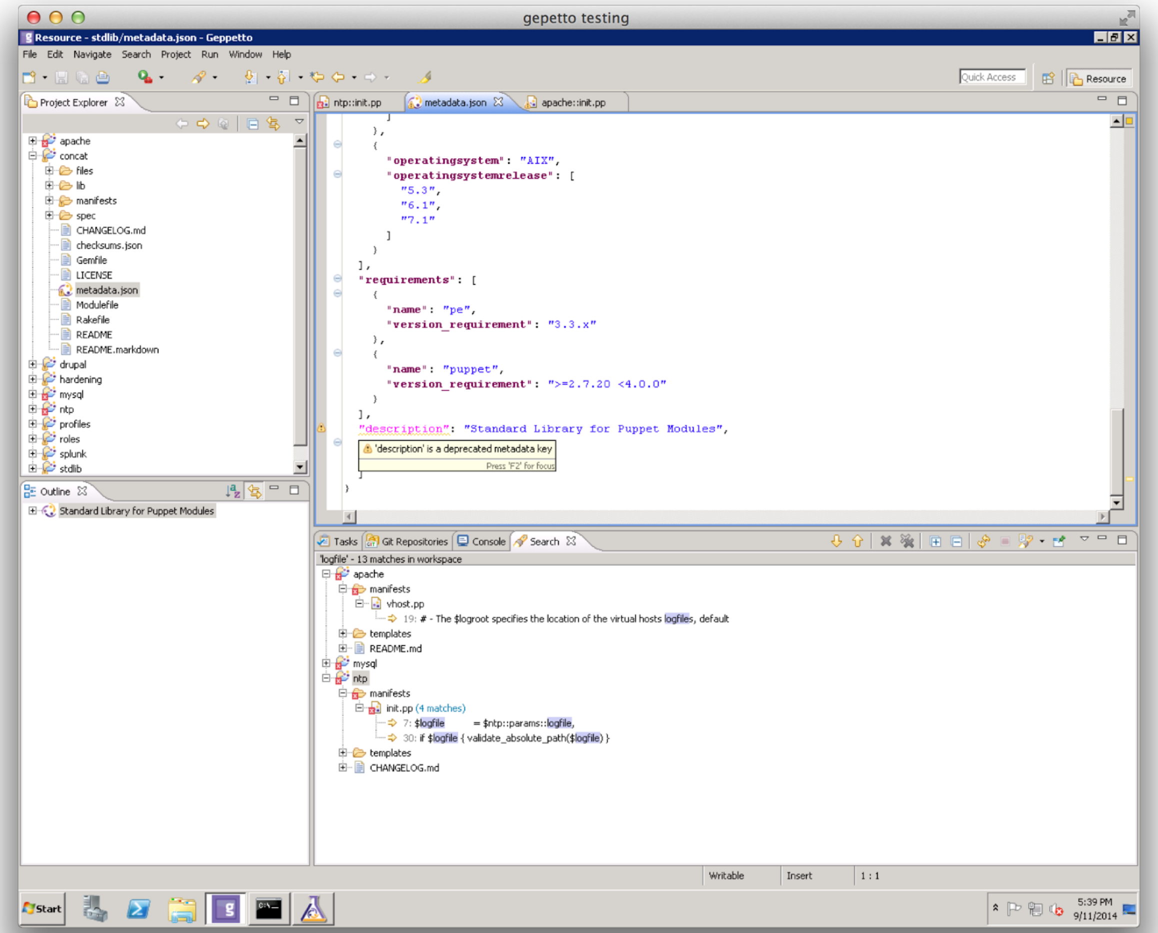
Using Geppetto in your infrastructure

Geppetto is an Eclipse-based IDE for developing Puppet modules. The editor provides features such as syntax highlighting, code completion, error highlighting, refactoring, and even integrates directly with the Puppet Forge.



```
      ]
    },
    {
      "operatingsystem": "AIX",
      "operatingsystemrelease": [
        "5.3",
        "6.1",
        "7.1"
      ]
    }
  ],
  "requirements": [
    {
      "name": "pe",
      "version_requirement": "3.3.x"
    },
    {
      "name": "puppet",
      "version_requirement": ">=2.7.20 <4.0.0"
    }
  ],
  "description": "Standard Library for Puppet Modules",
  ]
}
```

```
'description' is a deprecated metadata key
Press 'F2' for focus
```

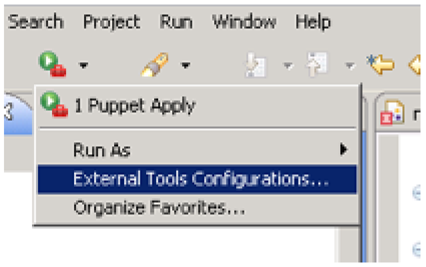
```
'logfile' - 13 matches in workspace
apache
  manifests
    vhost.pp
      19: # - The $logroot specifies the location of the virtual hosts logfiles, default
  templates
  README.md
mysql
ntp
  manifests
    init.pp (4 matches)
      7: $logfile         = $ntp::params::logfile,
      30: if $logfile { validate_absolute_path($logfile) }
  templates
  CHANGELOG.md
```

This tutorial describes how you can configure Geppetto for this style of development and a useful workflow for maintaining and updating your in-house Puppet modules. Unfortunately, some of the steps are still manual, but we will configure shortcuts for as much as we can. For simplicity, we will assume that you have some form of git server running on your Puppet Master, allowing you to easily create new repositories as needed with the proper hooks in place to keep the code in your Puppet modulepath updated.

Configuring your Workspace

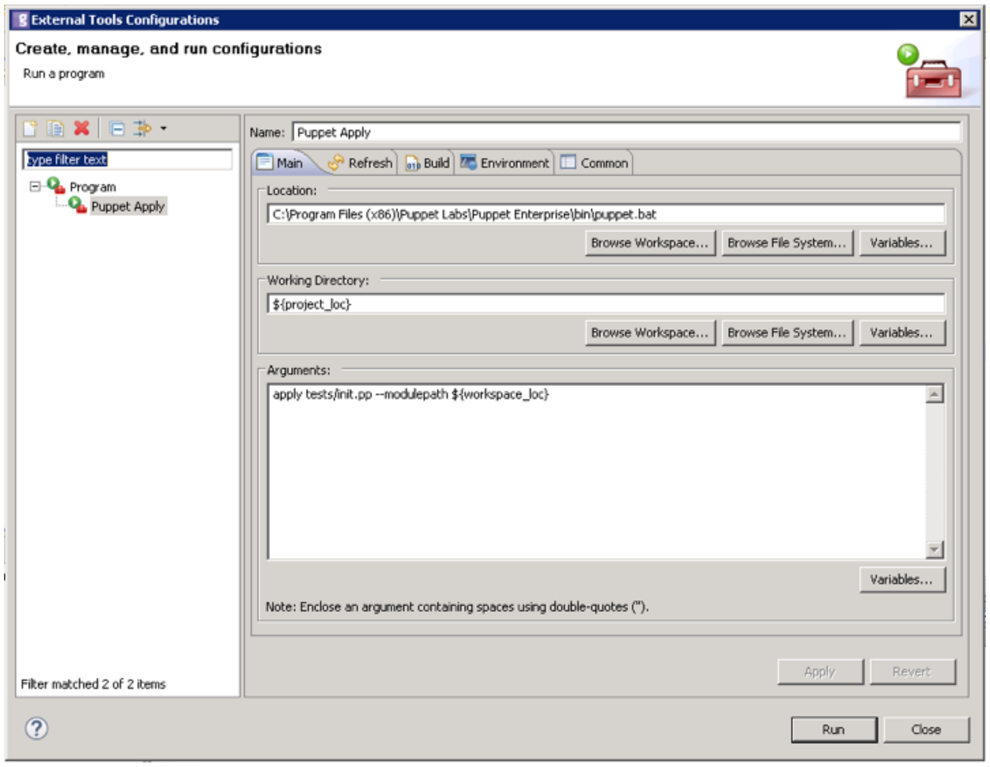
Your first step is to define a Workspace for your infrastructure. If this project is the only code that will be managed by Geppetto, go ahead and make it the default so you don't have to choose a workspace each time you start the environment.

Next, you'll want to configure an External Tool entry to run puppet apply for testing purposes. Click the downward facing arrow next to the *Run* button and choose "External Tools Configurations…"



In the configuration window, you'll want to add a new tool and configure it appropriately.

- Name:
    - Puppet Apply
- Working Directory:
    - `${project_loc}`
- Arguments:
    - `apply tests/init.pp --modulepath ${workspace_loc}`
- Configure the Location to match your system. For example, on a Windows workstation, you might set it to:
    - `C:\Program Files (x86)\Puppet Labs\Puppet Enterprise\bin\puppet.bat`



Name: Puppet Apply

Location:
C:\Program Files (x86)\Puppet Labs\Puppet Enterprise\bin\puppet.bat

Working Directory:
${project_loc}

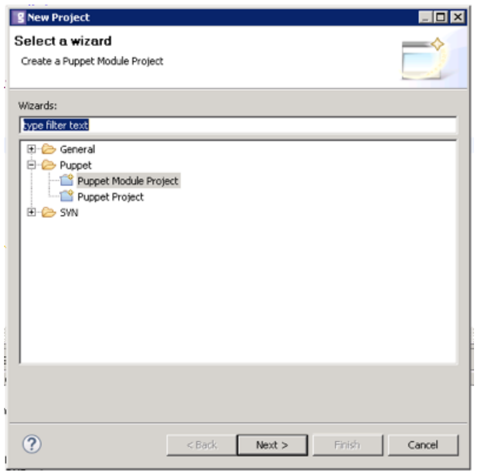
Arguments:
apply tests/init.pp --modulepath ${workspace_loc}

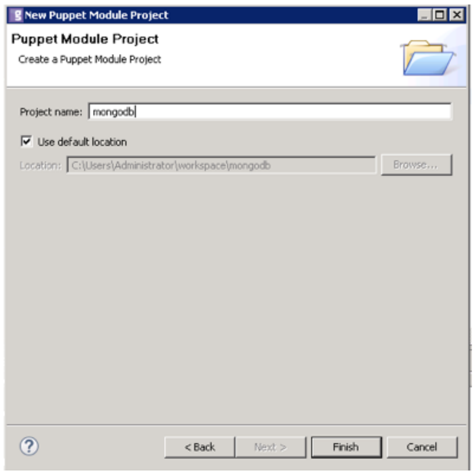
Note: Enclose an argument containing spaces using double-quotes (").

Now, when you select a module in the Project Explorer, you'll be able to enforce the `tests/init.pp` smoke test locally for quick class validation. You'll still be able to run Puppet from the command line for more in-depth usage if needed. You may want to duplicate this tool configuration and add `--noop` to the arguments list to also get a *Puppet Noop Apply* action.

Creating a module

Now it's time to build a module. Choose the New Project wizard by clicking the New button or right clicking in the Project Explorer and choosing *New -> Project*. Select the Puppet Module Project, select *Next*, and then give it a name. Allow Geppetto to add it to the default location.





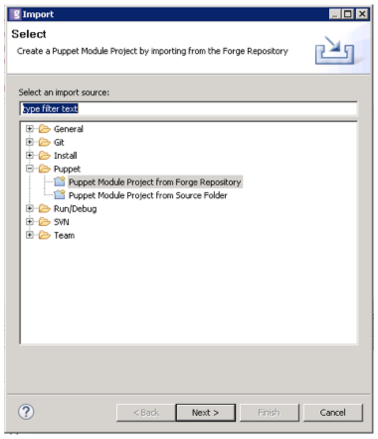
Project name: mongodb

Use default location: ✓

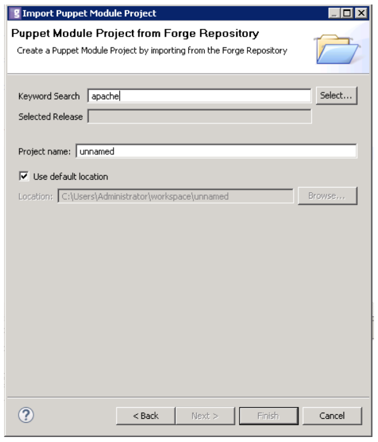
Location: C:\Users\Administrator\workspace\mongodb

At this point, you'll have a local project containing a module that you can develop and test as you like. The Puppet code in `tests/init.pp` will be enforced locally when you click the Run button. Add projects for each module you create.

Unless your modules are uncomplicated, it's likely that you'll be using some upstream modules from the Puppet Forge. Let's add those to our workspace too. This will allow us to test enforce our code locally, and it will allow Geppetto to work its code completion and error highlighting magic. Click the *File -> Import* menu option, or right click inside the Project Explorer and choose *Import*. Select the Forge Repository option in the wizard and click *Next*.


Import
Select
Create a Puppet Module Project by importing from the Forge Repository
Select an import source:
type filter text
General
Git
Install
Puppet
Puppet Module Project from Forge Repository
Puppet Module Project from Source Folder
Run/Debug
SVN
Team
< Back
Next >
Finish
Cancel

In the keyword search box, type the name of the module you're looking for and hit the *Select* button.


Import Puppet Module Project
Puppet Module Project from Forge Repository
Create a Puppet Module Project by importing from the Forge Repository
Select...
Browse...
< Back
Next >
Finish
Cancel

Keyword Search: apache

Selected Release:

Project name: unnamed

Use default location: ✓

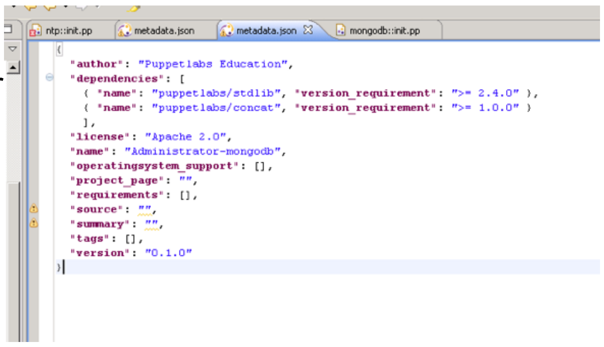
Location: C:\Users\Administrator\workspace\unnamed

This will bring up a dialog box with a list of modules that match the keyword(s) you entered. Choose the module you want and click

OK a couple times to import a local copy of the module into your workspace.

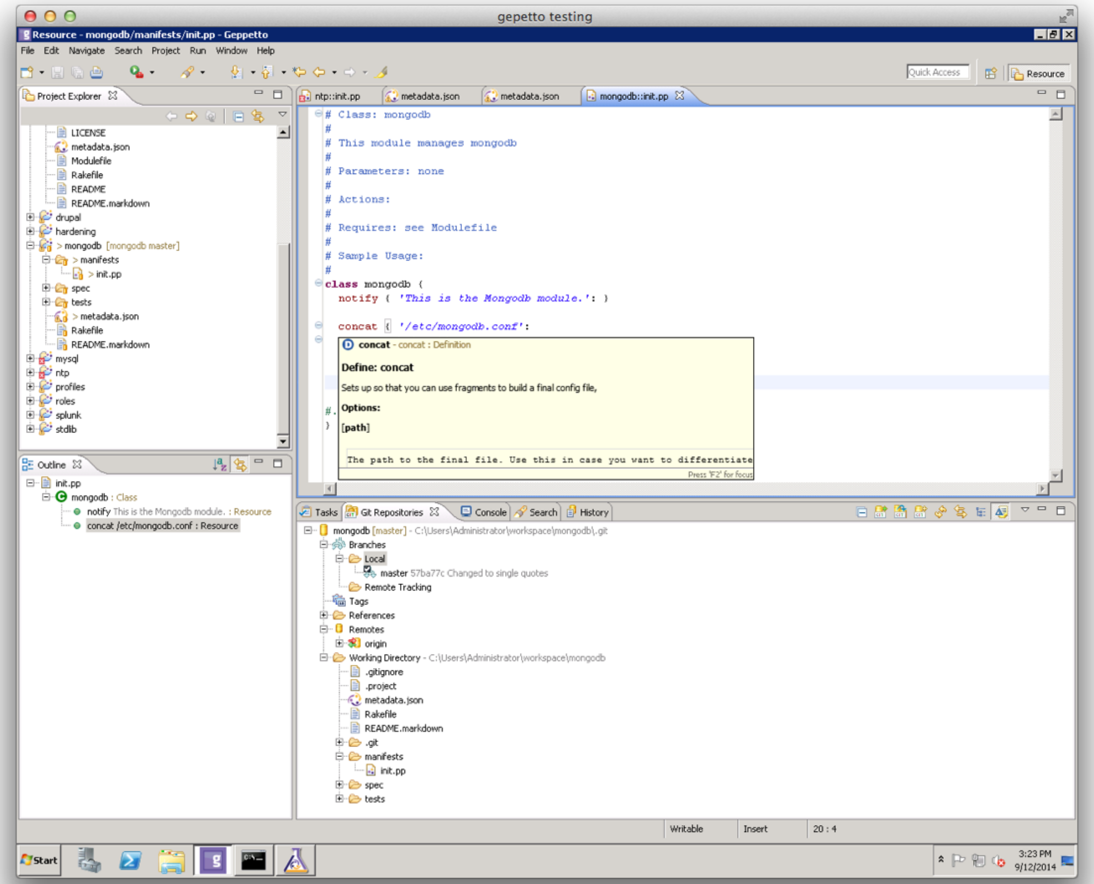
Module Dependencies

Now you've got at least a couple modules to work with. You'll need to tell Geppetto about their dependencies, so it knows to index and build references between them all for code completion and error highlighting. Edit each of your modules' `metadata.json` file and add the appropriate module dependencies.



```
{
  "author": "Puppetlabs Education",
  "dependencies": [
    { "name": "puppetlabs/stdlib", "version_requirement": ">= 2.4.0" },
    { "name": "puppetlabs/concat", "version_requirement": ">= 1.0.0" }
  ],
  "license": "Apache 2.0",
  "name": "Administrator-mongodb",
  "operatingsystem_support": [],
  "project_page": "",
  "requirements": [],
  "source": "",
  "summary": "",
  "tags": [],
  "version": "0.1.0"
}
```

Once the metadata is correct, Geppetto will be able to use the references from other modules, including defined types, classes, functions, and documentation for each.



```
LICENSE
metadata.json
Modulefile
Rakefile
README
README.markdown
drupal
hardening
> mongodb [mongodb master]
  > manifests
    > init.pp
  spec
  tests
  > metadata.json
  Rakefile
  README.markdown
mysql
ntp
profiles
roles
splunk
stdlib
```

```
init.pp
  mongodb : Class
    notify This is the Mongodb module. : Resource
    concat /etc/mongodb.conf : Resource
```

```
# Class: mongodb
#
# This module manages mongodb
#
# Parameters: none
#
# Actions:
#
# Requires: see Modulefile
#
# Sample Usage:
#
class mongodb {
  notify { 'This is the Mongodb module.': }

  concat { '/etc/mongodb.conf':
```

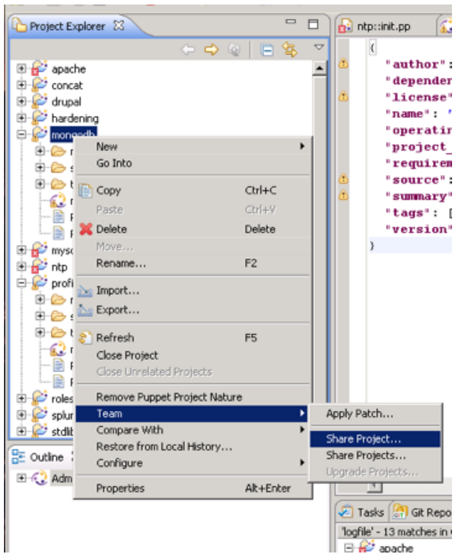
```
concat - concat : Definition

Define: concat

Sets up so that you can use fragments to build a final config file,

Options:

[path]

 The path to the final file. Use this in case you want to differentiate

Press 'F2' for focus
```

```
mongodb [master] - C:\Users\Administrator\workspace\mongodb\.git
  Branches
    Local
      master 57ba77c Changed to single quotes
    Remote Tracking
  Tags
  References
  Remotes
    origin
  Working Directory - C:\Users\Administrator\workspace\mongodb
    .gitignore
    .project
    metadata.json
    Rakefile
    README.markdown
    .git
    manifests
      init.pp
    spec
    tests
```

Configuring your repositories

Now the question is how you get your code to your Puppet master. The usual workflow is to use a git server configured either with r10k or with post-update hooks to update your modulepath. You should allow r10k to manage Forge modules rather than adding them to local repositories. For the purpose of this tutorial, we'll assume that you've got it covered once the updates hit your git server.

In order to deploy configuration updates from Geppetto, you will need two things. You'll need a git repository for each of your own modules to push commits to and you'll need to initialize each of your module projects with a local git repository and configure a remote linked to your upstream repository.

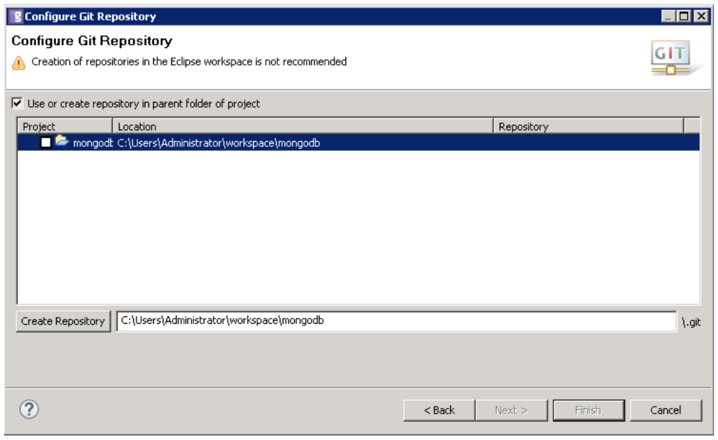


On your git server, create a new repository for each of your own modules, but don't add any commits to it. You'll want to create this as a bare repository, which many frontends like GitLab will do automatically. Add any integrations needed to update your Puppet master modulepath.

Then we'll need to configure your module to deploy updates to this new repository. Right click on your project and choose *Team -> Share Project*.

Next, you'll want to tell Geppetto to create a single repository for the project by checking the *Use or create repository in parent folder of project* checkbox. Then select the project name and click the Create Repository button. Once complete, you can close the dialog box.

This will create a local repository in your module's Project folder allowing you to manage revisions and branches locally.


Configure Git Repository

Configure Git Repository

Creation of repositories in the Eclipse workspace is not recommended

[✓] Use or create repository in parent folder of project

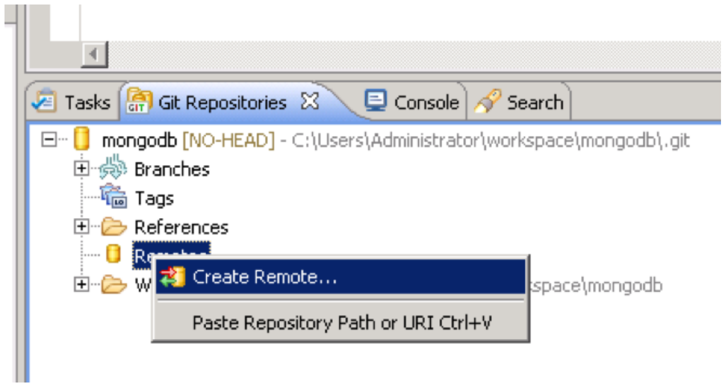
Project	Location	Repository
mongodb	C:\Users\Administrator\workspace\mongodb	

Create Repository C:\Users\Administrator\workspace\mongodb \.git

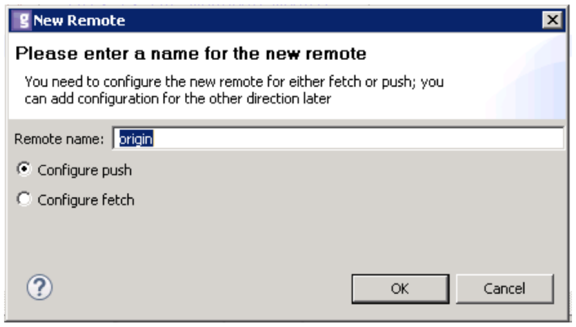
< Back Next > Finish Cancel

Adding remotes

In order to deploy your code, you'll need to configure a remote pointing to the repository on your Puppet master. Switch to the *Git Repositories* view and find your project. Expand it, right click on *Remotes* and select *Create Remote*.


Tasks
Git Repositories
Console
Search
mongodb [NO-HEAD] - C:\Users\Administrator\workspace\mongodb\.git
Branches
Tags
References
Create Remote...
Paste Repository Path or URI Ctrl+V

In the dialog box, accept the default remote name or *origin*, set the direction to *push*, and click the *OK* button.


New Remote

Please enter a name for the new remote

You need to configure the new remote for either fetch or push; you can add configuration for the other direction later

Remote name: origin

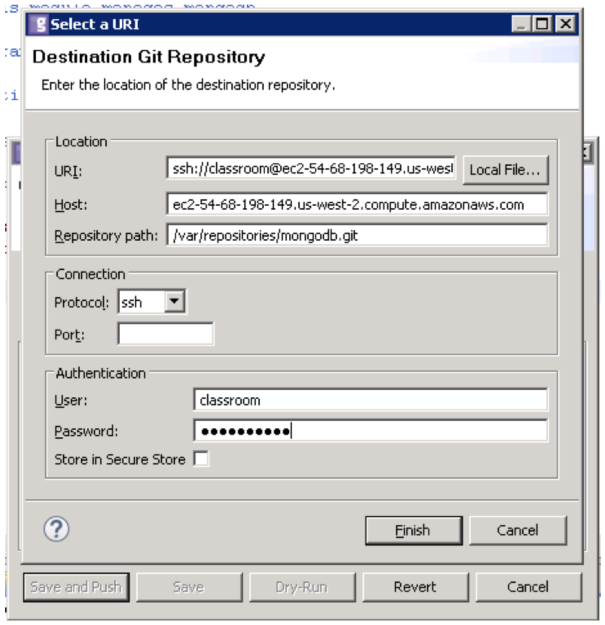
[✓ Configure push]

Configure fetch

Click the *Change* button to change the remote URI and configure the resulting dialog with the appropriate information for the git repository for this module. Enter the hostname of the git server and the path to the repository. Choose the network protocol and enter your user credentials.

This example shows the git remote configuration for a module in the Windows Essentials virtual training course.

Click the `Save` button when you are finished. Do not `Save and Push` yet, because we haven't committed any code to push.


Select a URI

Destination Git Repository

Enter the location of the destination repository.

Location

URI: ssh://classroom@ec2-54-68-198-149.us-wesl

Host: ec2-54-68-198-149.us-west-2.compute.amazonaws.com

Repository path: /var/repositories/mongodb.git

Connection

Protocol: ssh

Port:

Authentication

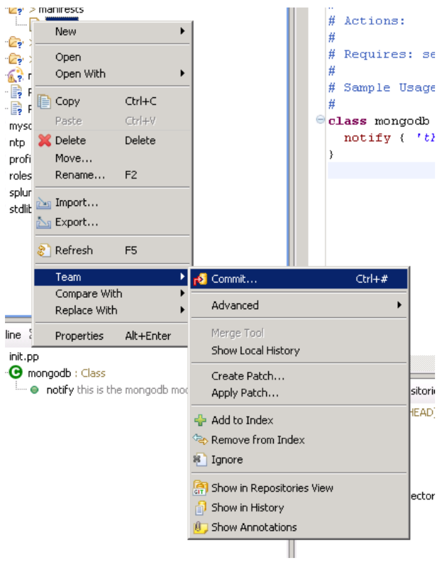
User: classroom

Password: ●●●●●●●●●●●

Store in Secure Store

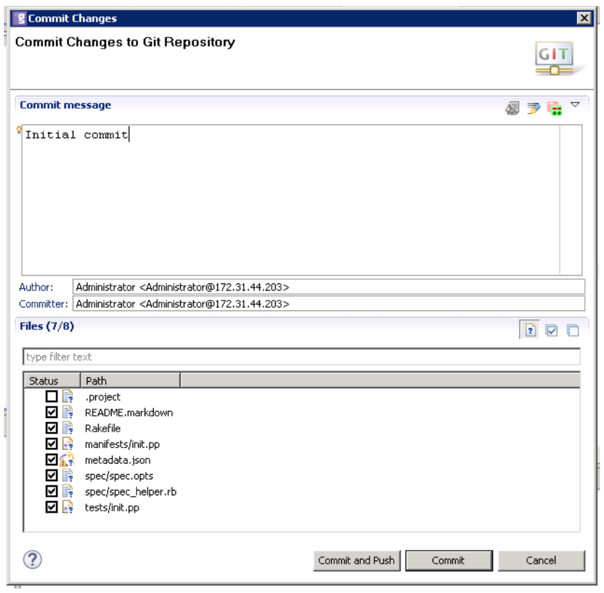
Making your first commit

Once you've saved the remote configuration, you're ready to commit some code updates! Right click on your project in the Project Explorer and choose *Team -> Commit*.



```
# Actions:
#
# Requires: se
#
# Sample Usage
#
class mongodb
  notify { 't
}
```

In the resulting dialog, you should enter a descriptive and meaningful commit message. This message should summarize briefly what you changed, but more importantly, it should explain why the changes were necessary. The commit message will serve as documentation for anyone later reading through history, or for collaborators in a code review environment.



```
Initial commit
```

Author: Administrator <Administrator@172.31.44.203>

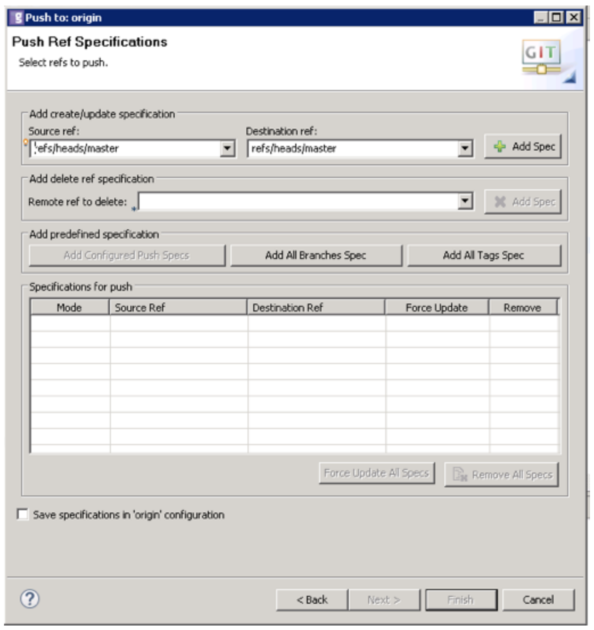
Committer: Administrator <Administrator@172.31.44.203>

Status	Path
☐	.project
☑	README.markdown
☑	Rakefile
☑	manifests/init.pp
☑	metadata.json
☑	spec/spec.opts
☑	spec/spec_helper.rb
☑	tests/init.pp

Below the text input area, you should check the files you want to commit. You may commit the `.project` file if you like. This file contains static project metadata that might make it easier for other Geppetto users to import the project. If you prefer not to commit it, you may [create a .gitignore file]() to instruct git to ignore the file.

Once you are satisfied, simply click the *Commit and Push* button. You may also choose to make several code commits before pushing them all at once.

The first time you push, Geppetto will require you to configure ref specifications. You should select the name of the local branch you are working on and the name of the remote branch to push it to. For the purposes of this tutorial, we will simply use the master branch, but in production you should be have feature or dev branches.



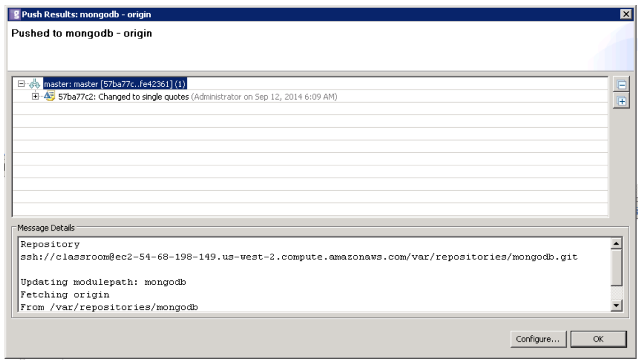
Source ref: refs/heads/master

Destination ref: refs/heads/master

Select the branch name in the source and destination ref dropdowns and click the `Add Spec` button. Click the `Finish` button to perform the push action.

Repository management and etiquette is beyond the scope of this tutorial, but a good rule of thumb is that you should do your best to make the repository easy to navigate for other members of your team. They will thank you for it later, and if you ever have to crawl through history, future-you will thank current-you.

Once the push is complete, you will get a results window iterating each of the branches or tags you pushed and the response from the git server If your `post-update` hook has any output, it will be displayed in the lower pane of this window.



```
master: master [57ba77c..fe42361] (1)
    57ba77c2: Changed to single quotes (Administrator on Sep 12, 2014 6:09 AM)
```

```
Repository
ssh://classroom@ec2-54-68-198-149.us-west-2.compute.amazonaws.com/var/repositories/mongodb.git

Updating modulepath: mongodb
Fetching origin
From /var/repositories/mongodb
```

This example is displaying the output of the `post-update` hook from the Windows Essentials virtual training course.

At this point, assuming that your git server environment has been configured properly to update your Puppet master's modulepath with your code updates, deploying code to your infrastructure is a simple matter of writing and testing code on your Geppetto workstation and then pushing commits upstream. Test it now by classifying a client node and enforcing configuration by running `puppet agent -t` on it.

While on the topic of workflow, I will caution you that this is only the beginning. A robust infrastructure should have many more barriers between developer workstations and the live production environment. You should implement a code review process and an automated testing gatekeeper such as Jenkins or Travis CI.

Most importantly, on a live infrastructure, you should only merge signed-off commits to the production branch and only commit to feature or testing branches.