Linux System Management with Puppet, Gitlab, and R10k

Scott Nolin, SSEC Technical Computing

22 June 2017

Introduction

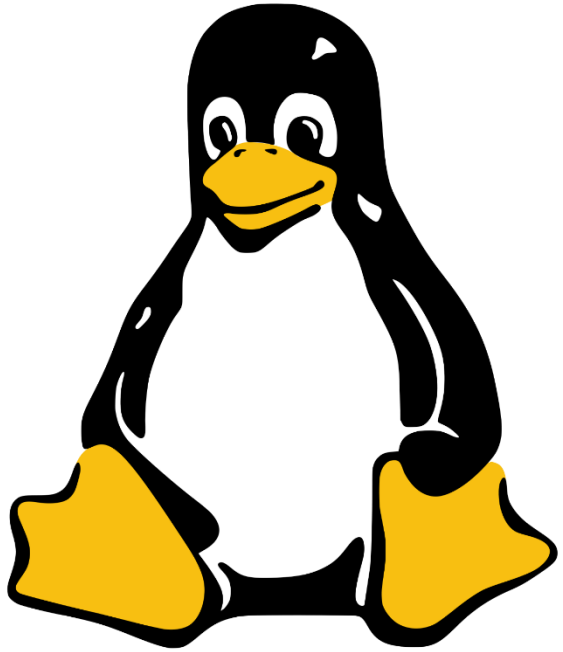
I am here to talk about how we do Linux configuration management at the Space Science and Engineering Center.

I am doing this to encourage others doing similar work so we can ideally share ideas and learn from each other.

Why We need Linux configuration management

SSEC's mission and structure requires many hundreds of systems, and many groups with different needs running systems. Linux systems are the core of our research systems.

- Ensure consistency and repeatability
- Faster time from install to production
- Adapt to change quickly
- Lower labor costs
- Computer Security
  - Applying computer policy consistently
  - Auditing



Why Version Control

- Version control of configurations reduces fear and increases accountability.
  - See Dave Plonka's 'Sysadmin File Revision Control with RCS' from 1998
  - Modern system administration favors more frequent small changes, and version control makes this easy.
- We had done this for many years with CVS, SVN, and now git for some of our configurations.
  - DNS zone files
  - Monitoring server configuration (check_mk/omd)
  - Version control was the **end** of the process – after changing a configuration it got stored in version control.
- With Puppet, you manage 'everything', so configuration management is even more important.
  - We moved version control to the forefront. We now **start** with version control and end with changes in puppet.
  - Version control is how we trigger a configuration change, not how we store a change we made.

SSEC System Components

- Puppet – configuration management server
- Gitlab– git server, with integrated Continuous Integration features (CI).
  - Includes gitlab-ci
  - CI Scripts you write do the work: bash, perl, python, whatever you want
- Foreman – Puppet GUI and node classifier
  - optional, but nice
- R10K
  - Code management tool that maps Git branches to Puppet environments.
  - Make a branch called ‘testing’ in git, and an puppet environment called ‘testing’ is created. Assign test machines to testing branch and test.

Puppet History at SSEC

2011, Initial: Puppet server for testing and light use

- Used to distribute science software rpms/lmod modules, yum repositories, and monitoring client checks
- Edit modules directly on server
- Assign modules to hosts in puppet server configuration files
- No integrated version control (ad-hoc for some modules)

February 2015, Configuration as Code: Puppet, Gitlab, gitlab hooks, and puppet-sync

- **Version Controlled**
- pdxcat/puppet-sync
  - Pre-receive hooks for tests, Post-receive hooks for puppet-sync
- branches are environments
- Git workflow accelerated development of modules
- Used for ‘everything’, new stuff and converting existing.
- Problem with git client and gitlab gui and hooks - tests were in pre-receive, never get triggered in GUI
- Monolithic gitlab repository bad for sharing with other groups

July 2016, Gitlab-CI method: Puppet, Gitlab, Gitlab-CI, and R10k

- The gitlab GUI is a first class citizen
- CI/CD methods are generally well understood and accepted
- Gitlab-CI provides a GUI, making it easier to visualize and maintain your production pipeline.

June 2017, Upgrade to Puppet 4

Foreman

- Foreman provides a GUI for Puppet, system provisioning and more.
- Foreman also provides the Puppet Server install
- To make new classes or environments available in Foreman, we import them via the GUI manually after our Gitlab/r10k/puppet workflow completes.

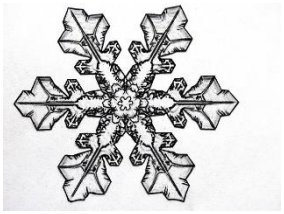
You will see “Foreman Import” in the CI steps, but this is just a placeholder in case we decide to automate further.

SSEC Workflow: Routine Administration

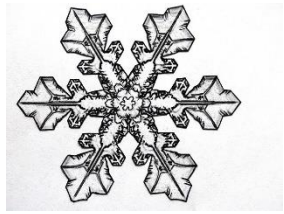
- **Node Classification** – we use Foreman as External Node Classifier
  - Assign nodes to environments, such as a test environment
  - Assign modules (puppet classes) to groups of nodes or individual nodes
- **Routine changes** can be done in master branch of control repository. We can use the Gitlab GUI or git cli.
  - User operations: add local users, override attributes (example: modify UID if needed), Allow access to a machine for a domain user.
  - Manage public keys
  - Sudoers
  - Add Applications

SSEC Workflow: Bigger Changes

1. Create short lived branches, for example “scott_test”
2. Assign test nodes to scott_test environment
3. Test changes
4. Merge test branches to master
5. Move test nodes back to master



Configuration Freeze



We typically use the system to enact change. We can also use the system to avoid changes. We did this for a group of critical systems for a week during GOES-16 deployment.

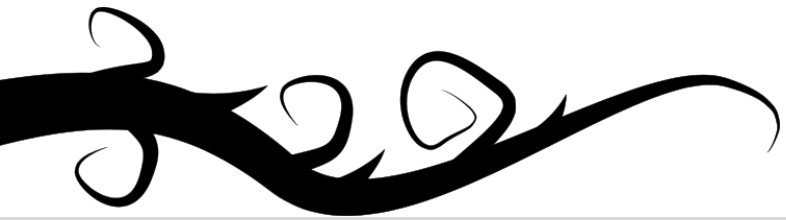
- Made a branch called ‘goes_freeze’, put hosts in that environment
- Pinned every puppet module to a specific commit
- Disabled automatic yum updates in our yum module
- When the freeze period was over, moved the hosts to the normal environment. **They then got all the changes that happened in that time period.**

In the past this may have caused us to simply pause some of our work, now it does not.

Long Lived Branch Workflow

Our short lived branch workflow is only one way to do this. Another idea is to have a long lived test branch.

- Force all changes to go to the test branch and merge to master (or ‘production’) after passing
- Have some hosts permanently assigned to the test environment
- This can be a highly structured workflow with checks and balances. Only some admins could be allowed rights to merge for example.
- These methods can be combined. Permanent testing VM’s for a long lived branch, but still doing the short-lived branch test methods is a direction we may go at SSEC. I especially like the idea of some nightly complete system rebuild testing.



Where did I get the idea for puppet/r10k/gitlab?

This is based on first looking at Phil Zimmerman's
'An Automated R10K Workflow That Works' and attempting to replicate it.
http://philzim.com/2014/05/02/an-automated-r10k-workflow-that-works/

This uses

- R10k
- Github Enterprise
- Sinatra
- Capistrano
- Jenkins
- Ruby
- https://github.com/pzim/reaktor

As I was wrestling with various components gitlab-ci developed rapidly and showed great promise. It worked.

R10K: Control Repository and Puppetfile

The control (git) repository controls the puppet environments, it is a critical component. The control repository contains the Puppetfile and more.

The puppetfile is a file listing all other repositories with modules you use. These can be any combination of your own git or puppet forge, your own private, or someone else's. For security we use a dedicated gitlab server, and download, curate and re-host any repositories we use from outside.

Consider that you likely would not share your control repository with others, but may share other modules, which are building blocks.

```
#Puppetfile example
#Here are some Puppet Modules. These reference our git server, but
#could be any git or puppet forge
mod 'ipmi',
  :git => 'git@sgit.ssec.wisc.edu:tc-puppet/ipmi.git',
  :ref => 'master'
mod 'winbind',
  :git => 'git@sgit.ssec.wisc.edu:tc-puppet/winbind.git',
  :ref => 'master'
mod 'augeasproviders_core',
  :git => 'git@sgit.ssec.wisc.edu:tc-puppet/augeasproviders_core.git',
  :ref => 'master'
mod 'zpool_scrub',
  :git => 'git@sgit.ssec.wisc.edu:tc-puppet/zpool_scrub.git',
  :ref => 'master'
```

R10K: Control Repository Contents

Anything that is routinely changed is more convenient in control than an external repository. You could have a mostly empty puppetfile and put most puppet modules in control, mimicking our monolithic puppet-sync method.

- **hieradata** – this can be it's own module, but for SSEC, is changed frequently. For example, it's how we allow users login to linux systems. I believe hieradata in control is fairly common.
- **site/<modulename>** – we put the 'site' directory is in the modulepath, so this adds modules in the control repo directly
  - You can put any modules you want in site/
  - We use - site/profiles, site/role, and site/hostgroups. Notice these are generally 'site specific' things for SSEC, likely would not be shared. They're how we compose things, using other basic building block modules from outside control repo, as defined in the Puppetfile.

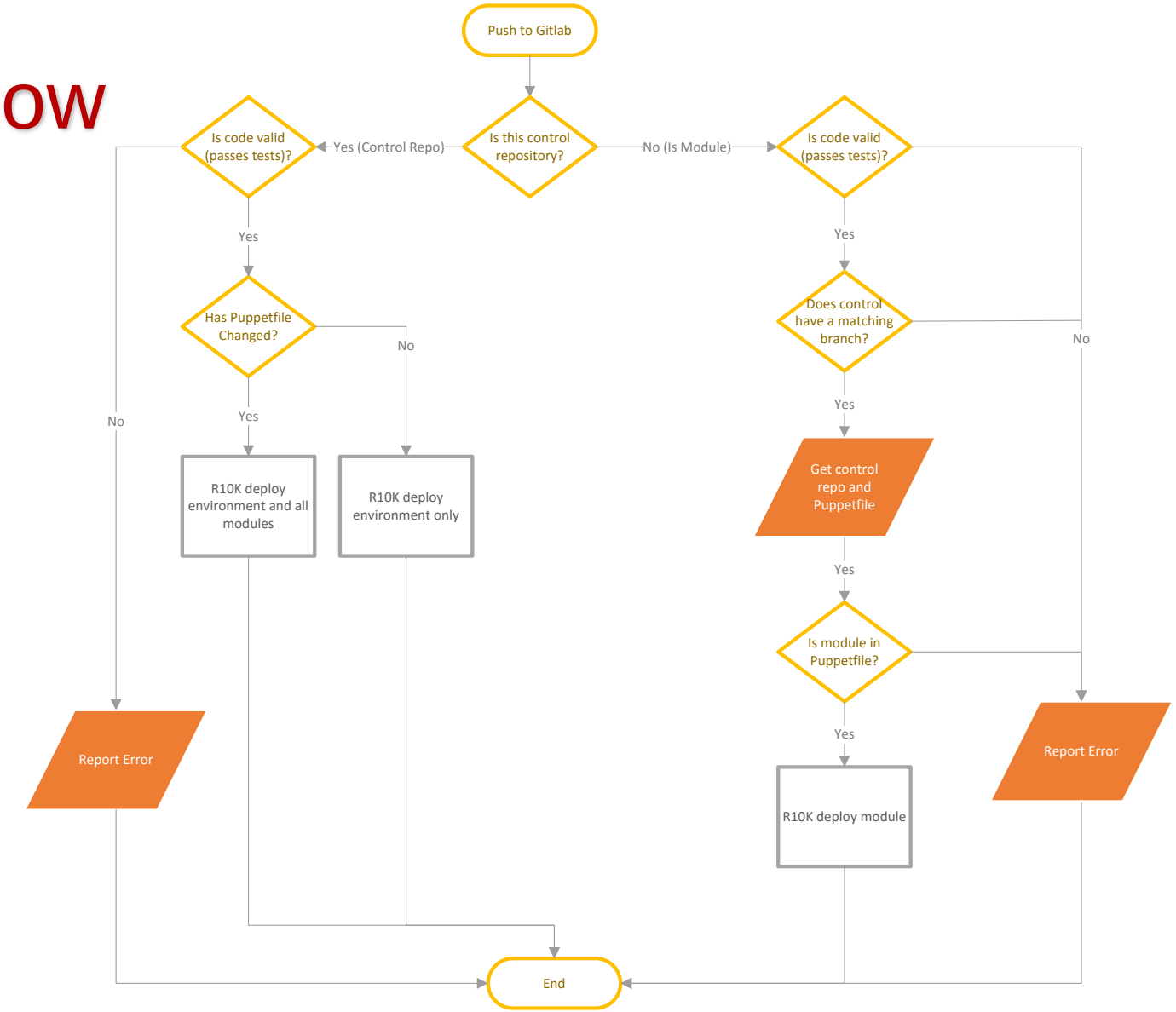
Gitlab CI

Gitlab CI is an integrated CI/CD system that comes with gitlab for testing, building, and deploying code

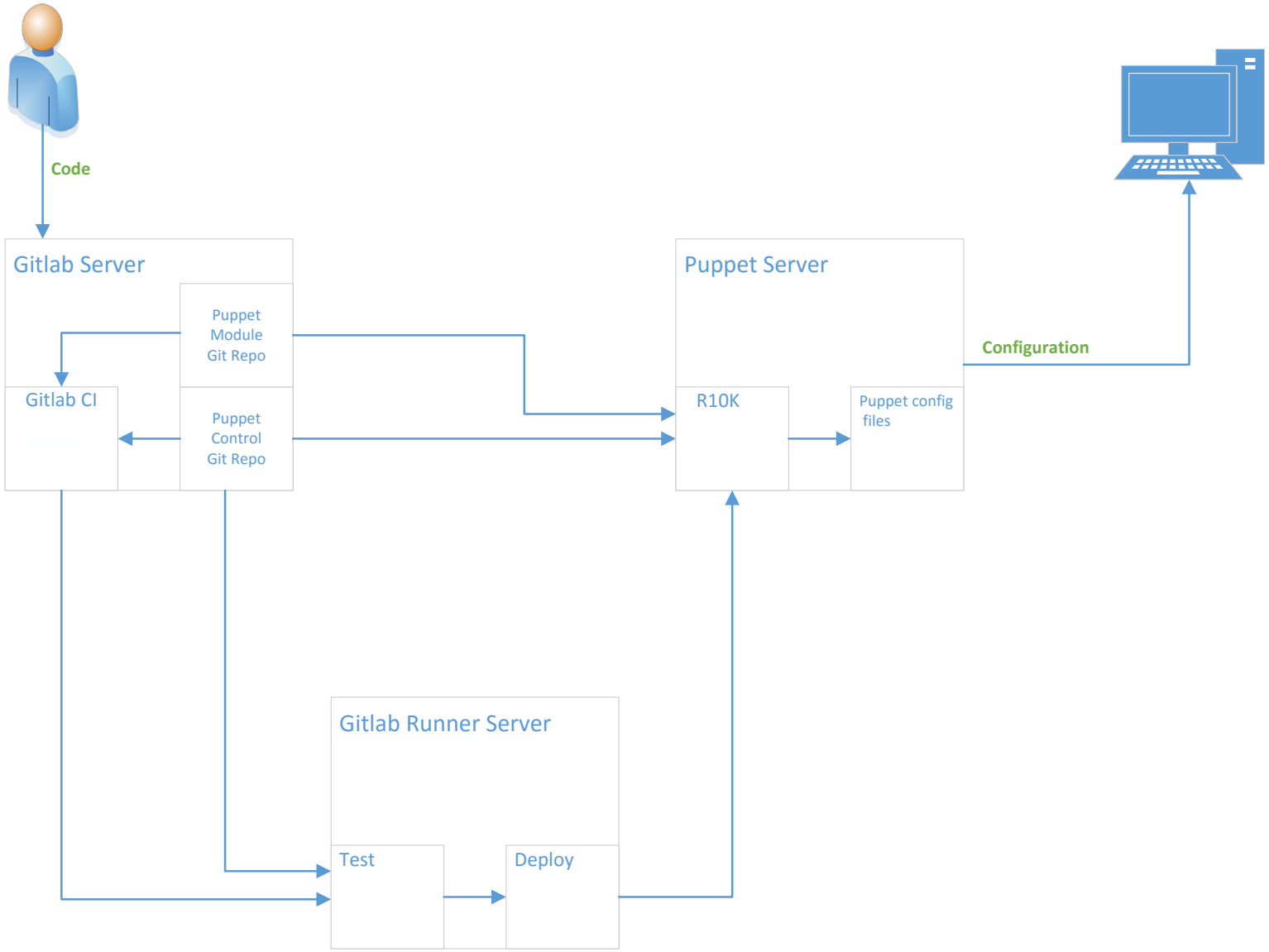
- Continuous Integration (CI) is a development practice where you integrate code for a project frequently. Check-ins are tested via an automatic build process.
- Continuous Deployment (CD) is similar, focused on delivery of software to the end user.

In our use case, we test our puppet modules, and deploy them to the puppet server. We use Gitlab-CI and R10K instead of our previous git server-side hooks and pdxcat/puppet-sync.

Process Flow



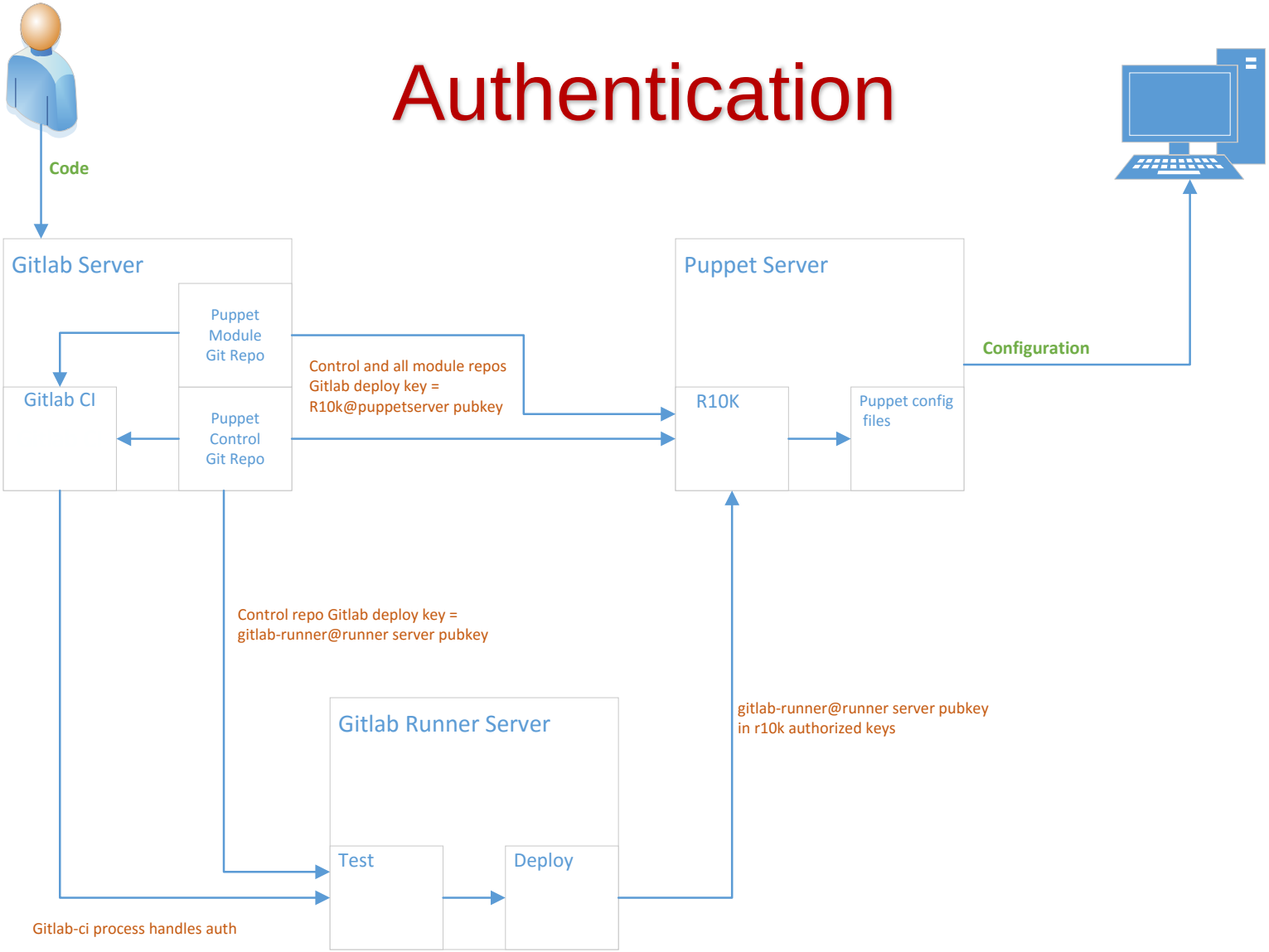
Data Flow


Code
Gitlab Server
Puppet Module Git Repo
Gitlab CI
Puppet Control Git Repo
Puppet Server
R10K
Puppet config files
Configuration
Gitlab Runner Server
Test
Deploy

Authentication Between Components

- **Gitlab deploy keys** – allow read-only access to a gitlab repository
  - R10K user @ puppet server's ssh public key is used as gitlab deploy key on control and module repos. This is how r10k@puppet_server grabs the repositories from gitlab.
  - Gitlab-runner user @ runner server's ssh public key is used as gitlab deploy key on the control repository only. The runner uses this to get the puppetfile for tests.
- **Normal system SSH keys**
  - Gitlab-runner@runner_server ssh public key is added to r10k@puppet_server authorized keys. This is how the gitlab runner can ssh to the puppet server to run r10k commands.
  - We manage public keys with puppet, adding them to authorized_keys. Of course there's the bootstrap issue for these in particular.

Authentication



Gitlab CI: Control Repository

Adding .gitlab-ci.yml file to the repo defines the process

```
stages:
  - test
  - r10k

puppet_test:
  stage: test
  script: puppet_test
  allow_failure: false

hiera_test:
  stage: test
  script: hiera_test
  allow_failure: false

r10kdeploy:
  stage: r10k
  script: r10k_control_deploy
  allow_failure: false
```

Scripts are run on the gitlab-runner linux host. There are docker and shell runner methods, we use shell, but either will work.

The gitlab-runner user simply executes whatever script you give it in it's shell.

This naturally has security concerns, the gitlab-runner needs to be treated as securely as all other system components. If compromised, all your hosts are compromised easily.

Pipeline #1615 with 4 builds for master in 6 seconds

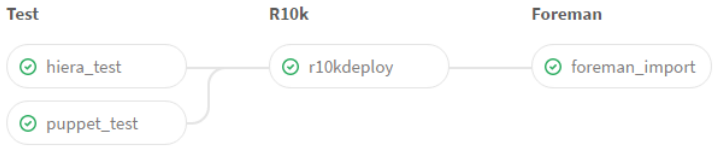
passed

Authored by **Scott Nolin** 9 minutes ago

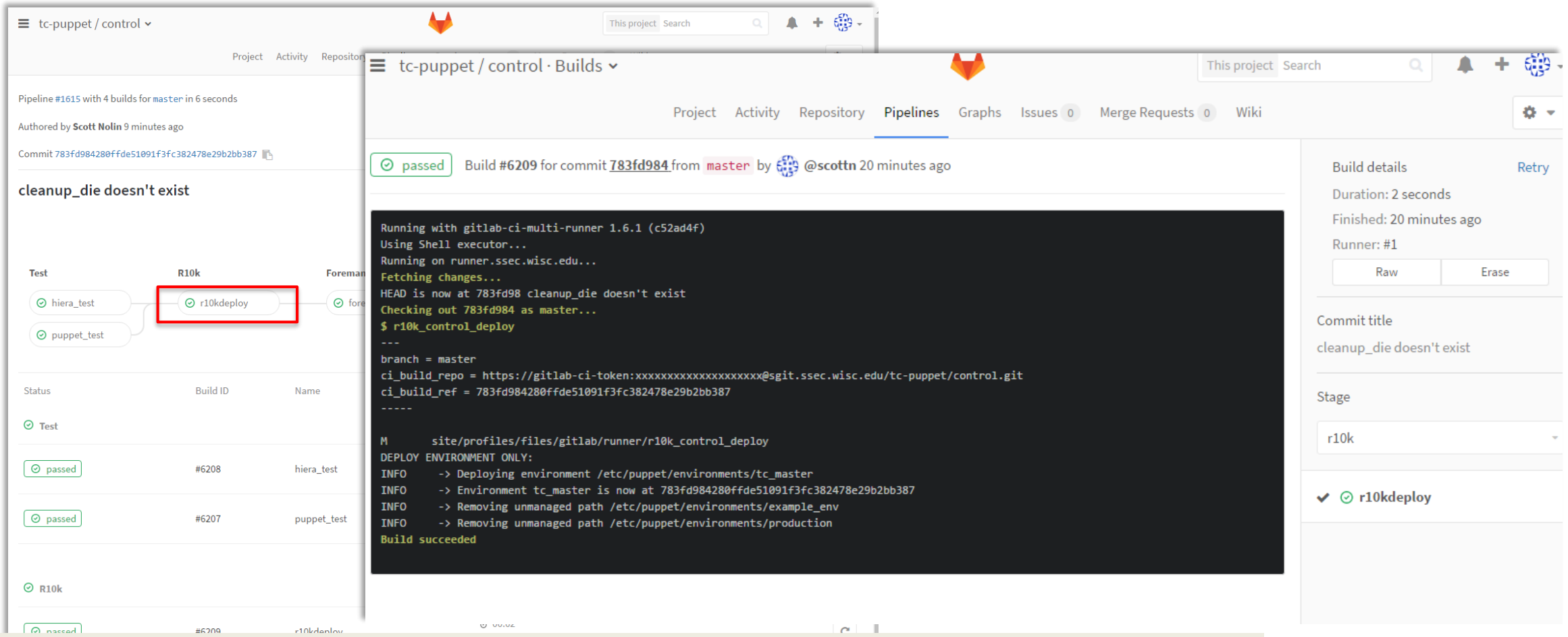
Commit 783fd984280ffde51091f3fc382478e29b2bb387

cleanup_die doesn't exist

Hide pipeline graph



Status	Build ID	Name		
Test				
passed	#6208	hiera_test	00:01 less than a minute ago	
passed	#6207	puppet_test	00:01 less than a minute ago	
R10k				
passed	#6209	r10kdeploy	00:02 less than a minute ago	
Foreman				
passed	#6210	foreman_import	00:01 less than a minute ago	



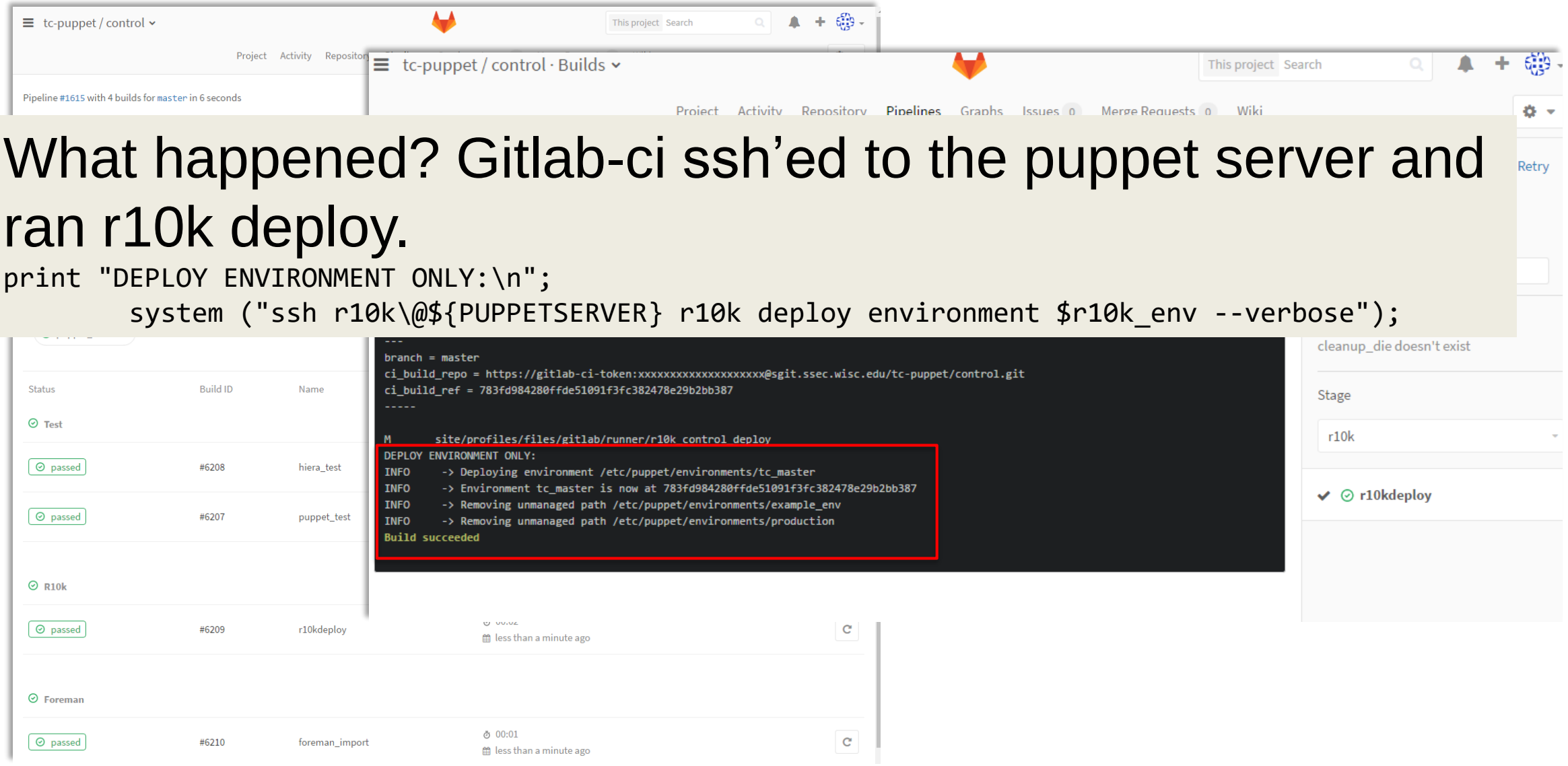
Clicking the CI Step shows the output.

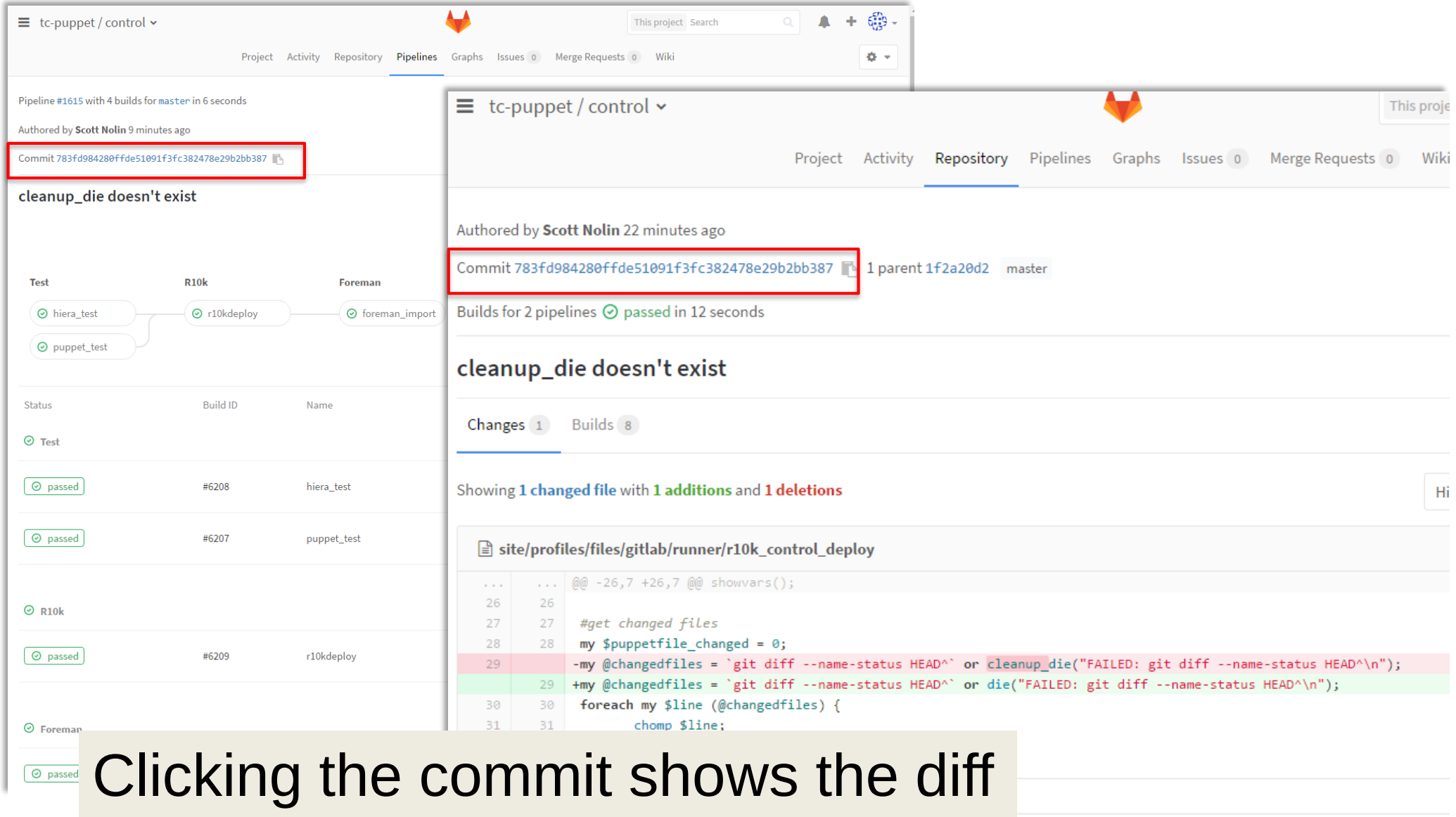
In this case I modified r10k_control_deploy itself.

I found a bug while making these slides.

What happened? Gitlab-ci ssh'ed to the puppet server and ran r10k deploy.

```
print "DEPLOY ENVIRONMENT ONLY:\n";
        system ("ssh r10k\@${PUPPETSERVER} r10k deploy environment $r10k_env --verbose");
```





Clicking the commit shows the diff

Gitlab CI: All Other Modules

The .gitlab-ci.yml file for modules is slightly different

```
stages:
  - test
  - r10kdeploy

puppet_test:
  stage: test
  script: puppet_test
  allow_failure: false

r10K_modtest:
  stage: test
  script: r10k_modtest
  allow_failure: false

r10k_moddeploy:
  stage: r10kdeploy
  script: r10k_moddeploy
  allow_failure: false
```

Our general puppet tests are the same, and notice we don't test hieradata – that lives in control only.

We do module specific tests and deployment.

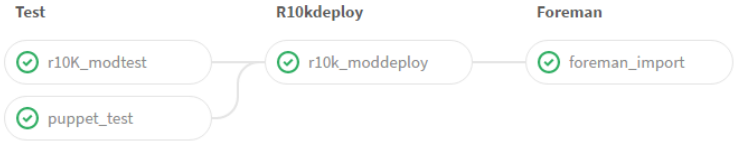
Pipeline #1729 with 4 builds for master in 3 seconds (queued for 1 second) — passed

Authored by **Jesse Stroik** about an hour ago

Commit 62a5768c95eff6b54ac3a41705a4e0c98e4010ba

replacing the silent.cfg files

Hide pipeline graph



Status	Build ID	Name		
Test				
passed	#6618	puppet_test	00:00 about an hour ago	
passed	#6619	r10K_modtest	00:01 about an hour ago	
R10kdeploy				
passed	#6620	r10k_moddeploy	00:01 about an hour ago	
Foreman				
passed	#6621	foreman_import	00:00 about an hour ago	

Gitlab CI Script Details

Gitlab-CI scripts are the glue that make the process work.

I designed the test scripts to also work as pre-commit scripts

- This avoids running pipelines that will fail tests and speeds up workflow
- Pre-commit scripts only work at the git cli

Example Scripts are here: https://gitlab.ssec.wisc.edu/scottn/pgr10k

- We are still changing these periodically, and Gitlab is in active development, so will not include the in these slides.

Discussion