Report there are now no defects found, though there is still a path to a report.

=================================================================

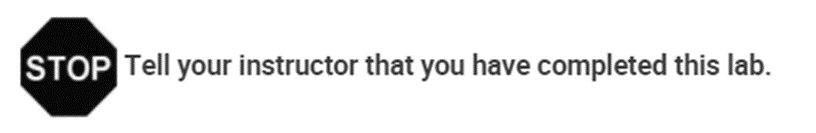
Total of 0 new coverity defects/clang compiler warnings detected

No new defects found. Use -a option with this script to see all defects.

Defects Report :

/b/workspace<SB>log/coverity/cov-logs/pio\_new\_<newer-timestamp>.txt

=================================================================

1. Return to your BaaS shell session, re-enter your volume, and change to the <SB>/obj-coverity/cov-logs directory.
2. Type cat pio\_new\_<newer-timestamp>.txt to view the defects report. Notice that this time Coverity has detected no defects. 

# Lab 10: Code Coverage

10

Introduction

This module shows you how to run code coverage.

Module Objectives

* Create a sparse sandbox
* Make a minor code change
* Create a diff
* Run a code coverage build
* Register your sandbox
* Generate a code coverage report
* Exercise 1 – Create a Volume and a Sparse Sandbox

In this exercise, you create a volume and a sparse sandbox.

1. Run **baas createvol -s <volumename>** **-z 1500Gi** to create a volume.
2. Edit the volume: **baas edit -s <volumename>**.
3. Checkout a sparse sandbox: **sb create -p master -n <sb name>**The project master will create some tools and little else. This command will take about 13 minutes to run.
4. Change directories to your sandbox: **cd <sb name>**
5. Type **ls** to see what is in your sandbox.
6. Add a component to your sandbox: **sb add forwarding/exprplus**This command takes about 1-2 minutes to run.

* Exercise 2 – Edit the File *JexprDfwManager.cpp*

In this exercise, you will add two spaces after a line in one of the source files.

1. Change directories: **cd forwarding/exprplus/applications/common/dfw/src**
2. **vi JexprDfwManager.cpp**
3. Set numbering: **:set nu**
4. Move to line 180: **:180**
5. Add two spaces after the text on line 180: **$a<space><space>**
6. Save and close the file: **:wq**

* Exercise 3 – Create a Diff of Your Code Change

Create a diff of the changes you just made.

1. Change directories to the root of your sandbox: **cd /baas/<your username>/<vol name>/<sb>**
2. Create a diff: **sb repo forall -v -p -c 'git diff -U15' > coverage.diff**   
   This command should take 1-2 minutes to run.

* Exercise 3 – Create a Code Coverage Build

Run the following to generate a code coverage build.

1. **baas build -b "sb make DEFAULT\_COMPONENTS=ptx-chassis-image DEBUG=0 COVERAGE=1 UT\_FAIL\_IGNORE=1" -x**
2. Open <https://baas.juniper.net> and go to My Builds.
3. Periodically hit **Refresh** to see the latest status of your build. The build will take about 15 minutes.

* Register Your Sandbox with the Code Coverage Portal

1. Type **pwd** to make sure you are in the root directory of your sandbox.
2. Run the following command: **/volume/labtools/bin/ccc/ccp\_sandbox\_register**
3. Take note of the messaging that follows. It includes the command you will need to run the code coverage report.

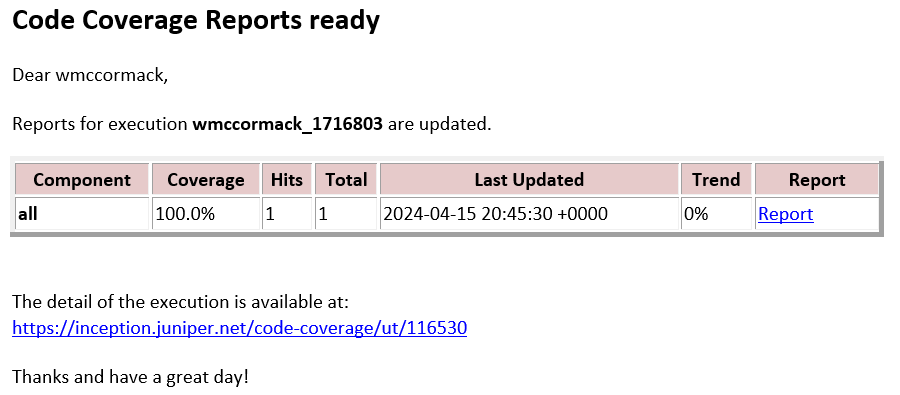
* Run a Code Coverage Report

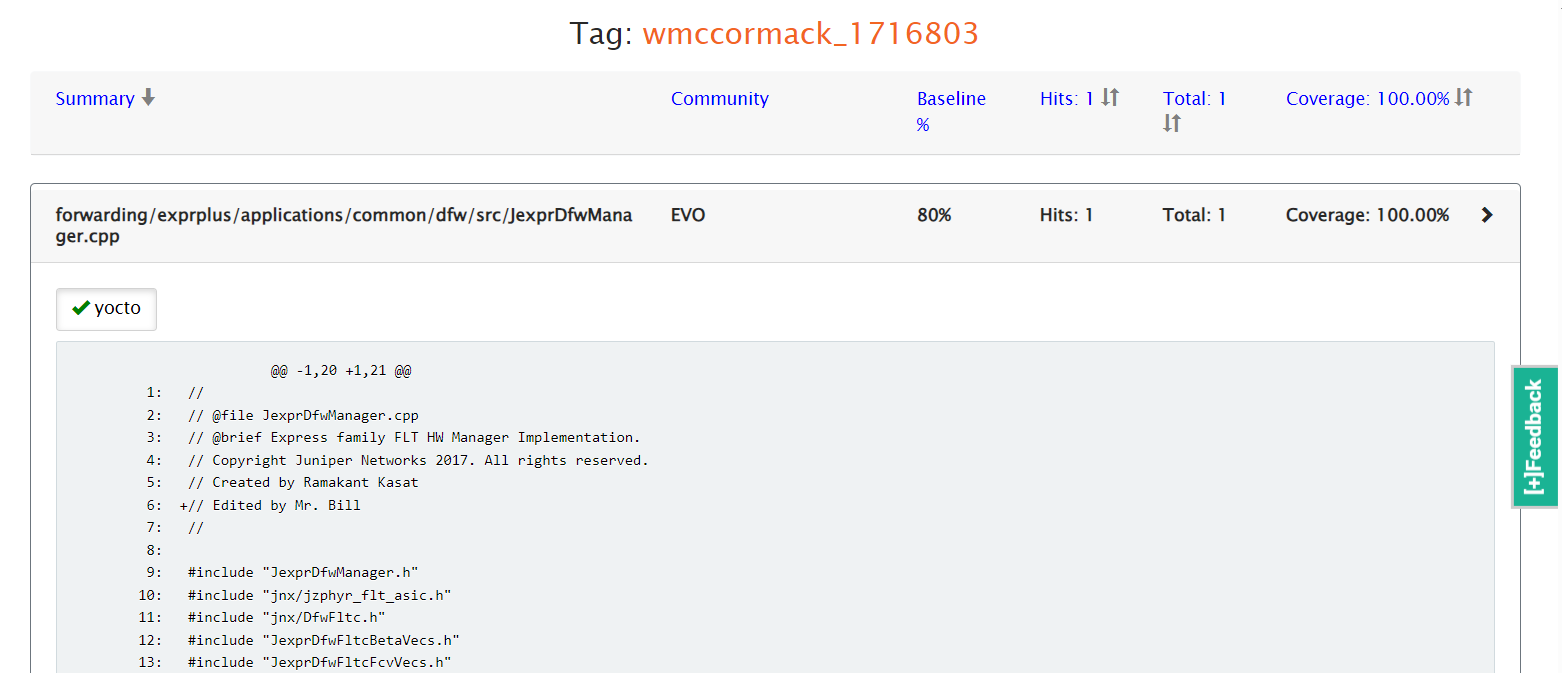
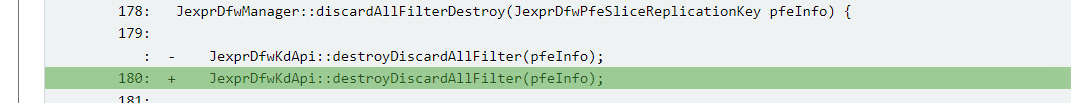
Using the command syntax outputted from the ccp\_sandbox\_register command shown above, run a report to see how thorough your code coverage testing was.

The syntax you will use is:

/volume/labtools/bin/ccc/code\_coverage -host -activity\_type ut -pr <pr\_number> -gcov\_sb\_host <gcov\_sb\_host> -gcov\_sb\_path <gcov\_sb\_host>

1. Run the Code Coverage report command.   
   EXAMPLE:   
   **/volume/labtools/bin/ccc/code\_coverage -host -activity\_type ut -pr 1716803 -gcov\_sb\_host wmccormack-evocc -gcov\_sb\_path /baas/wmccormack/evocc/mastersb**
2. This command will take about 1-2 minutes to run.
3. Wait another 5 minutes to get an email with the following message:



1. Click Report to see details of your code coverage.  
     
   
2. Click the **>** button on the far right (circled above) to see your modified file.  
   
3. Scroll down to see line 180.  
   

11

# Lab 11: Code Review with Gerrit

Introduction

This module shows you how to submit code for review and viewing your code review with Gerrit.

Module Objectives

* Create a new sandbox
* Create a changeset to gather modifications to your sandbox
* Create a code change and commit it to your sandbox
* Upload a code change for review and validation
* Use Gerrit to view your code review
* Exercise 1 – Edit Your Full Evo Sandbox

You will be working with the **full** sandbox that you created based on the private branch, **evo-dev-training**. Do not use your sparse sandbox.

1. Run **baas edit -s <volumename>** to access the volume holding your full sandbox.
2. Change directories to your sandbox directory.

* Exercise 2 – Create a changeset

In this exercise, you create a new file in your sandbox that will be used for our code review. However, before we start, we will create a changeset. Step 2 allows you to create a change-set. This step will not be needed outside of our training environment.

1. **cd <sb>/csets/evo-csets**
2. Enter the following two commands. These are needed for our lab environment.  
      
   **git config --local branch.\_\_no\_cset\_active.remote origin  
   git config --local branch.\_\_no\_cset\_active.merge refs/heads/evo/dev/training**
3. Change directories to **<sandbox>/evo-dev-training**.
4. To see available options with creating a changeset, type: **sb cset create –h**

usage: sb cset create [-h] -t TOPIC -p PR [-R RESET\_PRS]

optional arguments:

-h, --help show this help message and exit

-t TOPIC, --topic TOPIC

One line topic for change set

-p PR, --pr PR comma separated list of PRs

-R RESET\_PRS, --reset\_prs RESET\_PRS

Force overwrite the PRs associated with outstanding

reviews from same SB.

1. Now create a changeset. The topic (**-t**) should include your name, and the PR (**-p**) should be the PR number you created earlier:  
   $ **sb cset create –t “<userid>-file” –p <PR number>**

**$ sb cset create -t "freds-file" -p 1382456**

INFO: Creating new cset...

INFO: Enforcing PRWF state checks for csets/evo-csets.

Created cset freds-file-1382456-2864c7f27 https://svl-evogit-01.juniper.net/93318

All subsequent commits in the sandbox will be associated with this cset.

To deactivate the cset, run "sb cset deactivate"

Sample Output. The topic is built by the “text you provide”-<PR Number>-<hash>.

1. To view your changesets, you can type: $ **sb cset info**  (or) $ **sb cset list**

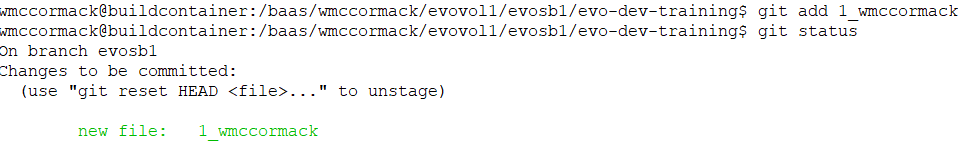
* Exercise 3 – Create a New File

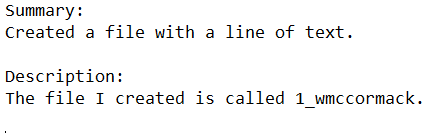
In this exercise, you create a new file in your sandbox that will be used for our code review.

1. Type **pwd** to verify that you are in **<sandbox>/evo-dev-training**.
2. Use an editor such as vi to create a file called **1\_<your-userid>**. For example, **1\_wmccormack**.
3. Add some text to the file and save it.

* Exercise 4 – Git commands to commit your change locally

In this exercise, you will add a file to your local Git repository (in your sandbox) by using the **git add** and **git commit** commands.

1. Run the following command to identify your new file as a candidate for committing.  
   **git add 1\_<your-userid>**
2. To validate your add command, type:  **git status** *(it should show something is staged).*****
3. Type **git commit** to commit your change(s) to your local repository.
4. After entering **git commit**, an editor opens. Scroll the bottom of the text and create a summary and description like the following. Be sure to include a blank line below each section.



1. Type **:wq** to quit the editor.
2. To see your local commit, type: **sb cset info**

Topic : pats-file1-1292384-38569d15a

State : LOCAL

Cset URL : https://svl-evogit-01.juniper.net/33304

Sandbox : svl-evodev-patricks:/build/home/patricks/next-try

Changes : evo-dev-training 973865103a0c2727d25e15b1d2e12730afdc699c

I am doing the initial add of 1\_pats file.

Sample Output

* Exercise 5 – Upload your changes for Code Review

In this exercise, you upload your changes so that they can be validated and reviewed.

1. Examine the description file by entering the command:  
   **sb upload -p <PR Number>**

INFO: Uploading change sets pats-file1-1292384-38569d15a

INFO: Checking cset dependencies

INFO: Uploading 973865103a0c2727d25e15b1d2e12730afdc699c --> evo-dev-training

INFO: Successfully uploaded / updated following URLs

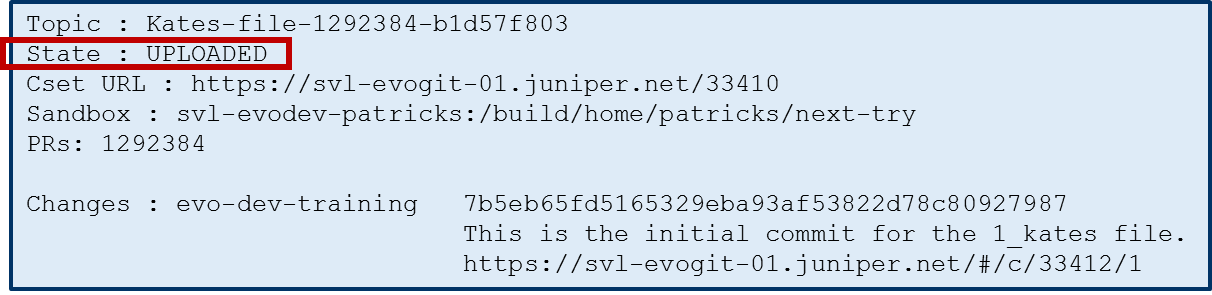
INFO: Uploaded : https://svl-evogit-01.juniper.net/33305

INFO: Trying to set default reviewers...

INFO: Setting review group for https://svl-evogit-01.juniper.net/33305 to eng-training-team-code-review

Sample Output

1. Note: In the output of “sb upload,” copy or jot down the URL in “Uploaded :”. You will use this later in the exercise.
2. To validate you have uploaded your changes, type: **sb cset info**



Sample Output

# Lab 12: Commit Code Using PCT

12

Introduction

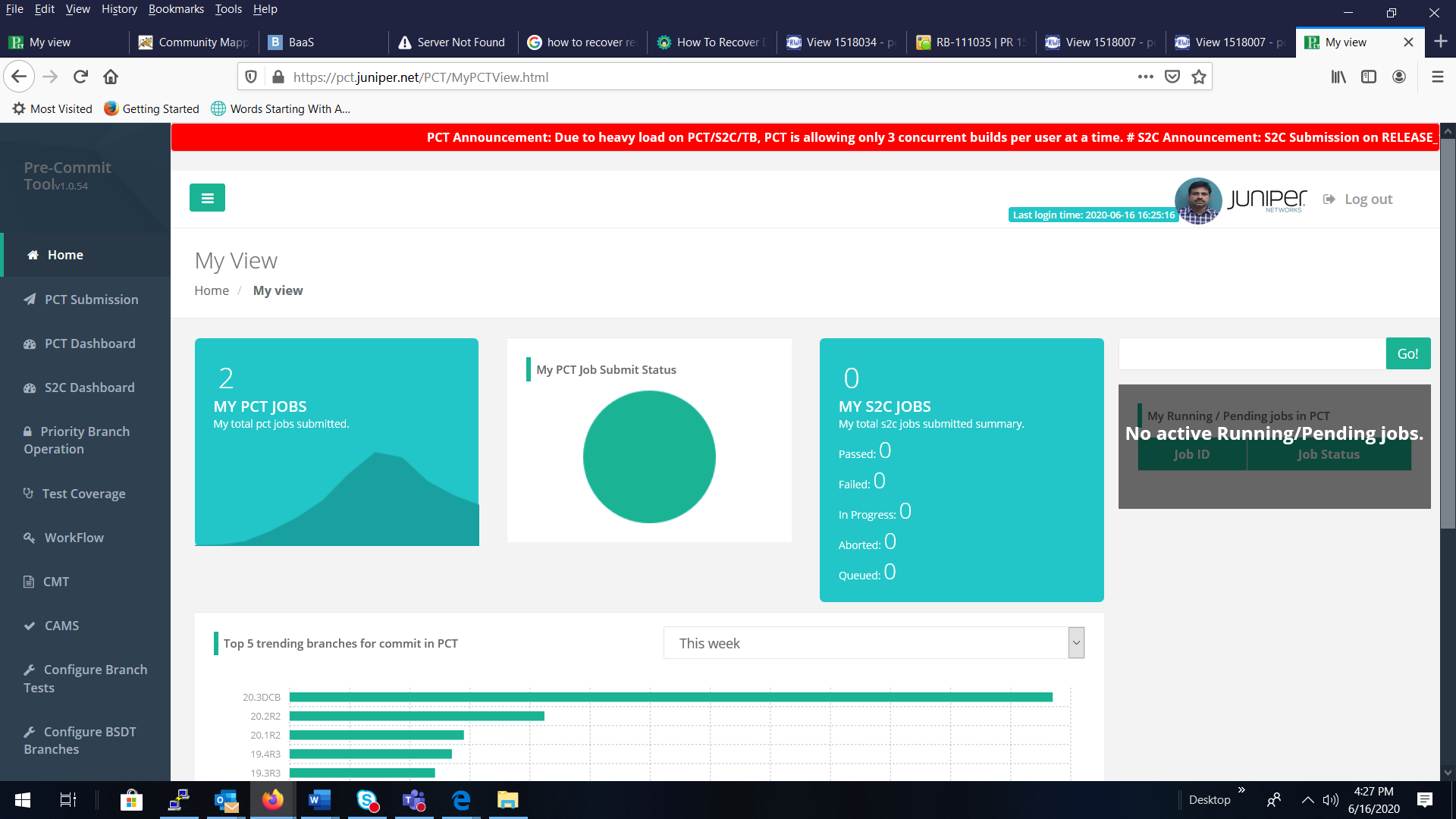
This module shows you how to commit code using PCT.

Module Objectives

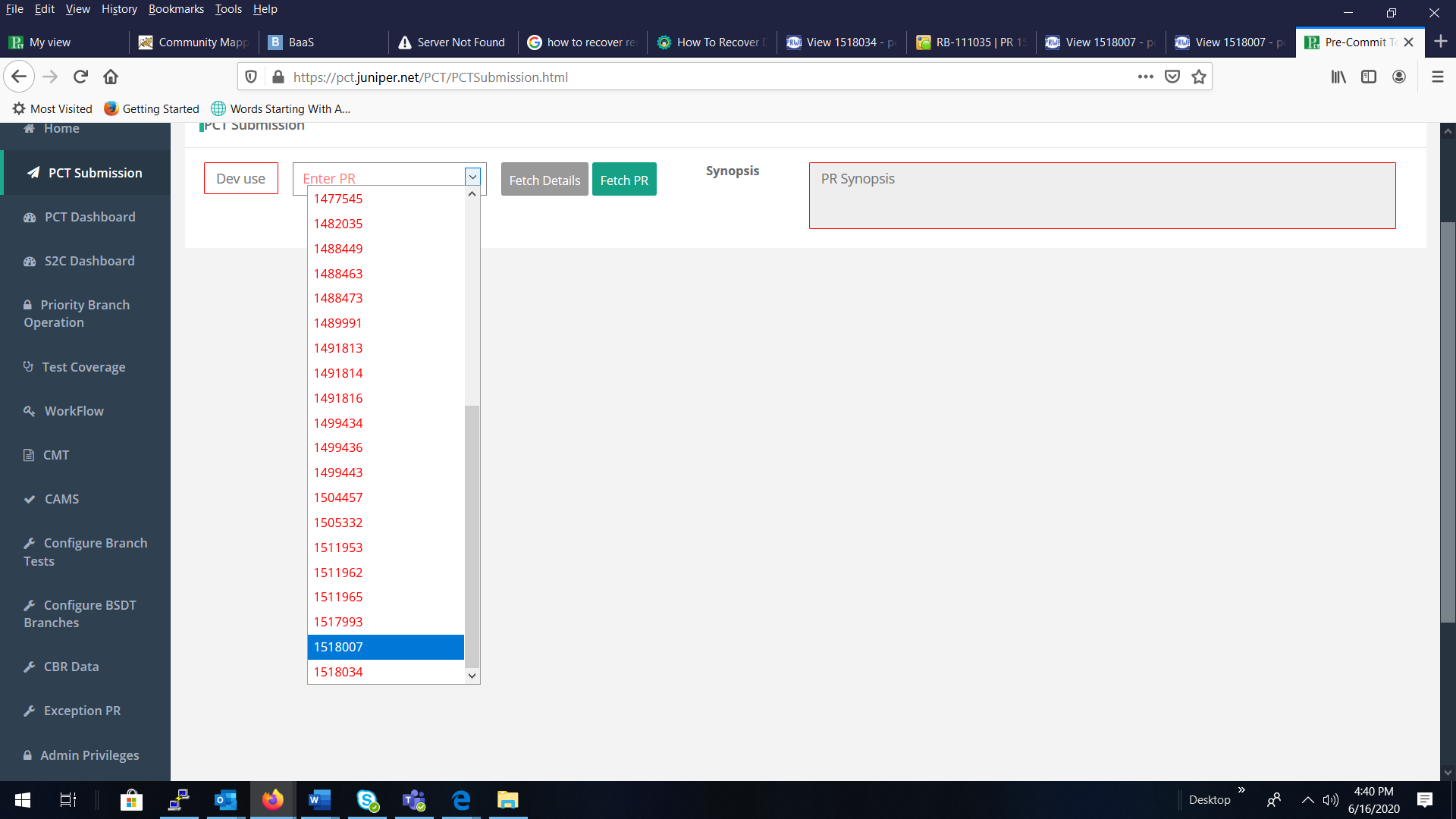
* Commit your code
* Exercise 1 – Commit Using PCT

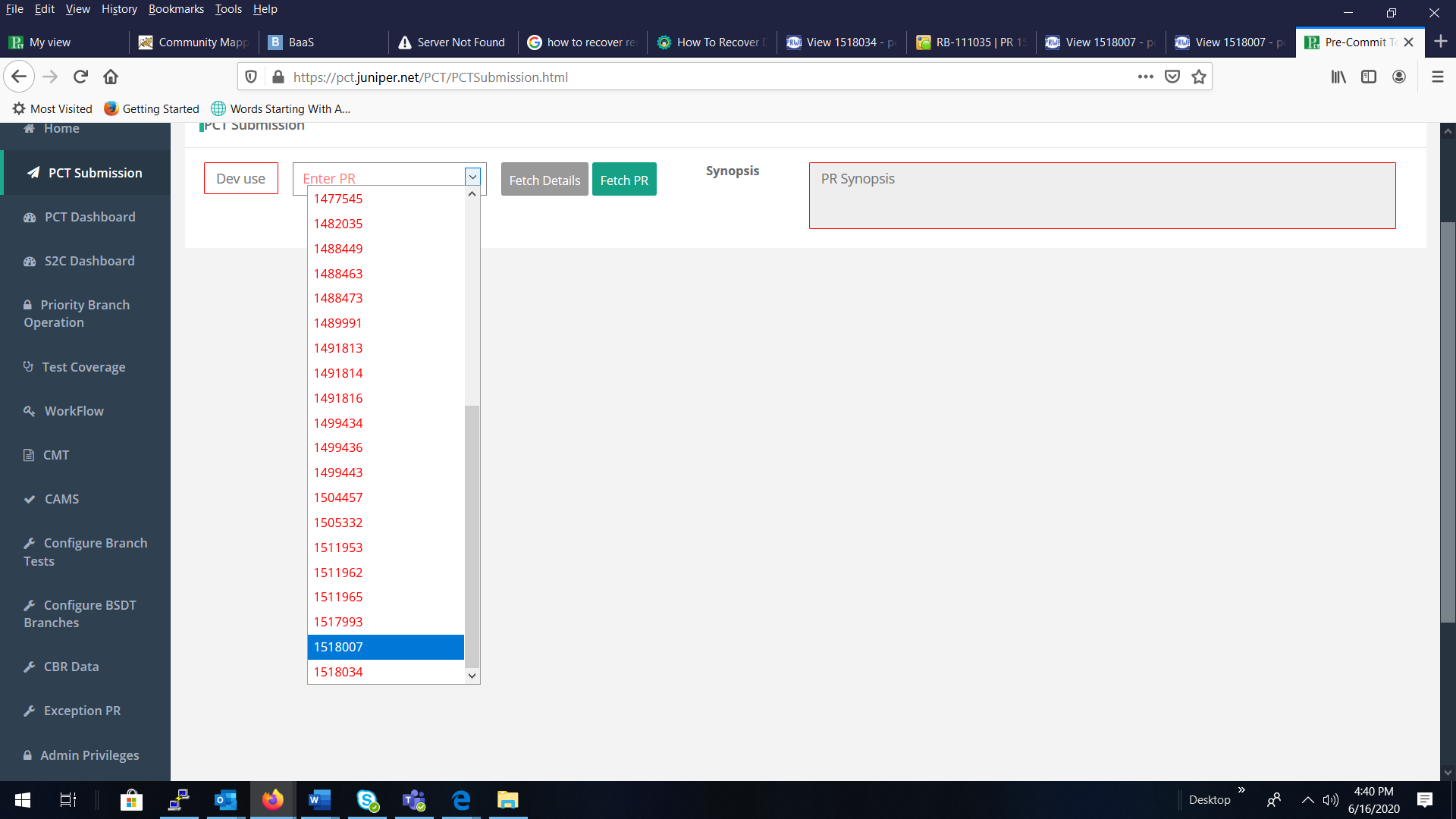
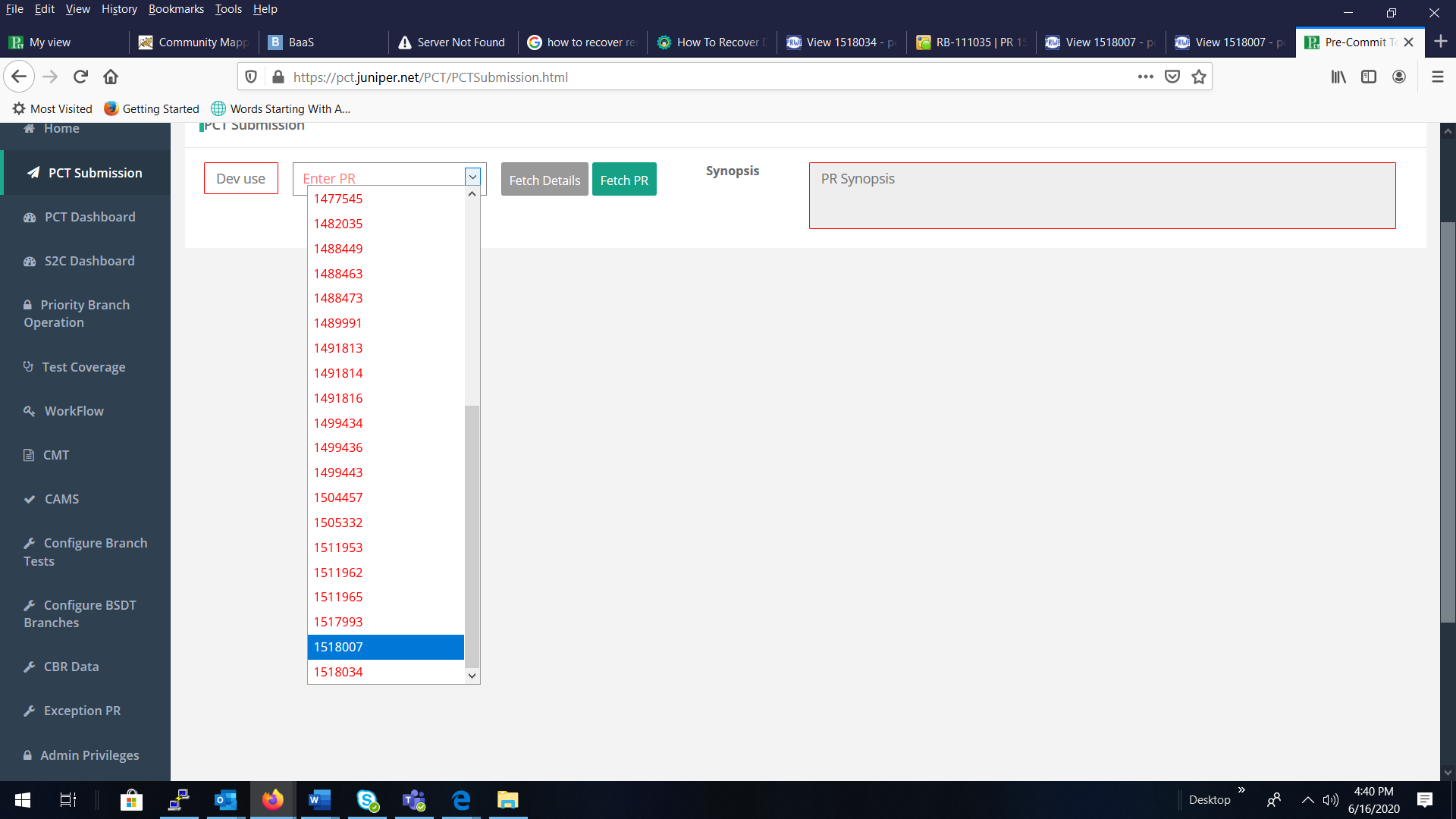
In this exercise, you commit using PCT.

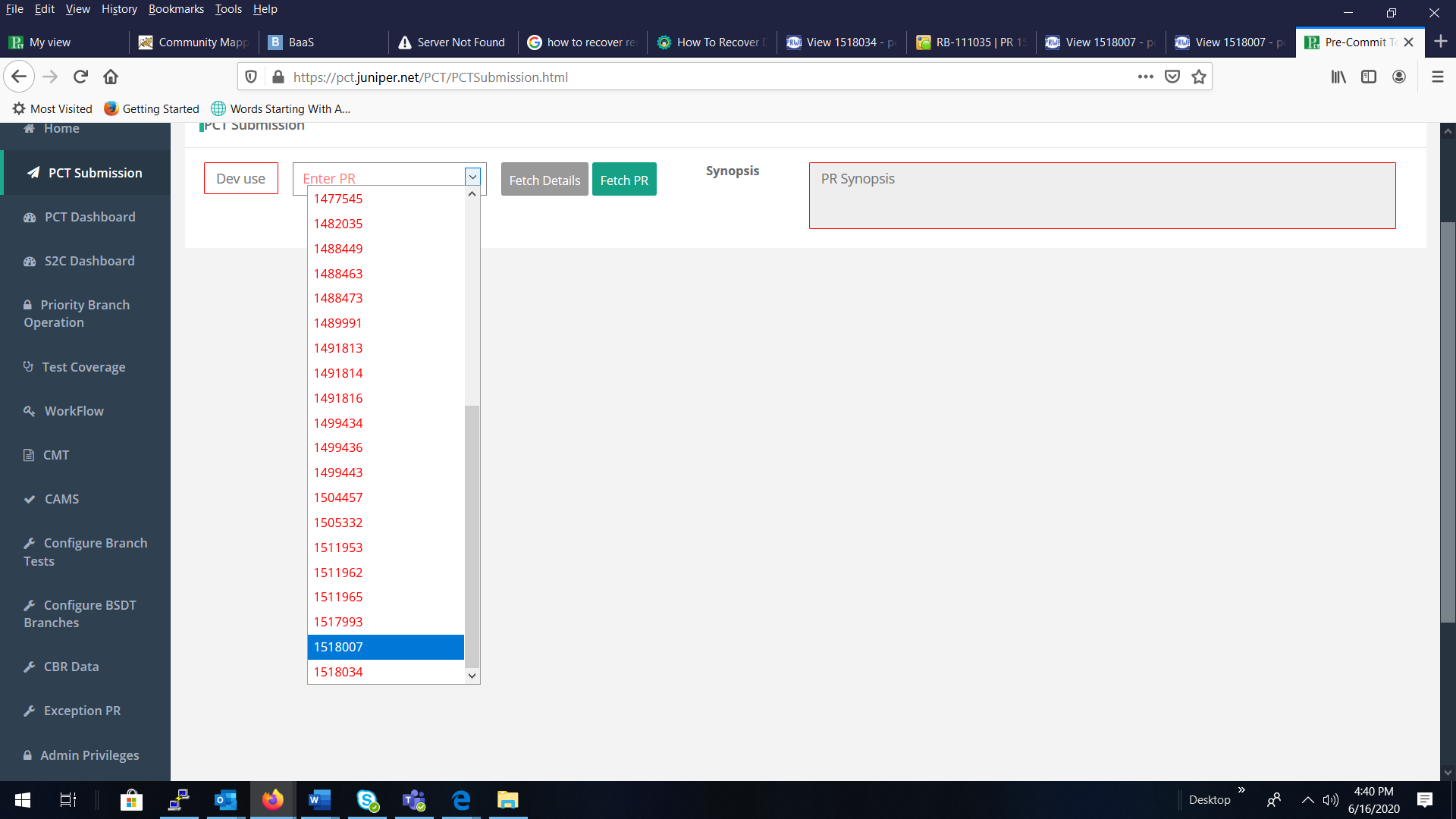
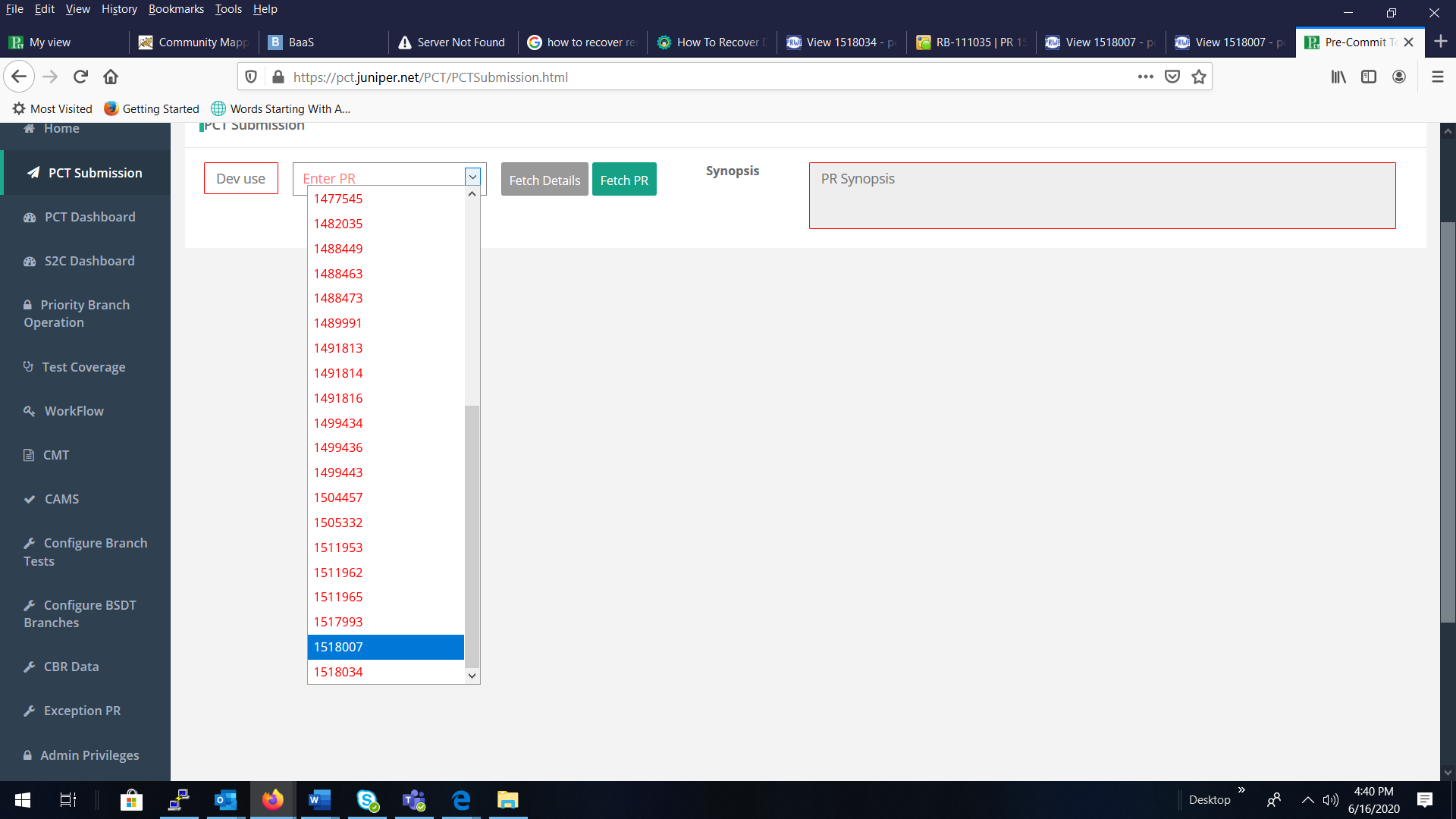
1. Open a browser and go to **https://pct.juniper.net**.

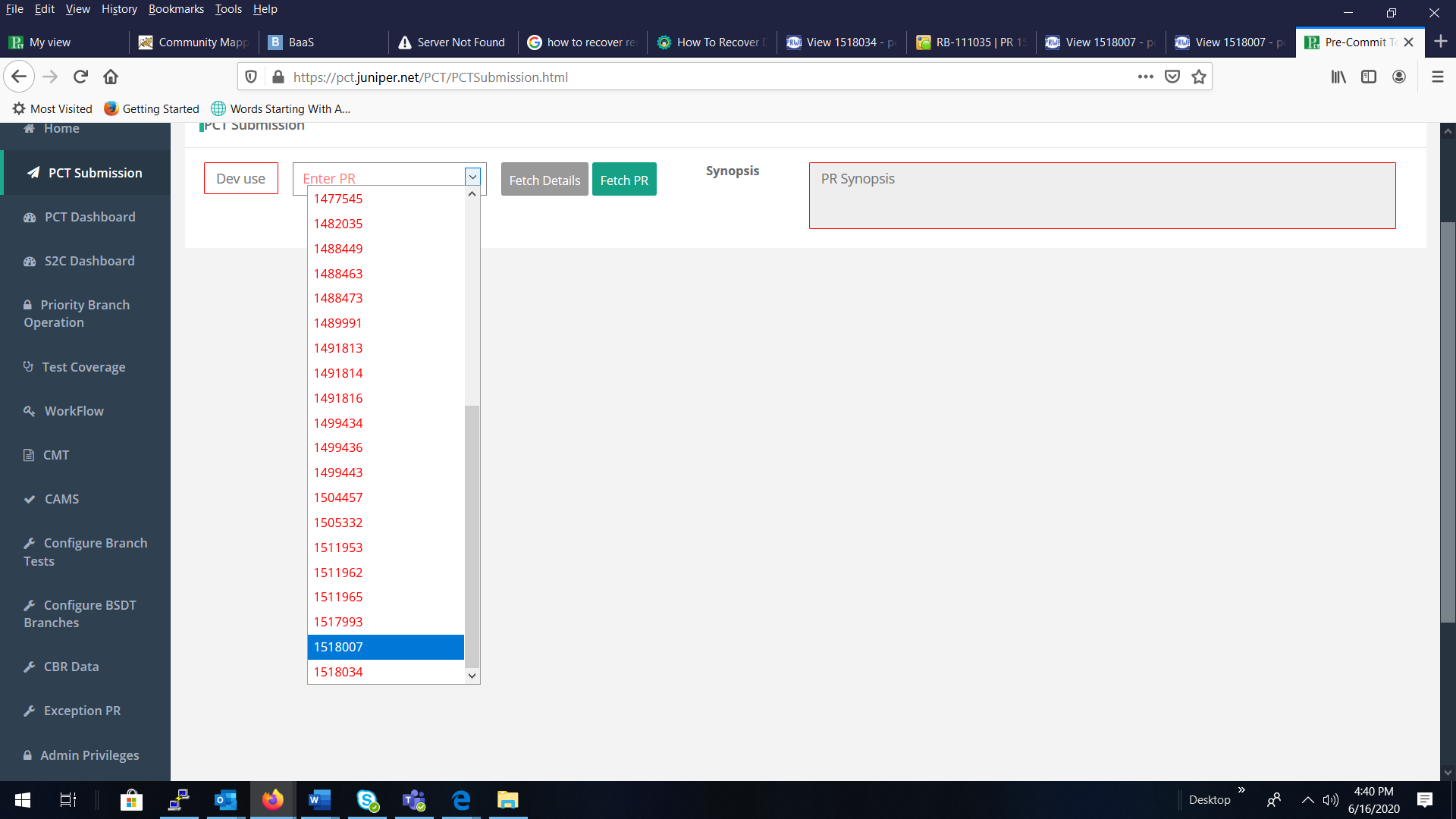
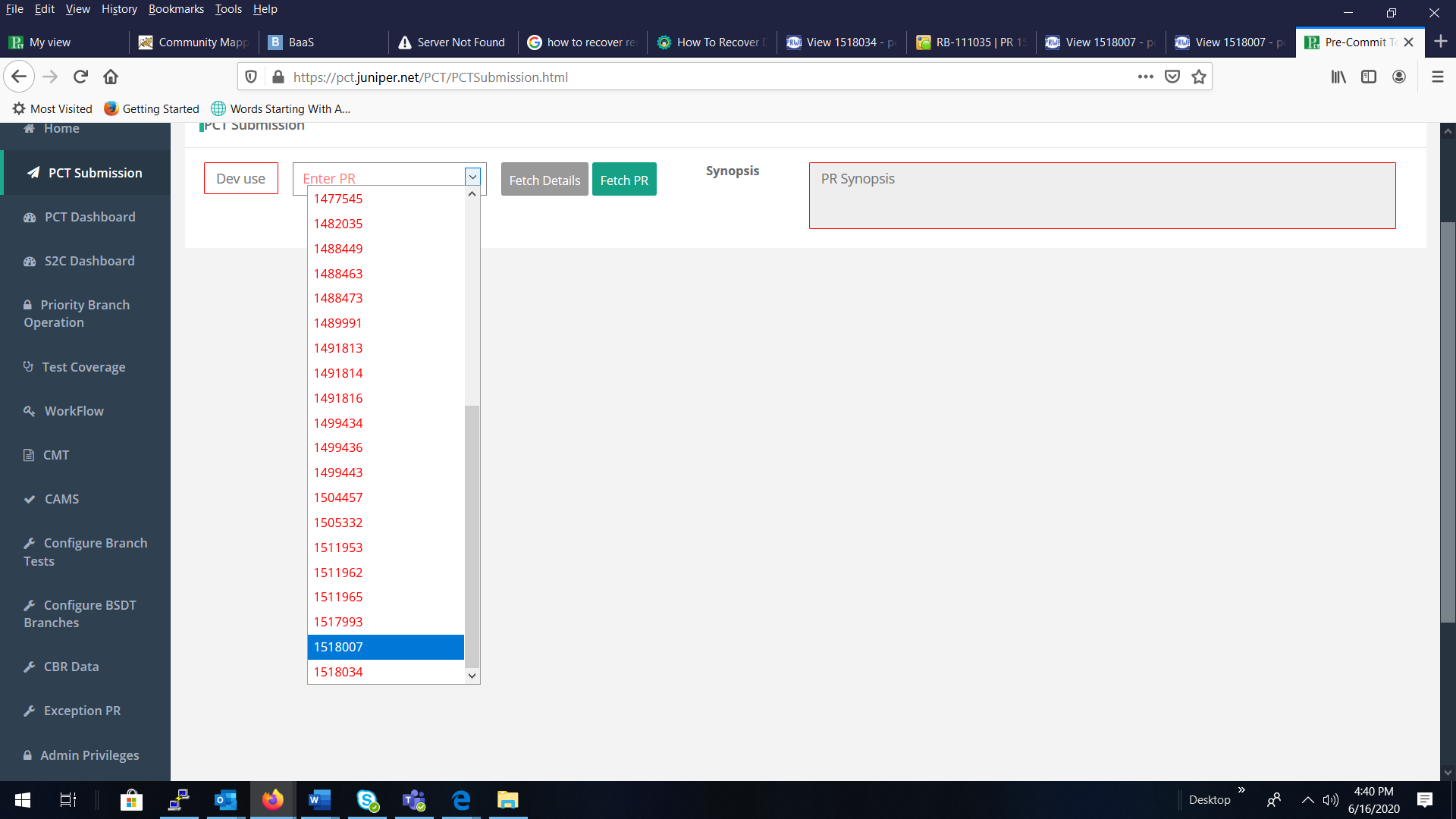


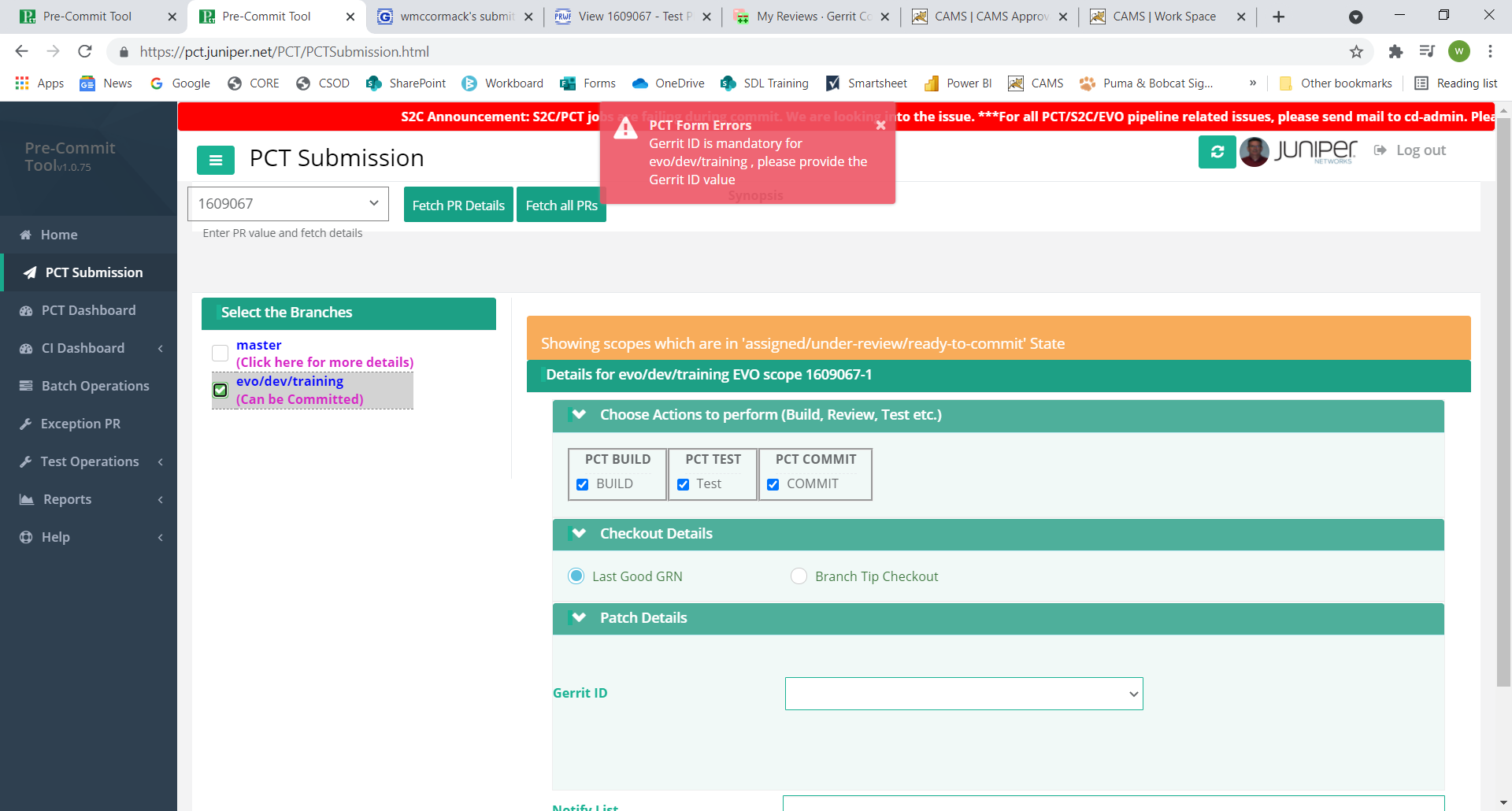
1. Click **PCT Submission** in the menu on the left. This will open the PCT Submission page.
2. Enter your PR number or click **Fetch PR** to get a list of your PR’s in dropdown box. Then select your PR from the list.

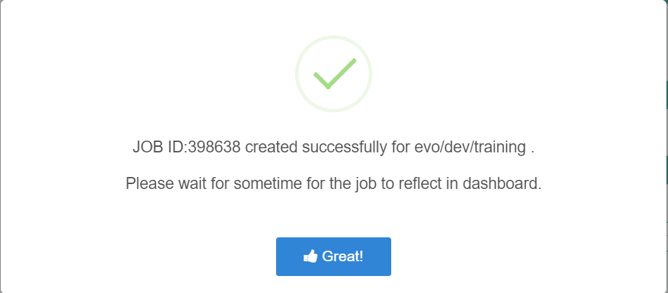


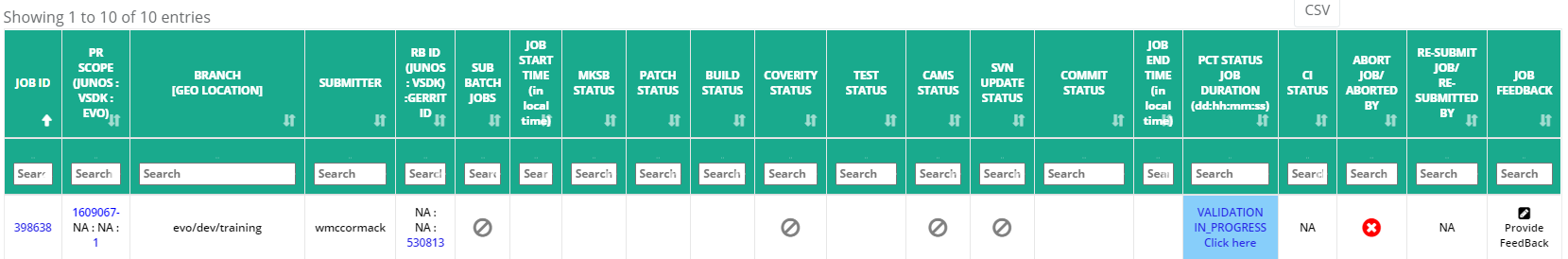
3  

2 

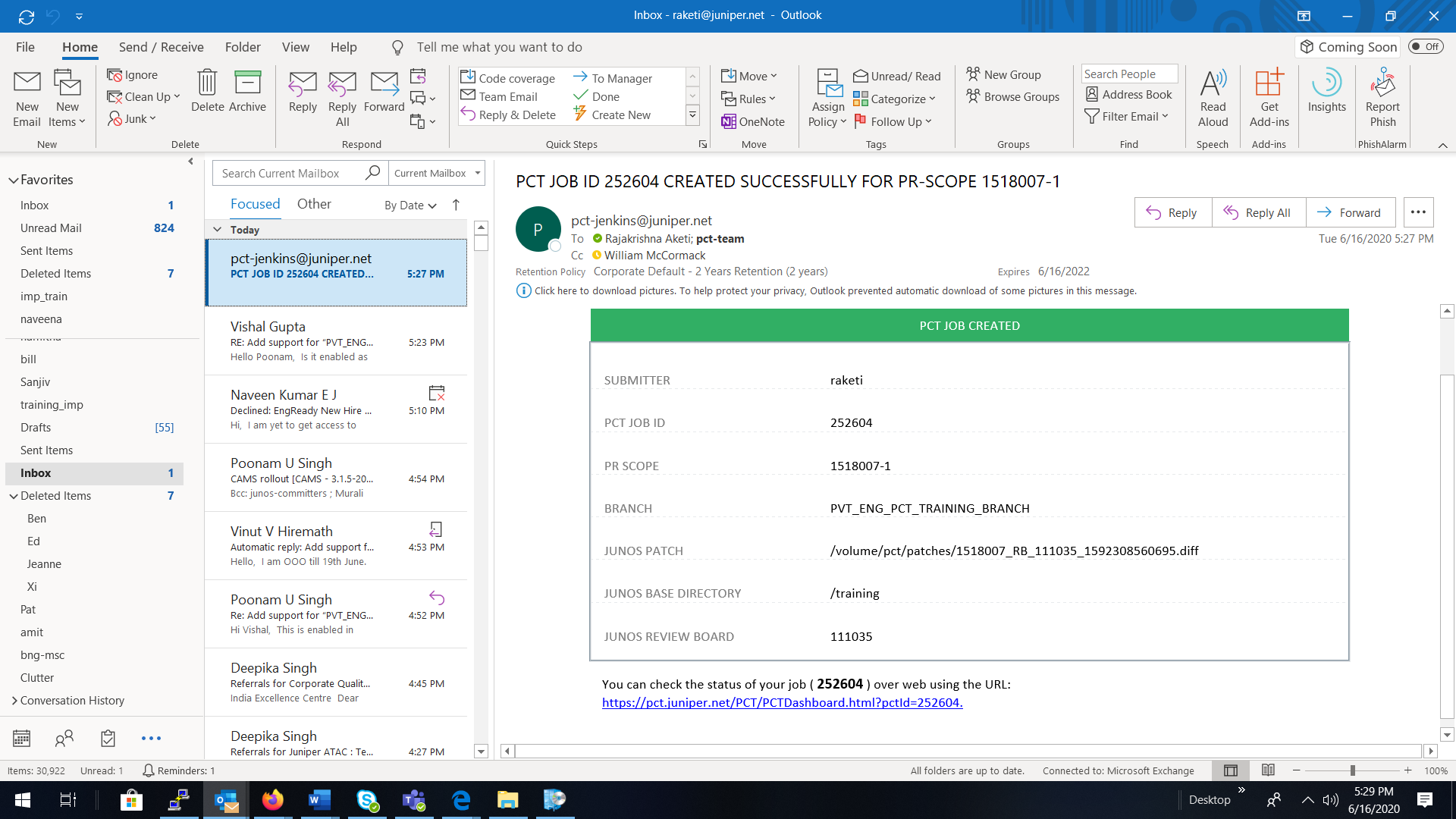
1  

1. A list of target branches appears on the left. Put a check next to the branch **evo/dev/training** from the menu. This will cause data to appear about the branch as shown below.  
   
2. In the **Gerrit ID** text box, click the arrow to pull up your Gerrit ID, then click **Fetch Info.**
3. Click **Submit**, and then **Submit** again when a pop-up message appears. Afterwards, you should get a success message.

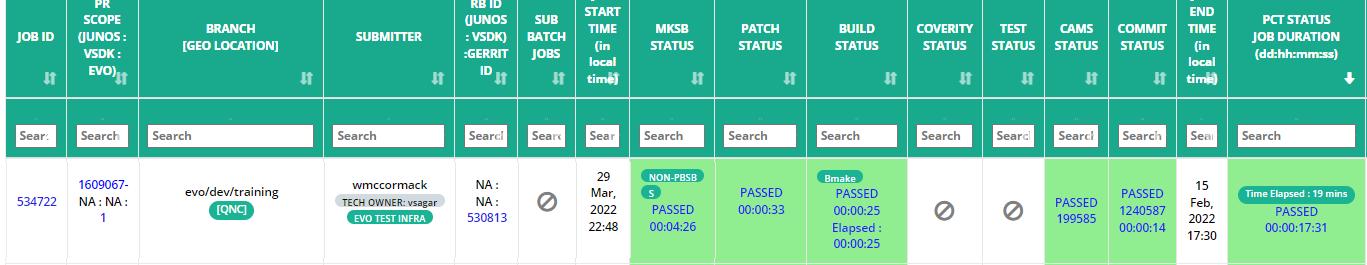


After clicking the “**Great**!” button, the **PCT Dashboard** should open. Your submission should be in the “**Validation in Progress**” state. Your submission will take anywhere from 15 minutes to an hour to progress through the various stages. Refresh your browser periodically to see the latest stage.  


1. You should get an email message confirming that you have created a PCT job.



1. Keep refreshing the dashboard until your job is completed. When it finishes, the Commit Status field will turn green and will say “PASSED.”



1. Once the dashboard says PASSED, you are done. Congratulations, you have successfully committed!

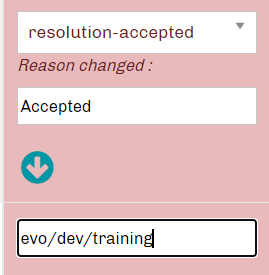
* Exercise 2 – Close Out Your PR

Once you have committed, the state of your PR in GNATS should move to Awaiting Build. Normally, you would wait for the automated build system to create an image so you can verify that the fix works. In our private branch, there is no automated build system. Instead, you will manually move the state to Verify Resolution. After that, you can manually close the PR.

Step 1: Move the state from **Awaiting Build** to **Verify Resolution** by doing the following:

* 1. Open your PR in GNATS.
  2. Edit the PR.
  3. Go to the Scope tab.
  4. Move the state to **Verify Resolution** and provide a reason (any text).
  5. Scroll down to the Target field and enter ***evo/dev/training****.*
  6. **Click the Update PR button**.

1. Move the state from **Verify Resolution** to **Closed** by doing the following:
   1. Edit the PR.
   2. Go to the Scope tab.
   3. Move the state from Verify Resolution to **Closed** and provide a reason.
   4. Go to the **Verification Status** field and enter **Resolution Accepted**.
   5. Provide a reason in the ***Reason changed*** field.
   6. Go to the Verified In field and enter **evo/dev/training**.



* 1. Click the **Update PR** button**.** At this point, the state of your PR should be **Closed**.