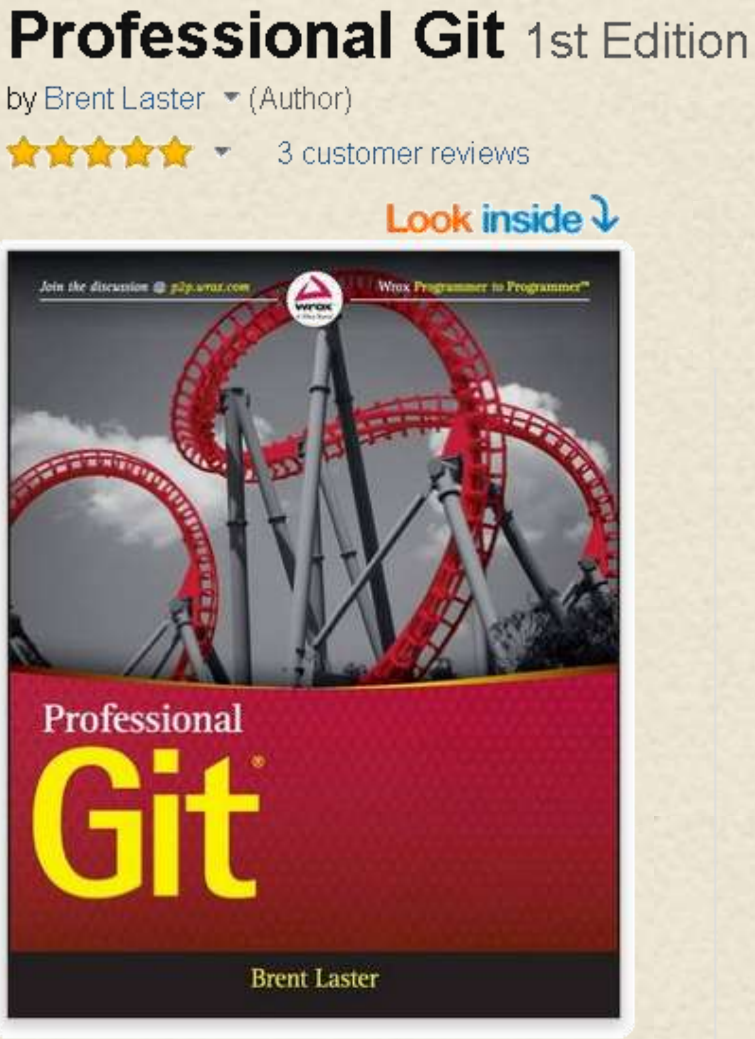
Jenkins 2 – Coding Continuous Delivery Pipelines

Brent Laster

About me

- Senior Manager, R&D
- Global trainer – training (Git, Jenkins, Gradle, Gerriit, Continuous Pipelines)
- Author -
  - NFJS magazine – series on Gerrit
  - Professional Git book
  - Jenkins 2 – Up and Running book
  - Continuous Integration vs. Continuous Delivery vs. Continuous Deployment mini-book on Safari
- https://www.linkedin.com/in/brentlaster
- @BrentCLaster

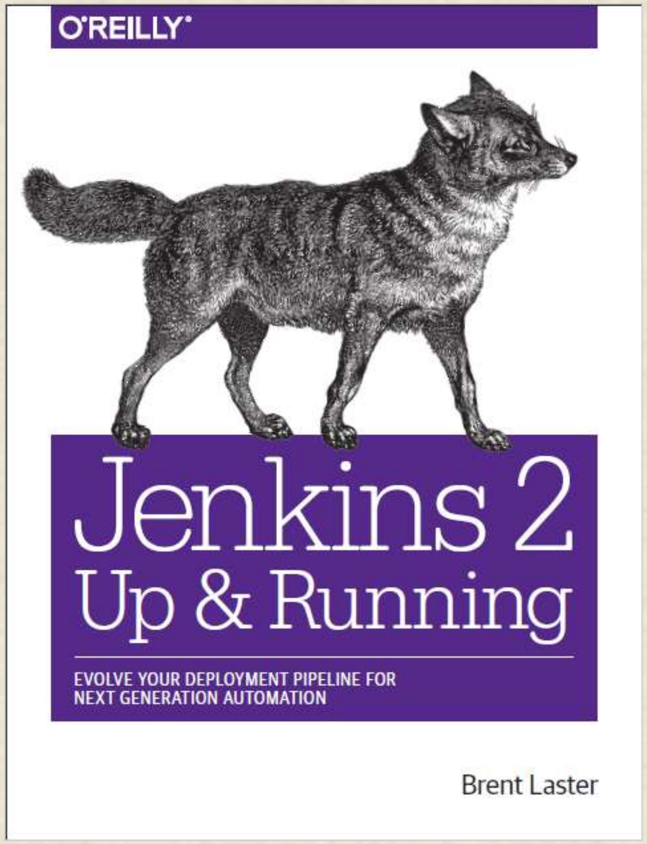
Books


Professional Git 1st Edition

by Brent Laster (Author)

★★★★★ 3 customer reviews

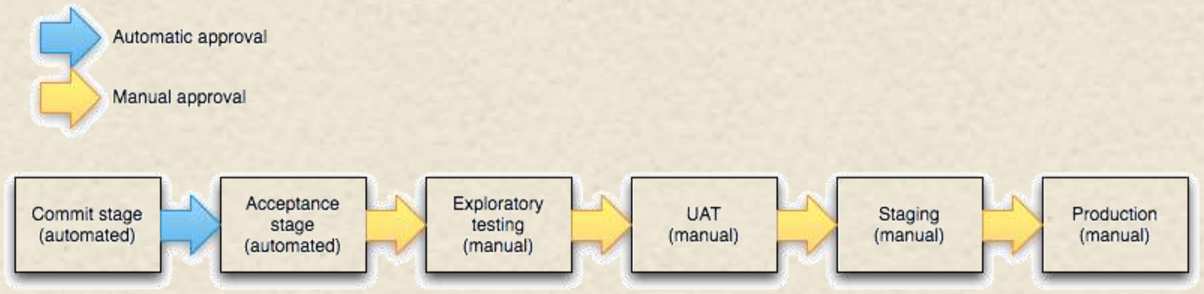
Look inside
Join the discussion @ p2p.wrox.com
Wrox Programmer to Programmer™
Professional
Git
Brent Laster


O'REILLY
Jenkins 2
Up & Running
EVOLVE YOUR DEPLOYMENT PIPELINE FOR NEXT GENERATION AUTOMATION
Brent Laster

- First 4 chapters available on Safari at https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/
- Full book available late April

Continuous Delivery Pipeline

- "… an automated implementation of your application's build, deploy, test, and release process.

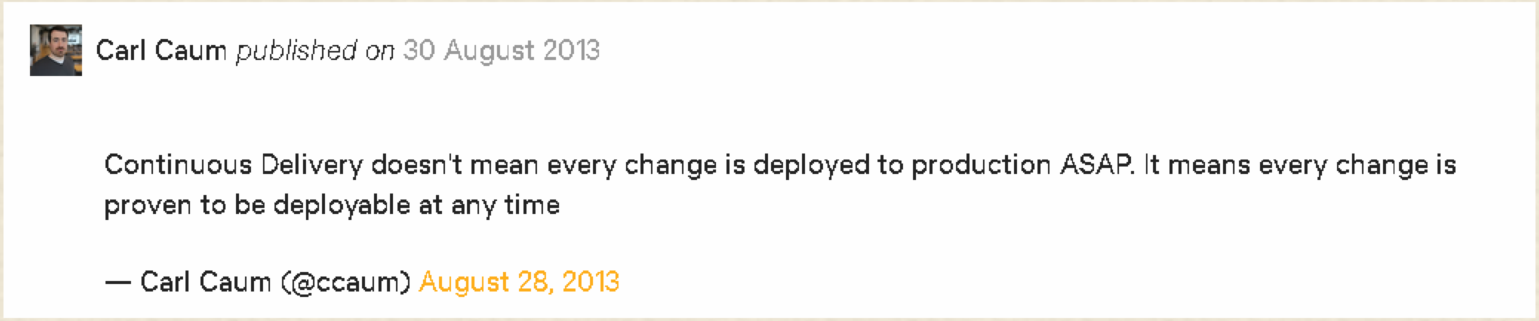


- Every change made to configuration, source code, environment or data triggers a new instance of the pipeline.
- Change motivates production of binaries and then a series of tests are done to prove it is releasable.
- Levels of testing provide successive levels of confidence.

Continuous Pipelines

Practice

- Theme is automation of software production process
- Combines 3 core practices/disciplines
  - Continuous Integration
  - Continuous Delivery
  - Continuous Deployment (if desired)
- Includes Configuration Management



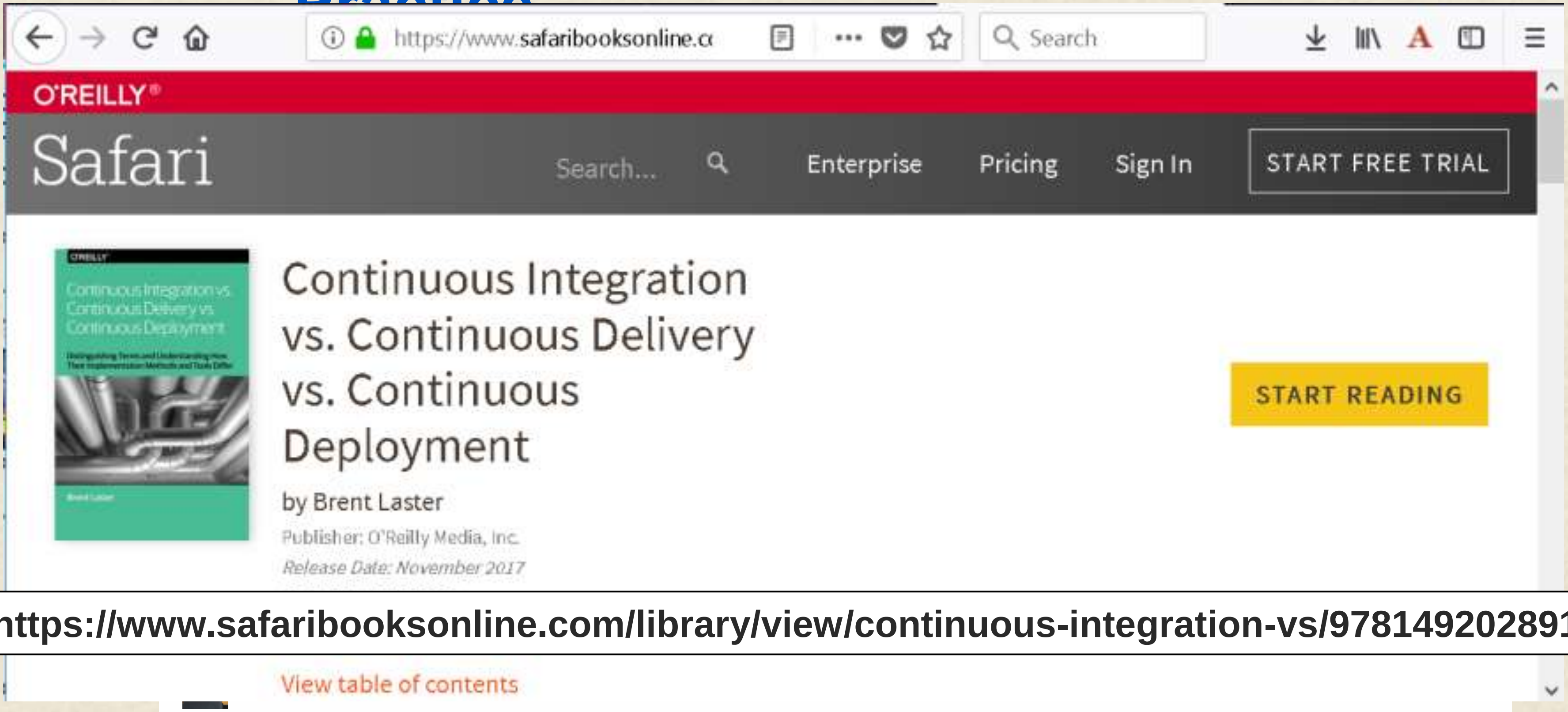
Carl Caum *published on* 30 August 2013

Continuous Delivery doesn't mean every change is deployed to production ASAP. It means every change is proven to be deployable at any time

— Carl Caum (@ccaum) August 28, 2013

Continuous Pipelines

Practice



https://www.safaribooksonline.com/library/view/continuous-integration-vs/9781492028918/

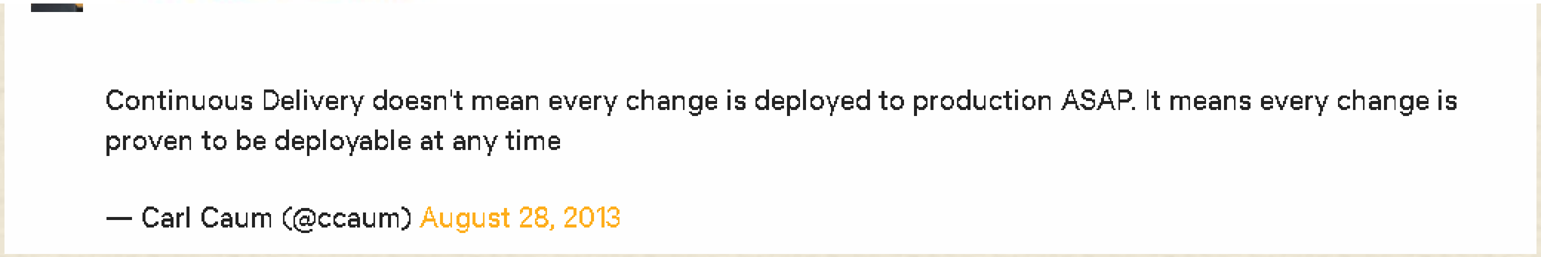
Continuous Integration vs. Continuous Delivery vs. Continuous Deployment

by Brent Laster

Publisher: O'Reilly Media, Inc.

Release Date: November 2017

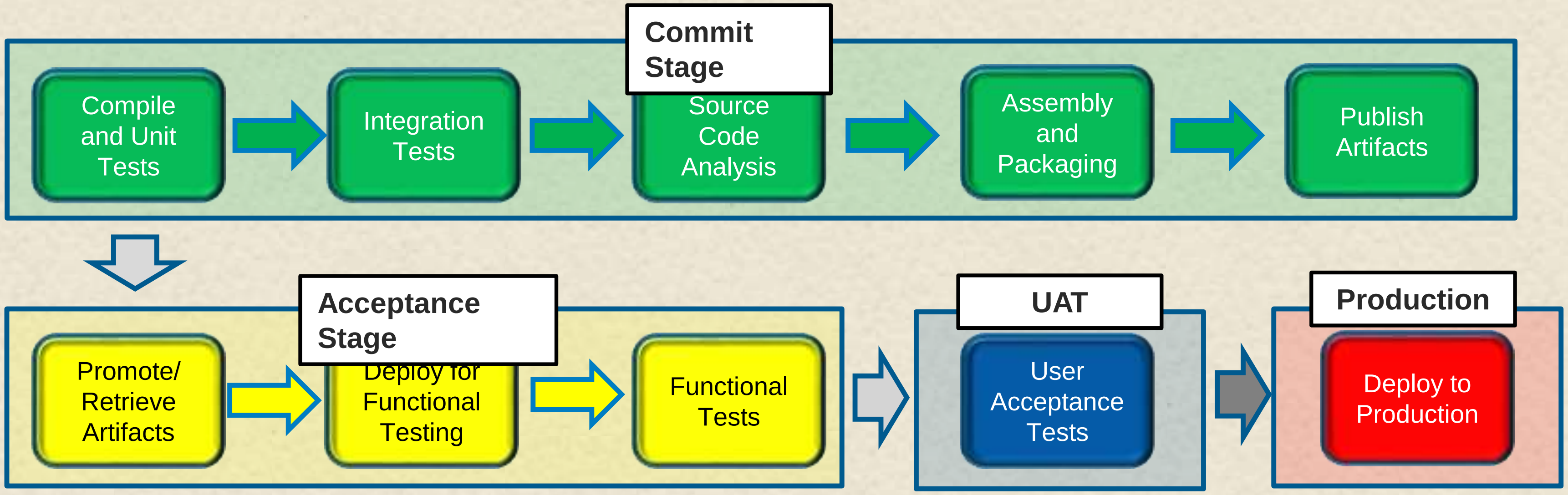
View table of contents



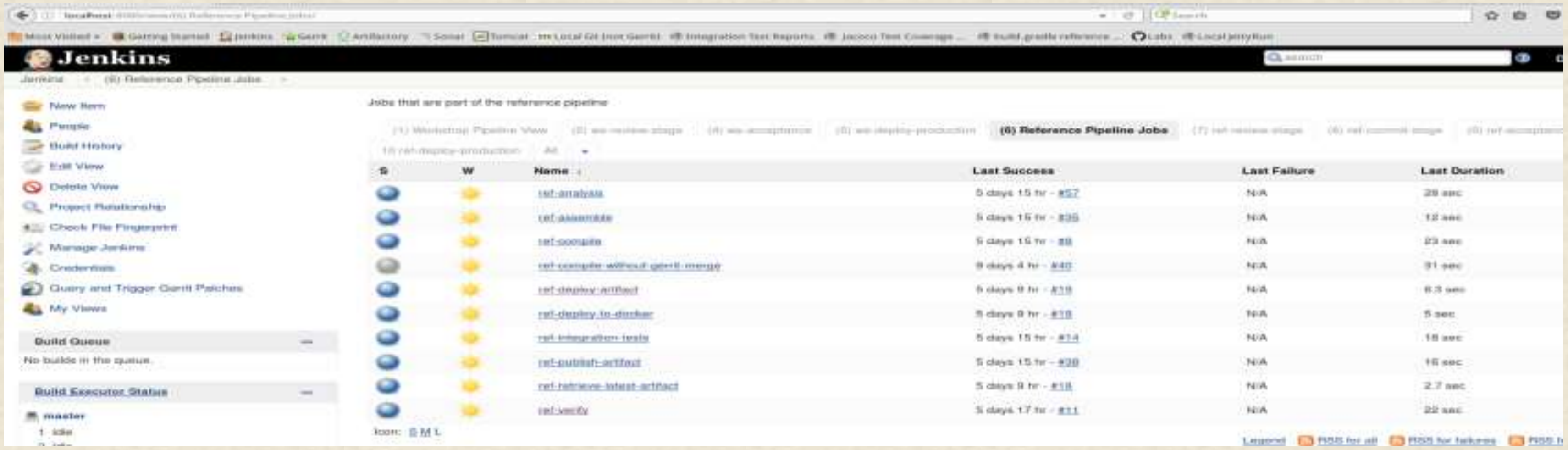
Continuous Delivery doesn't mean every change is deployed to production ASAP. It means every change is proven to be deployable at any time

— Carl Caum (@ccaum) August 28, 2013

Continuous Delivery Pipeline Stages



Popular way to implement this has been as a series of Jenkins jobs .



Jobs that are part of the reference pipeline

(6) Reference Pipeline Jobs

S	W	Name	Last Success	Last Failure	Last Duration
●	☀	ref-analysis	5 days 15 hr - #57	N/A	28 sec
●	☀	ref-assemble	5 days 15 hr - #35	N/A	12 sec
●	☀	ref-compile	5 days 15 hr - #8	N/A	23 sec
●	☀	ref-compile-without-gerrit-merge	9 days 4 hr - #40	N/A	31 sec
●	☀	ref-deploy-artifact	5 days 9 hr - #19	N/A	6.3 sec
●	☀	ref-deploy-to-docker	5 days 9 hr - #18	N/A	5 sec
●	☀	ref-integration-tests	5 days 15 hr - #14	N/A	18 sec
●	☀	ref-publish-artifact	5 days 15 hr - #38	N/A	16 sec
●	☀	ref-retrieve-latest-artifact	5 days 9 hr - #18	N/A	2.7 sec
●	☀	ref-verify	5 days 17 hr - #11	N/A	22 sec

Icon: S M L

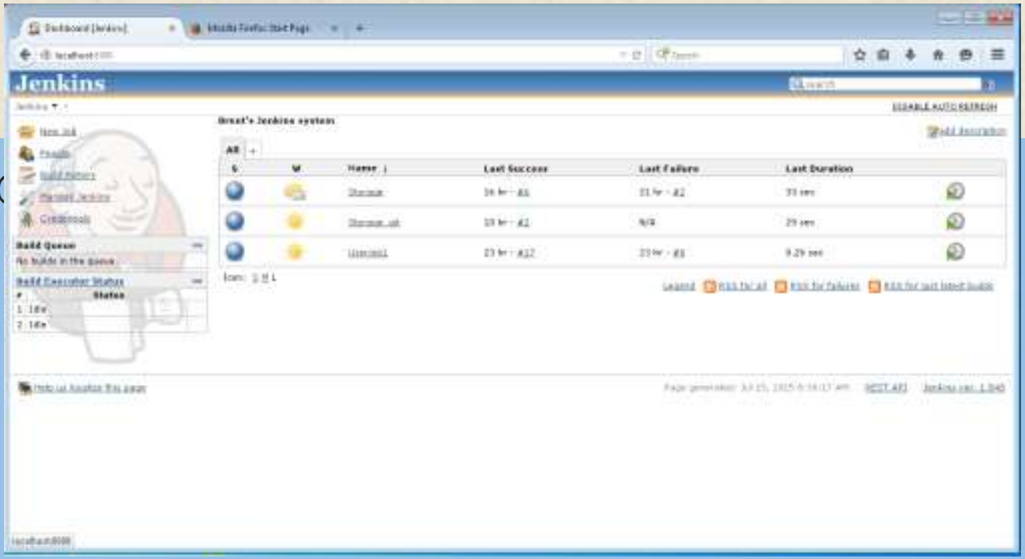
Legend RSS for all RSS for failures

About Jenkins – What is it?

- Open-source framework for defining, running, and monitoring jobs/projects
- Jobs/projects execute processes – such as builds, tests, etc.
- Allows "global" configuration of available tools, servers, etc. and "local" configuration within jobs to execute processes using those tools
- Frequently used to define a sequence of processes to construct a deployment pipeline
- Created by Kohsuke Kawaguchi
- Formerly known as Hudson
- Over 1000 plug-ins that provide functionality
- https://jenkins.io
- Enterprise version supported by Cloudbees
- Free community version supported by users and Cloudbees

The Jenkins “Object Model”

Jenkins Dashb



Jenkins

DISABLE AUTO REFRESH

Brent's Jenkins system

add description

All +

S	W	Name ↓	Last Success	Last Failure	Last Duration	
●	☀	[illegible]	16 hr - #1	21 hr - #2	33 sec	
●	☀	[illegible]	19 hr - #2	N/A	29 sec	
●	☀	[illegible]	23 hr - #12	23 hr - #8	9.29 sec	

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

Build Queue

No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

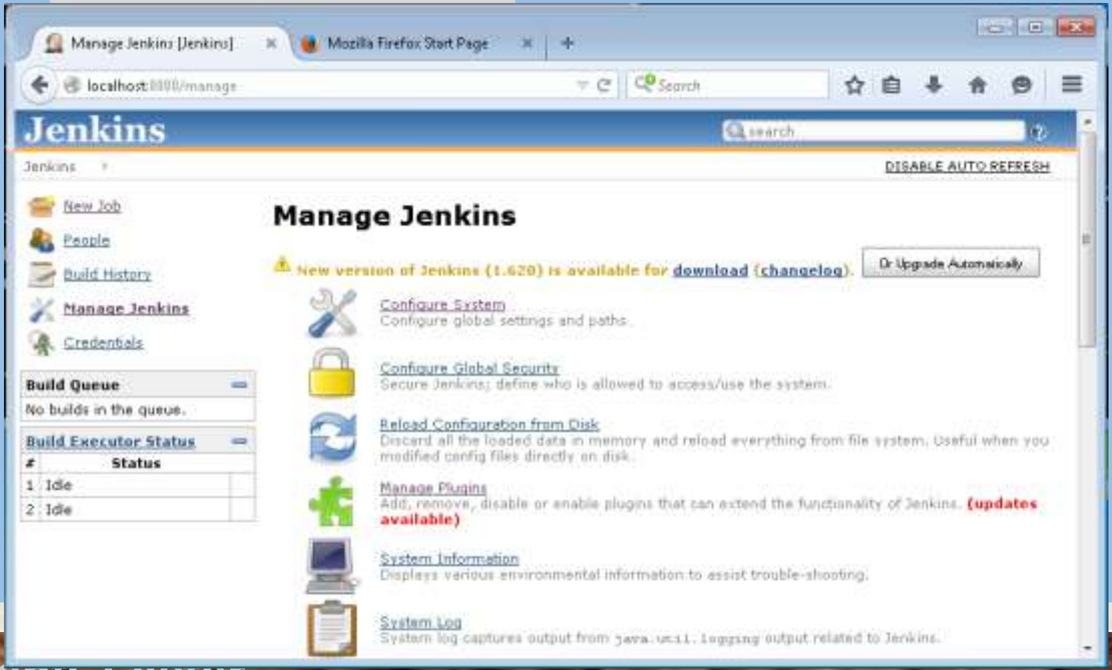
Help us localize this page

Page generated: Jul 15, 2015 [illegible] AM REST API Jenkins ver. 1.[illegible]

The Jenkins “Object Model”

Jenkins Dashboard (Latest Build Jobs Statuses)

Global Management



Build Queue

No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

Manage Jenkins

New version of Jenkins (1.620) is available for download (changelog).

Configure System
Configure global settings and paths.

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

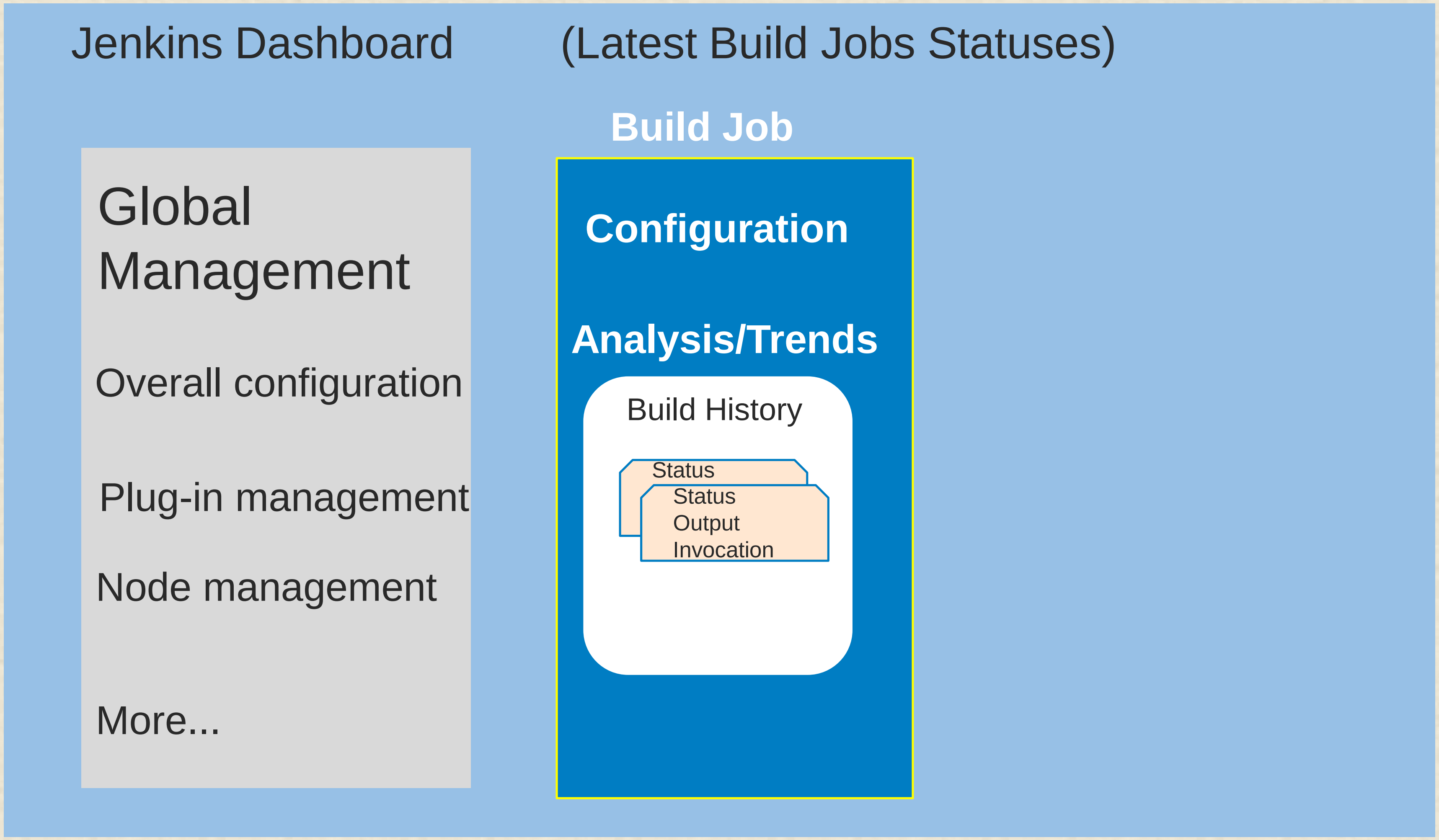
Reload Configuration from Disk
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins. **(updates available)**

System Information
Displays various environmental information to assist trouble-shooting.

System Log
System log captures output from java.util.logging output related to Jenkins.

The Jenkins “Object Model”



The Jenkins “Object Model”

Jenkins Dashboard (Latest Build Jobs Statuses)

Build Job

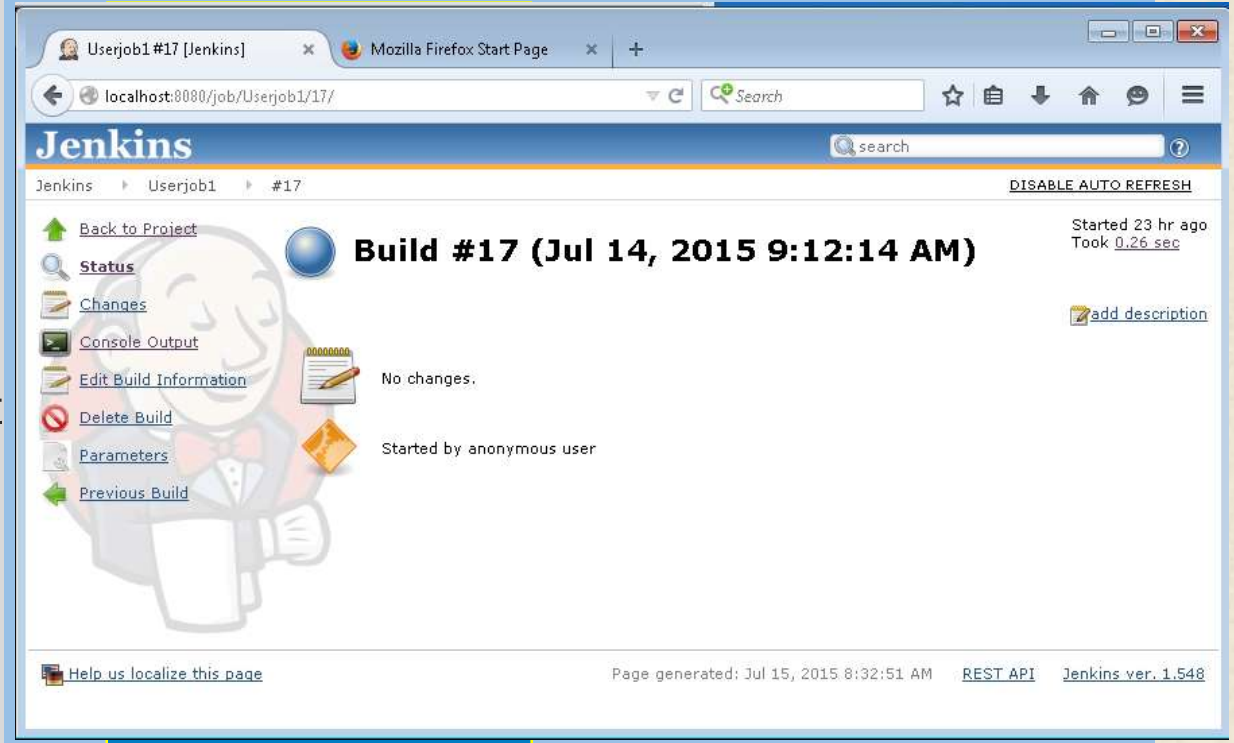
Global Management

Overall configuration

Plug-in management

Node management

More...


Userjob1 #17 [Jenkins]
Mozilla Firefox Start Page
localhost:8080/job/Userjob1/17/

Jenkins » Userjob1 » #17 — DISABLE AUTO REFRESH

Back to Project
Status
Changes
Console Output
Edit Build Information
Delete Build
Parameters
Previous Build

Build #17 (Jul 14, 2015 9:12:14 AM)

Started 23 hr ago
Took 0.26 sec

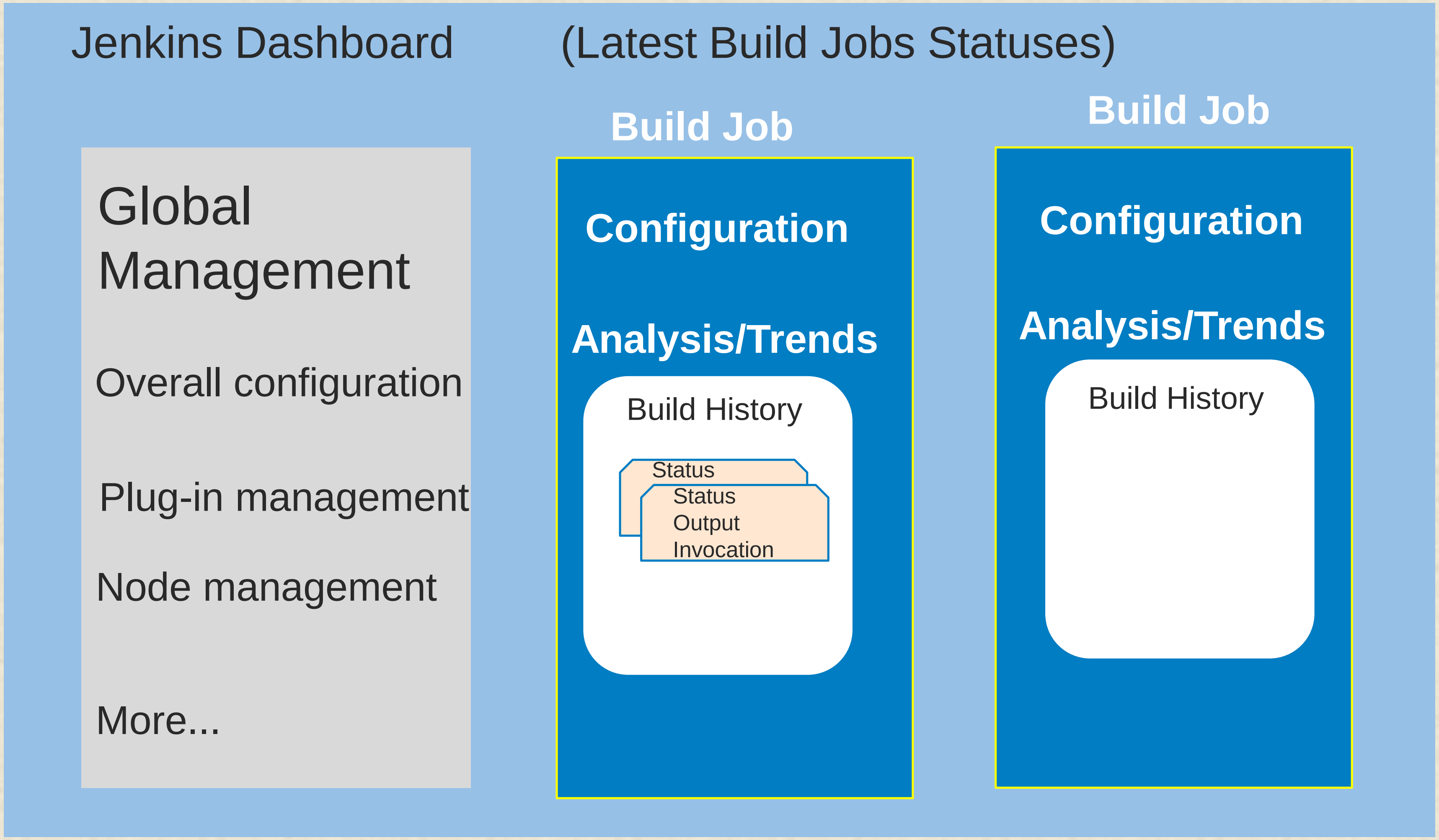
add description

No changes.

Started by anonymous user

Help us localize this page — Page generated: Jul 15, 2015 8:32:51 AM — REST API — Jenkins ver. 1.548

The Jenkins “Object Model”



The Jenkins "Object Model"

Jenkins Dashboard (Latest Build Jobs Statuses)

Build Job

Configuration

Build Job

Configuration

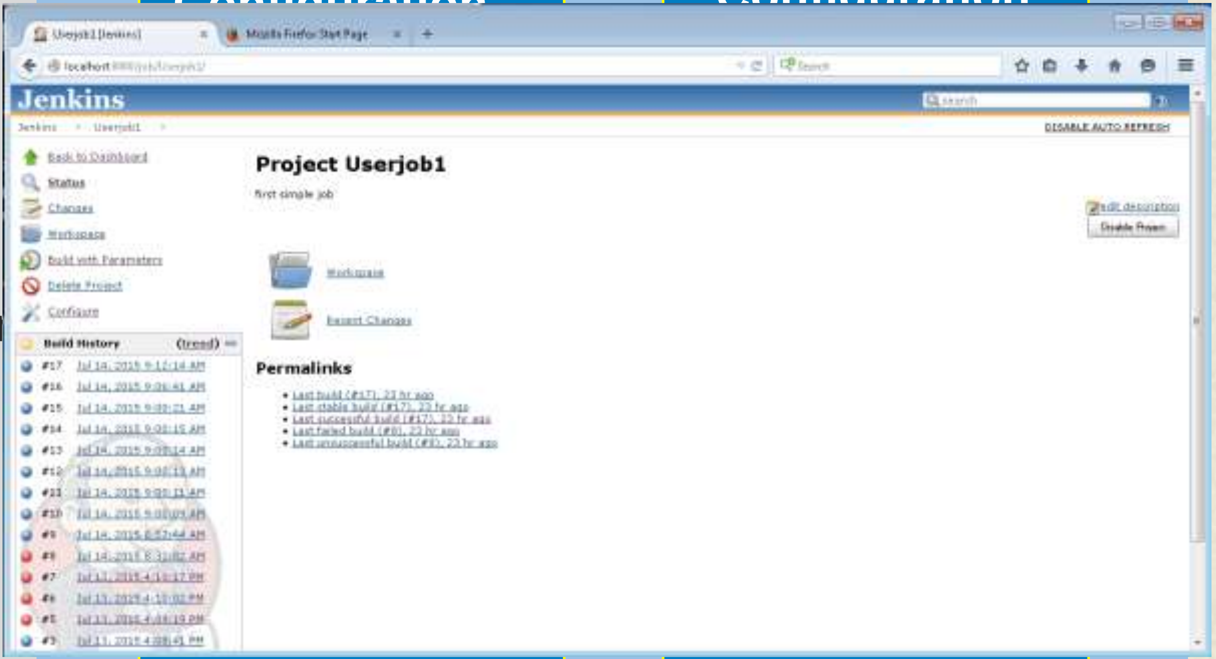
Global Management

Overall configuratio

Plug-in manageme

Node management

More...



Userjob1 [Jenkins] | Mozilla Firefox Start Page

localhost:8080/job/Userjob1/

Jenkins

Jenkins » Userjob1 »

DISABLE AUTO REFRESH

- Back to Dashboard
- Status
- Changes
- Workspace
- Build with Parameters
- Delete Project
- Configure

Build History (trend)

#17 Jul 14, 2015 9:12:14 AM
#16 Jul 14, 2015 9:06:41 AM
#15 Jul 14, 2015 9:03:21 AM
#14 Jul 14, 2015 9:02:15 AM
#13 Jul 14, 2015 9:00:14 AM
#12 Jul 14, 2015 9:00:13 AM
#11 Jul 14, 2015 9:00:11 AM
#10 Jul 14, 2015 9:00:09 AM
#9 Jul 14, 2015 8:57:44 AM
#8 Jul 14, 2015 8:31:02 AM
#7 Jul 13, 2015 4:18:17 PM
#6 Jul 13, 2015 4:18:02 PM
#5 Jul 13, 2015 4:14:19 PM
#3 Jul 13, 2015 4:08:41 PM

Project Userjob1

first simple job

edit description

Disable Project

Workspace

Recent Changes

Permalinks

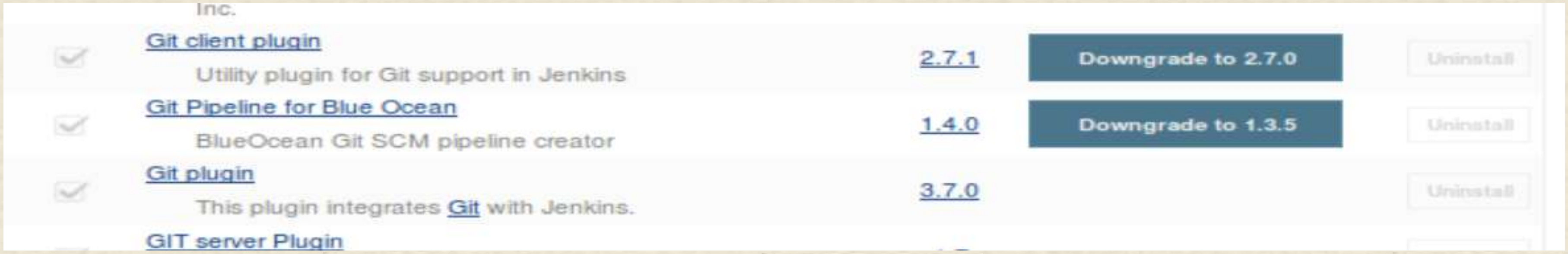
- Last build (#17), 23 hr ago
- Last stable build (#17), 23 hr ago
- Last successful build (#17), 23 hr ago
- Last failed build (#8), 23 hr ago
- Last unsuccessful build (#8), 23 hr ago

Jenkins Structure

- Basic Jenkins Hierarchy is:
  - Dashboard
    - » Individual project
      - » Individual build results for a project
- Each one may have subsections
  - Examples:
    - » build project may have configure and trend information
    - » build results have status and console output
- Supporting pieces
  - Jenkins global configuration (Manage Jenkins)
- Additional functionality added via plugins
- Most functionality has global configuration and local (job-level) steps that can be executed
- **Define globally, select locally**
  - May be multiple instances (different versions)
  - Then, for individual jobs (projects), you select the instance/version of thing you want from ones globally defined

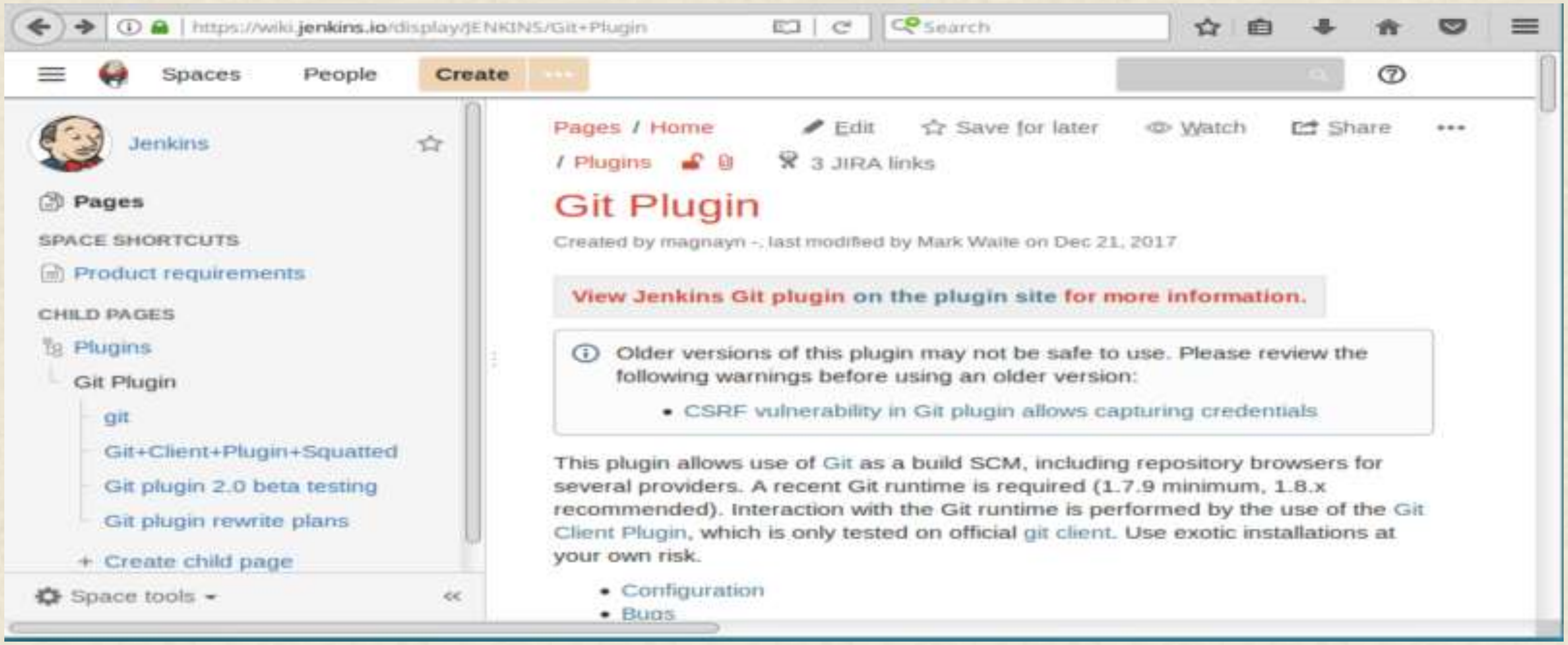
Example: Integration with Git (Plugin)

- Install plugin (via Manage Jenkins->Manage Plugins)



Inc.

Git client plugin Utility plugin for Git support in Jenkins	2.7.1	Downgrade to 2.7.0	Uninstall
Git Pipeline for Blue Ocean BlueOcean Git SCM pipeline creator	1.4.0	Downgrade to 1.3.5	Uninstall
Git plugin This plugin integrates Git with Jenkins.	3.7.0		Uninstall
GIT server Plugin			



https://wiki.jenkins.io/display/JENKINS/Git+Plugin

Git Plugin

Created by magnayn -, last modified by Mark Waite on Dec 21, 2017

View Jenkins Git plugin on the plugin site for more information.

Older versions of this plugin may not be safe to use. Please review the following warnings before using an older version:

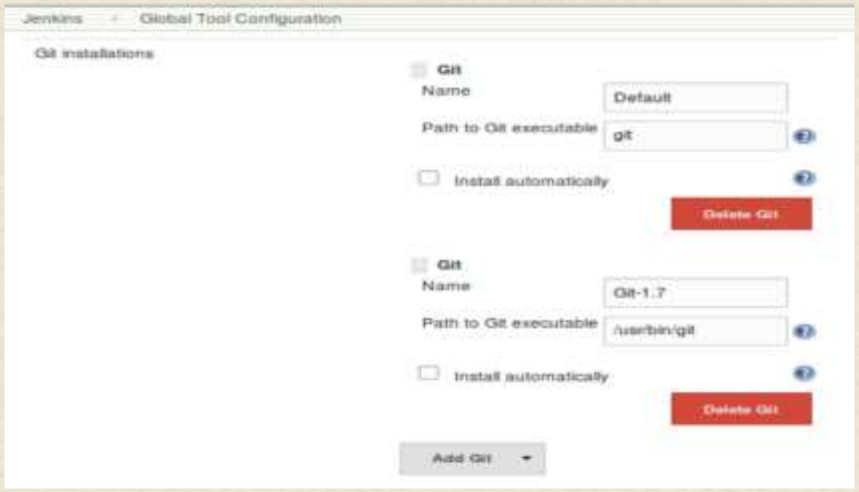
- CSRF vulnerability in Git plugin allows capturing credentials

This plugin allows use of Git as a build SCM, including repository browsers for several providers. A recent Git runtime is required (1.7.9 minimum, 1.8.x recommended). Interaction with the Git runtime is performed by the use of the Git Client Plugin, which is only tested on official git client. Use exotic installations at your own risk.

- Configuration
- Bugs

Example: Integration with Git (Job)

Global Config (Manage Jenkins)



Jenkins › Global Tool Configuration

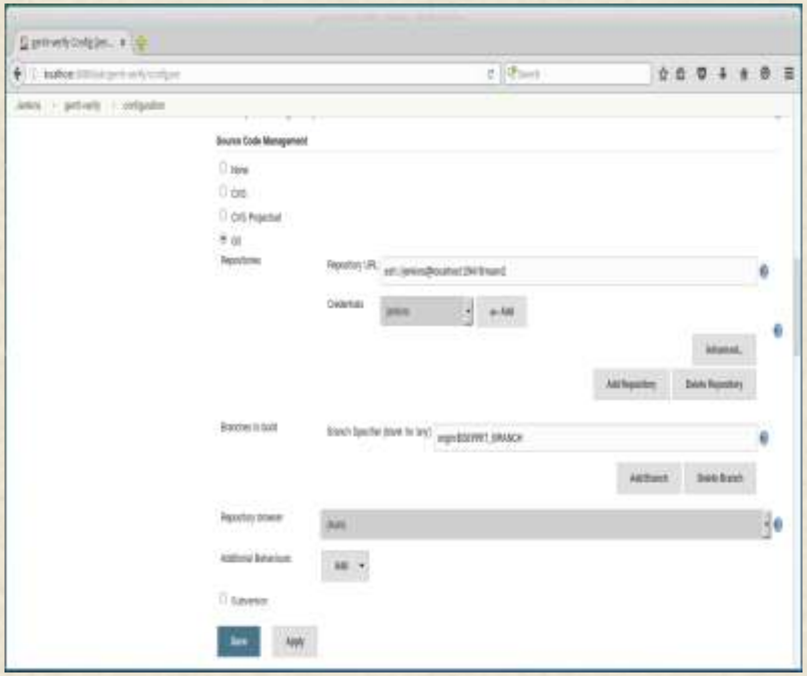
Git installations

Git
Name: Default
Path to Git executable: git
☐ Install automatically
Delete Git

Git
Name: Git-1.7
Path to Git executable: /usr/bin/git
☐ Install automatically
Delete Git

Add Git

Use Locally (Job)



Jenkins › gerrit-verify › configuration

Source Code Management

○ None
○ CVS
○ CVS Projectset
● Git

Repositories
Repository URL: ssh://jenkins@localhost:29418/[illegible]
Credentials: jenkins
Advanced...
Add Repository | Delete Repository

Branches to build
Branch Specifier (blank for 'any'): origin/$GERRIT_BRANCH
Add Branch | Delete Branch

Repository browser: (Auto)

Additional Behaviours: Add

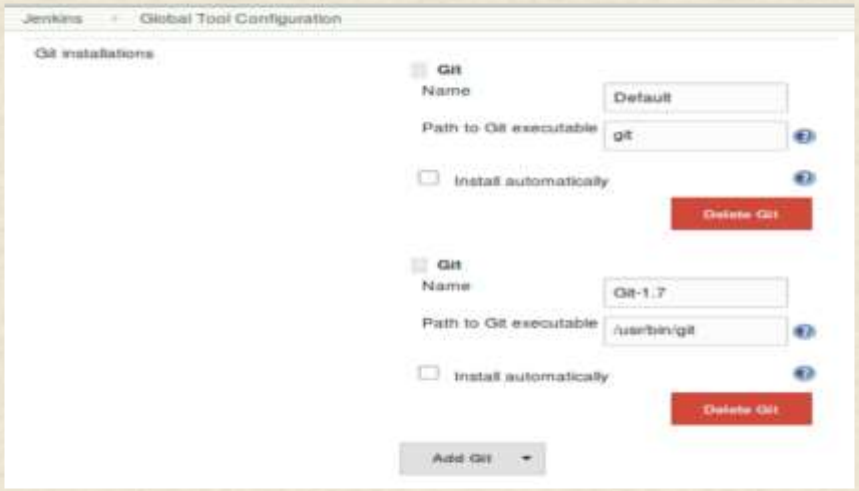
○ Subversion

Save | Apply

Run (job output page and console log)

Example: Integration with Git (Job)

Global Config (Manage Jenkins)


Jenkins » Global Tool Configuration

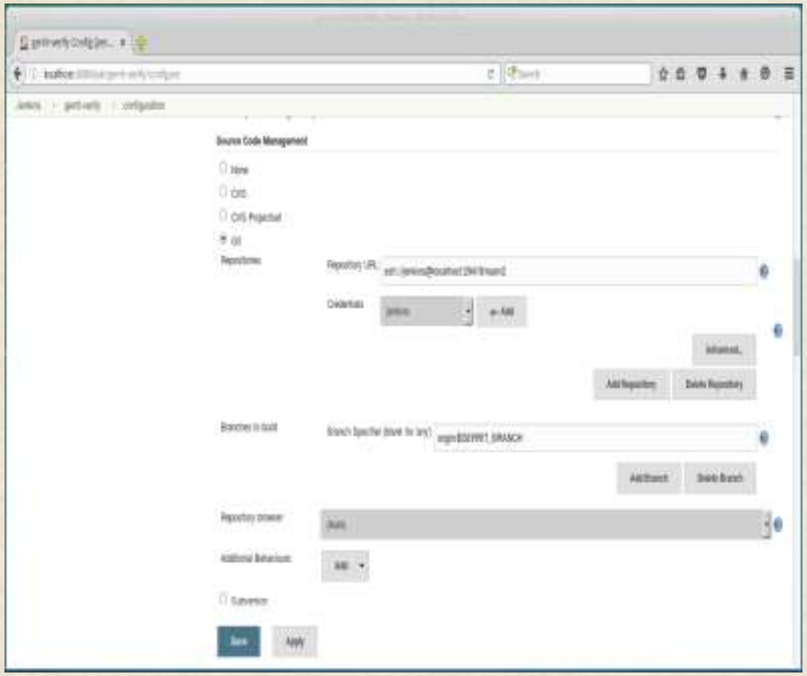
Git installations

Git
Name: Default
Path to Git executable: git
☐ Install automatically
Delete Git

Git
Name: Git-1.7
Path to Git executable: /usr/bin/git
☐ Install automatically
Delete Git

Add Git

Use Locally (Job)


Jenkins » gerrit-verify » configuration

Source Code Management

None
CVS
CVS Projectset
[✓ Git]

Repositories: Repository URL: [illegible]
Credentials: jenkins
Add
Advanced...
Add Repository Delete Repository

Branches to build: Branch Specifier (blank for 'any'): origin/$GERRIT_BRANCH
Add Branch Delete Branch

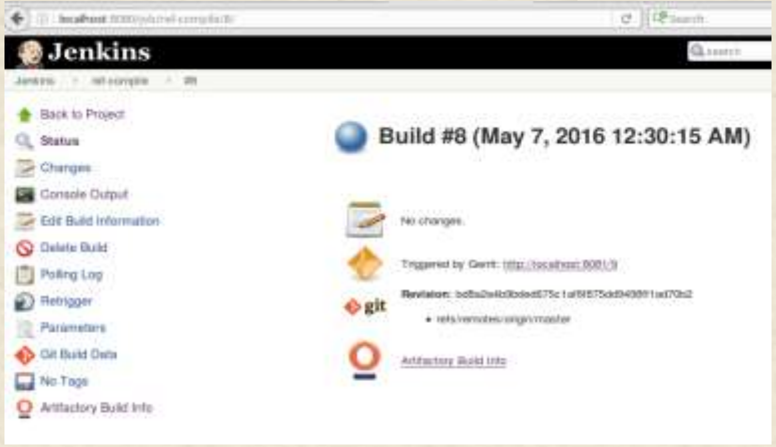
Repository browser: (Auto)

Additional Behaviours: Add

Subversion

Save Apply

Run (job output page and console log)


Jenkins » [illegible] » #8

Back to Project
Status
Changes
Console Output
Edit Build Information
Delete Build
Polling Log
Retrigger
Parameters
Git Build Data
No Tags
Artifactory Build Info

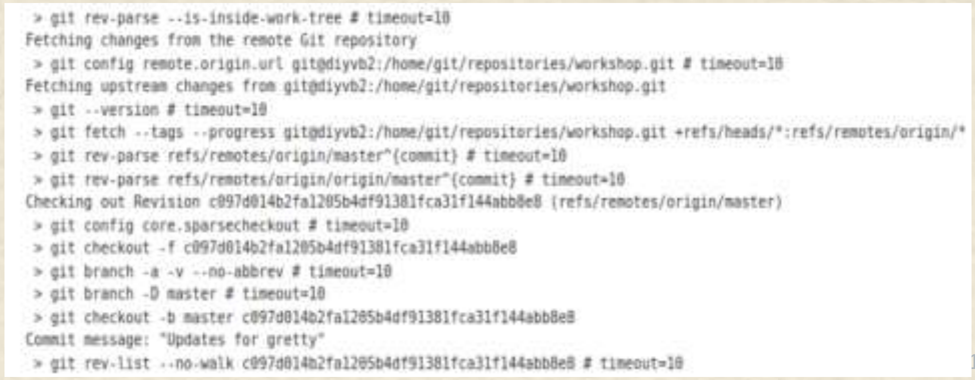
Build #8 (May 7, 2016 12:30:15 AM)

No changes.

Triggered by Gerrit: [illegible]

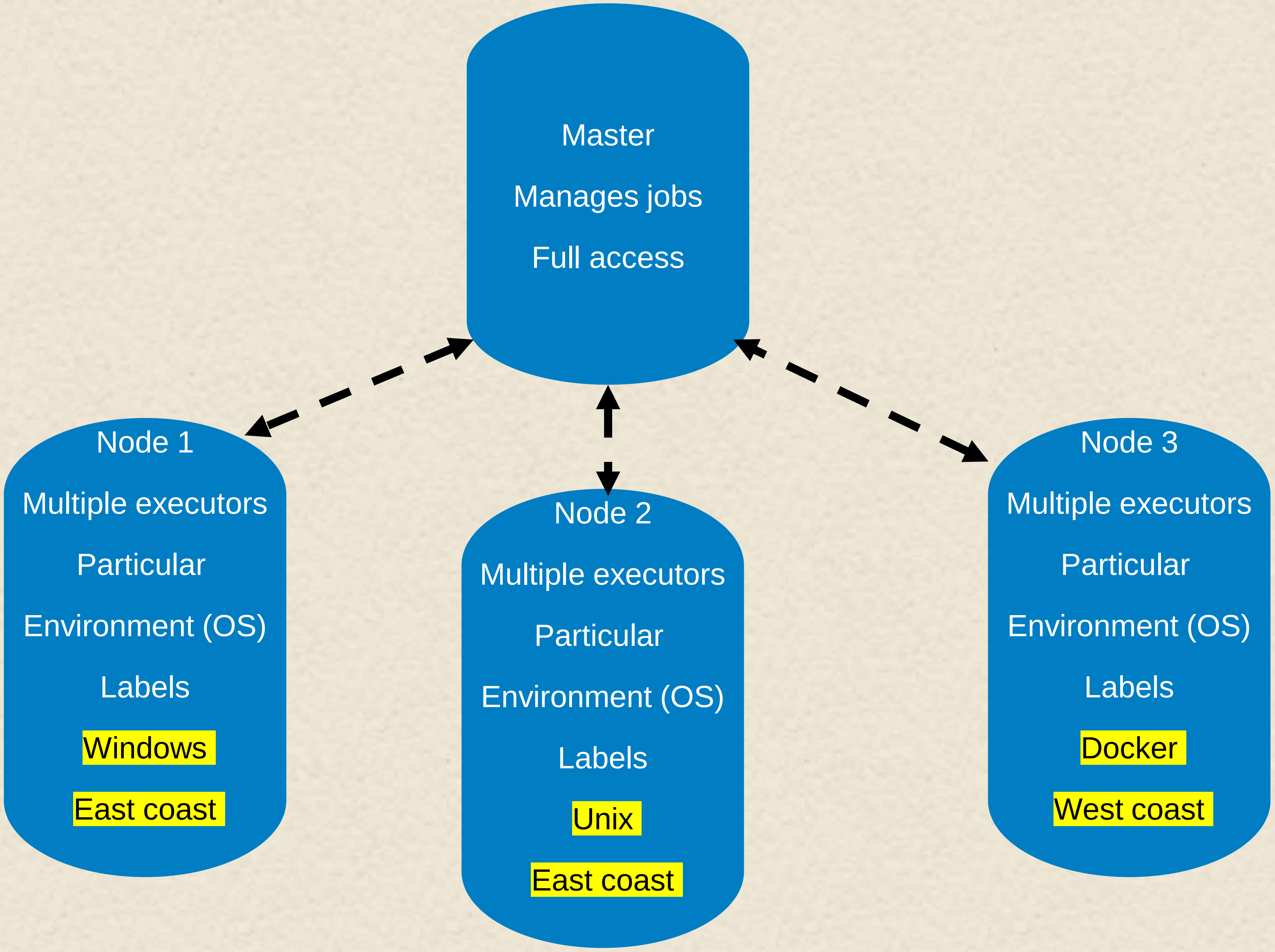
Revision: [illegible]
- refs/remotes/origin/master

Artifactory Build Info


```
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url git@diyvb2:/home/git/repositories/workshop.git # timeout=10
Fetching upstream changes from git@diyvb2:/home/git/repositories/workshop.git
 > git --version # timeout=10
 > git fetch --tags --progress git@diyvb2:/home/git/repositories/workshop.git +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision c097d014b2fa1205b4df91381fca31f144abb8e8 (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f c097d014b2fa1205b4df91381fca31f144abb8e8
 > git branch -a -v --no-abbrev # timeout=10
 > git branch -D master # timeout=10
 > git checkout -b master c097d014b2fa1205b4df91381fca31f144abb8e8
Commit message: "Updates for gretty"
 > git rev-list --no-walk c097d014b2fa1205b4df91381fca31f144abb8e8 # timeout=10
```

- Jobs run on Jenkins Nodes

Jenkins Nodes



About Jenkins jobs

- ❑ Name
- ❑ Description
- ❑ General properties
- ❑ Parameters (optional)
- ❑ SCM : (Git, Subversion, CVS, etc.)
- ❑ Triggers – what initiates the "build" (processing) : polling, another job finishing, notifications, etc.
- ❑ Steps
- ❑ Post-build Actions : (mail, notifications, archiving artifact, etc.)

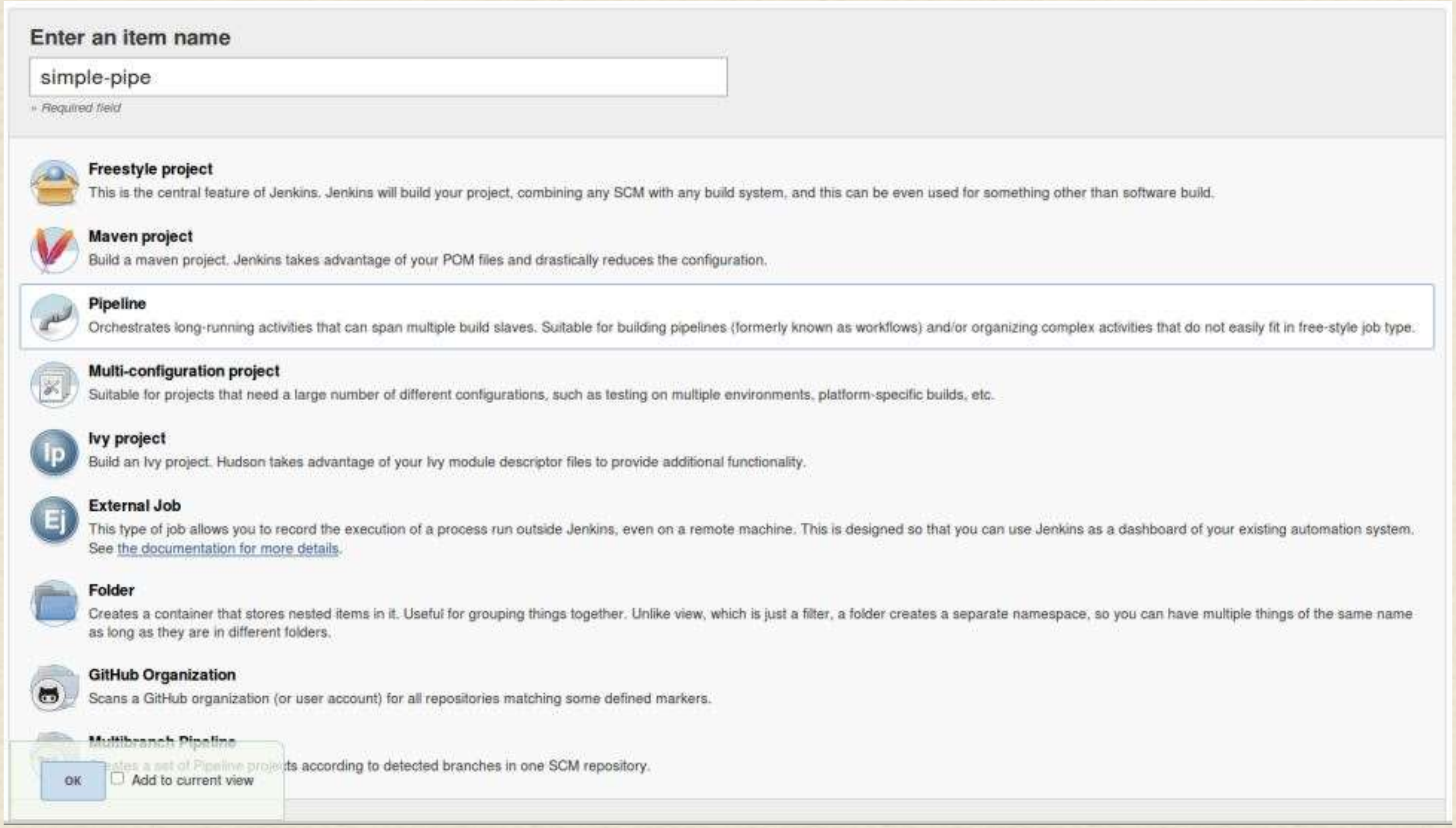
- ❑ Contains configuration, commands, and history of past builds
- ❑ Can be set to run on particular nodes
- ❑ Persist history
- ❑ Have their own "dashboard"
- ❑ Have their own workspace
- ❑ Can be initiated in several ways – manually, SCM polling, from other jobs

What is Jenkins 2 (2.0+)?

- **Features**
  - Next evolution of Jenkins
  - Includes more integrated support for pipelines-as-code
    - Pipelines-as-code is not new with 2.0
  - Pipeline DSL improvements
  - Support for pipeline scripts stored in source control - Jenkinsfiles
  - Automatic project creation based on Jenkinsfile presence in branches
  - Improved DSL structure and processing via Declarative Pipelines
  - Advanced interface - Blue Ocean
  - Still supports FreeStyle

- **Motivations**
  - Treat pipelines as a first class citizen
  - Build support around them as a construct
  - Allow to express in coding
  - Use programming logic
  - Treat like code
    - » Store in SCM
    - » Reviewable
  - Easy to test
  - Text-based
  - Handle exceptional cases
  - Restarts

Types of Projects



Enter an item name

simple-pipe

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Ivy project
Build an Ivy project. Hudson takes advantage of your Ivy module descriptor files to provide additional functionality.

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See the documentation for more details.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

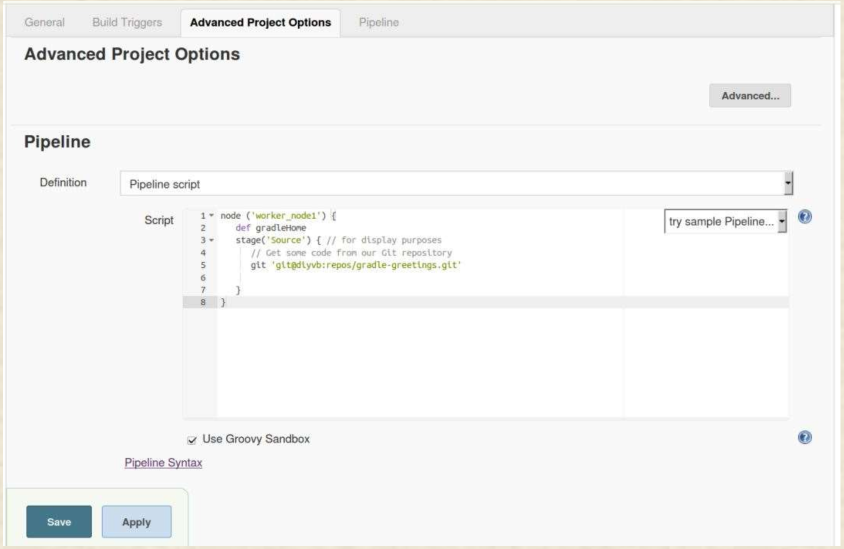
GitHub Organization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

Pipeline-based projects can be written in Jenkins DSL instead of configuring everything through web forms.

Jenkins Domain Specific Language (DSL)

- Groovy-based
- Allows for orchestrating process steps
- Can be written and stored as pipeline script in job itself or as an external Jenkinsfile stored in the repository



Advanced Project Options

Pipeline

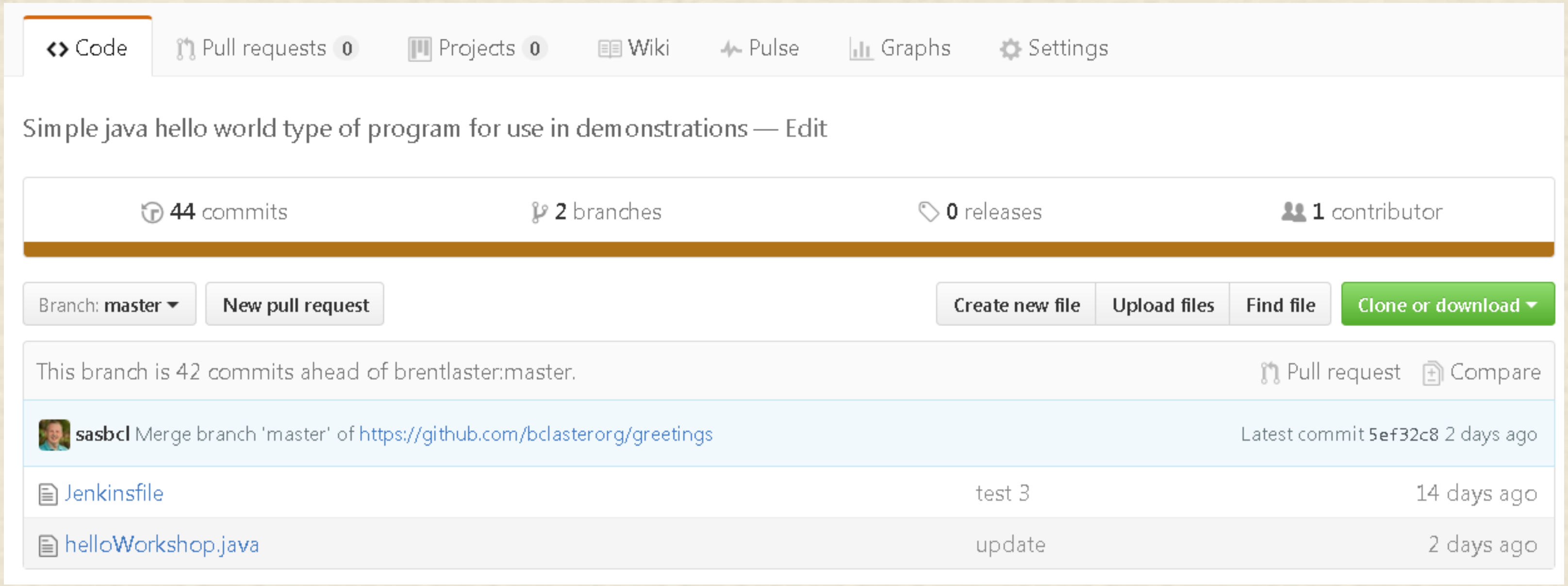
Definition: Pipeline script

Script:

```
node ('worker_node1') {
   def gradleHome
   stage('Source') { // for display purposes
      // Get some code from our Git repository
      git 'git@diyvb:repos/gradle-greetings.git'

   }
}
```

☑ Use Groovy Sandbox

Pipeline Syntax



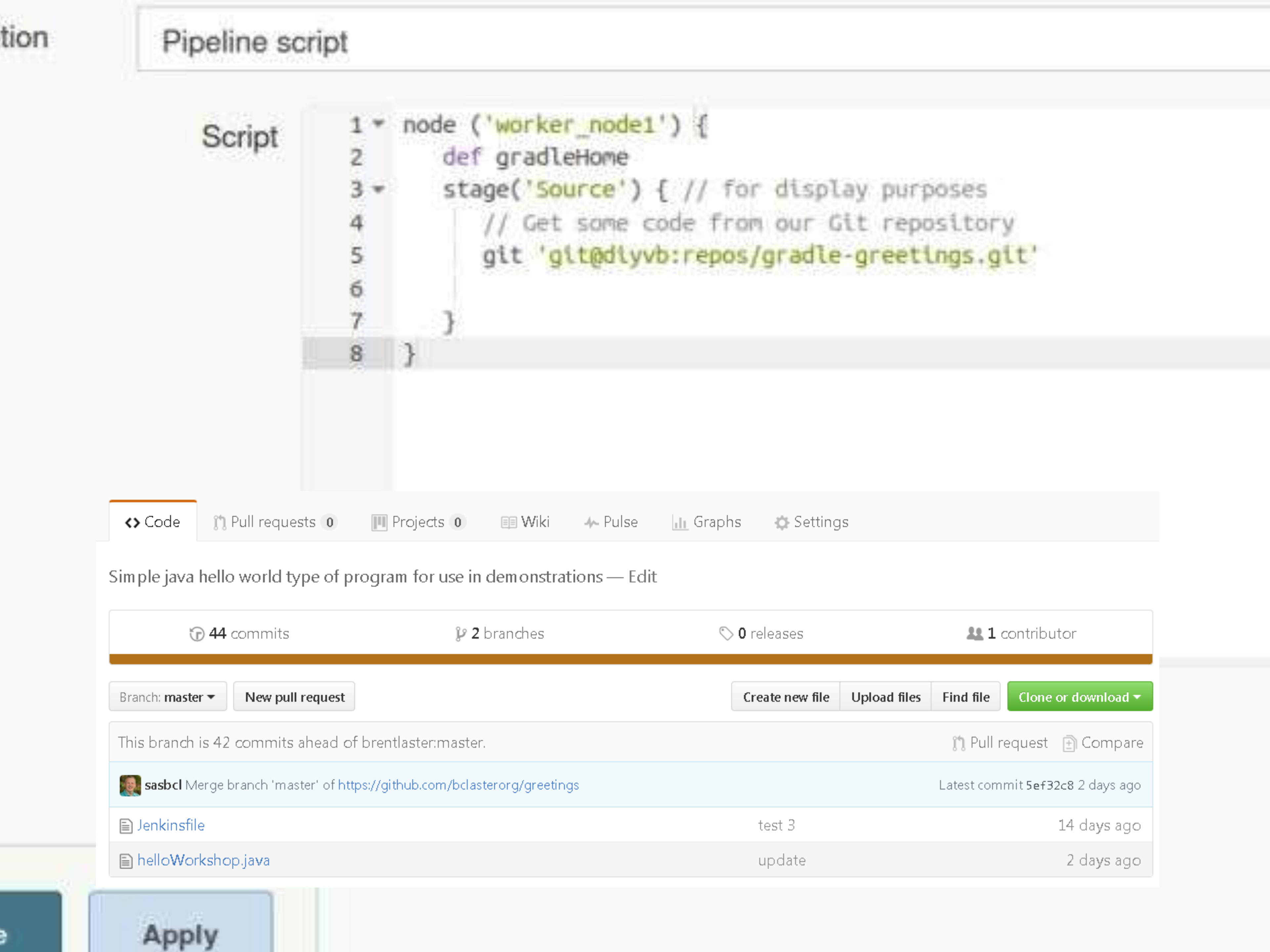
Simple java hello world type of program for use in demonstrations — Edit

44 commits | 2 branches | 0 releases | 1 contributor

This branch is 42 commits ahead of brentlaster:master.

sasbcl Merge branch 'master' of https://github.com/bclasterorg/greetings — Latest commit 5ef32c8 2 days ago

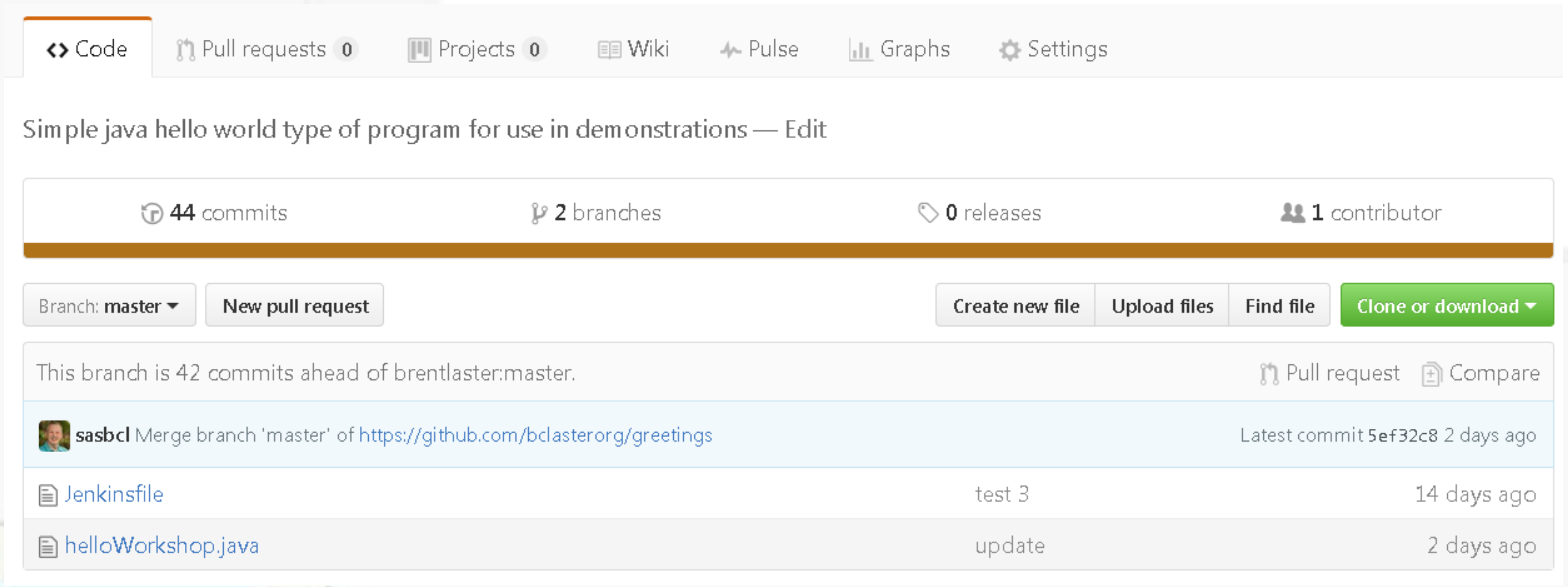
Jenkinsfile	test 3	14 days ago
helloWorkshop.java	update	2 days ago



Pipeline script

Script

```
node ('worker_node1') {
    def gradleHome
    stage('Source') { // for display purposes
      // Get some code from our Git repository
      git 'git@diyvb:repos/gradle-greetings.git'

    }
}
```



Simple java hello world type of program for use in demonstrations — Edit

44 commits | 2 branches | 0 releases | 1 contributor

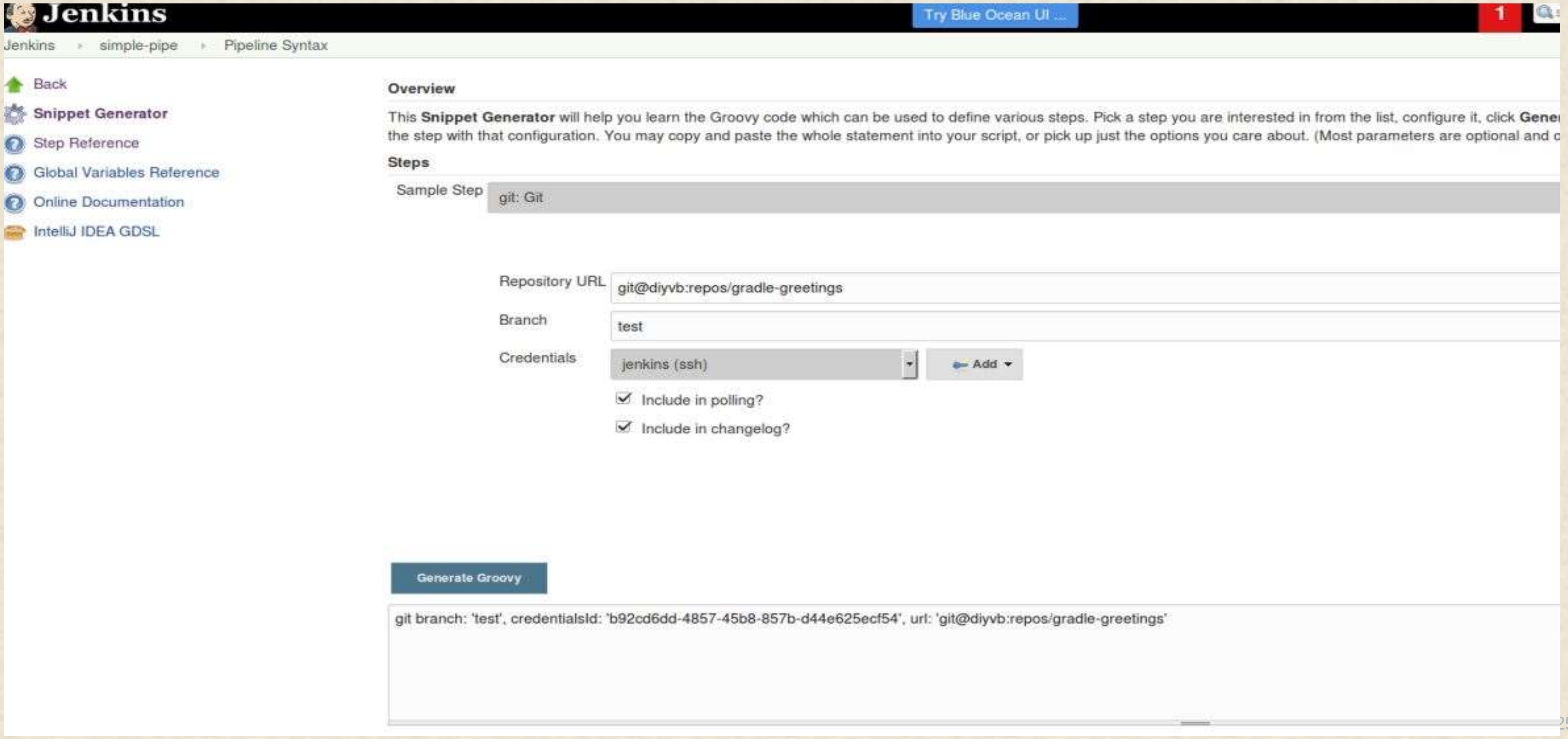
This branch is 42 commits ahead of brentlaster:master.

sasbcl Merge branch 'master' of https://github.com/bclasterorg/greetings — Latest commit 5ef32c8 2 days ago

Jenkinsfile	test 3	14 days ago
helloWorkshop.java	update	2 days ago

Automatic DSL - Snippet Generator

- Facilitates generating Groovy code for typical actions
- Select the operation
- Fill in the arguments/parameters
- Push the button
- Copy and paste



Overview

This **Snippet Generator** will help you learn the Groovy code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Gene**
the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and c

Steps

Sample Step: git: Git

Repository URL: git@diyvb:repos/gradle-greetings

Branch: test

Credentials: jenkins (ssh)

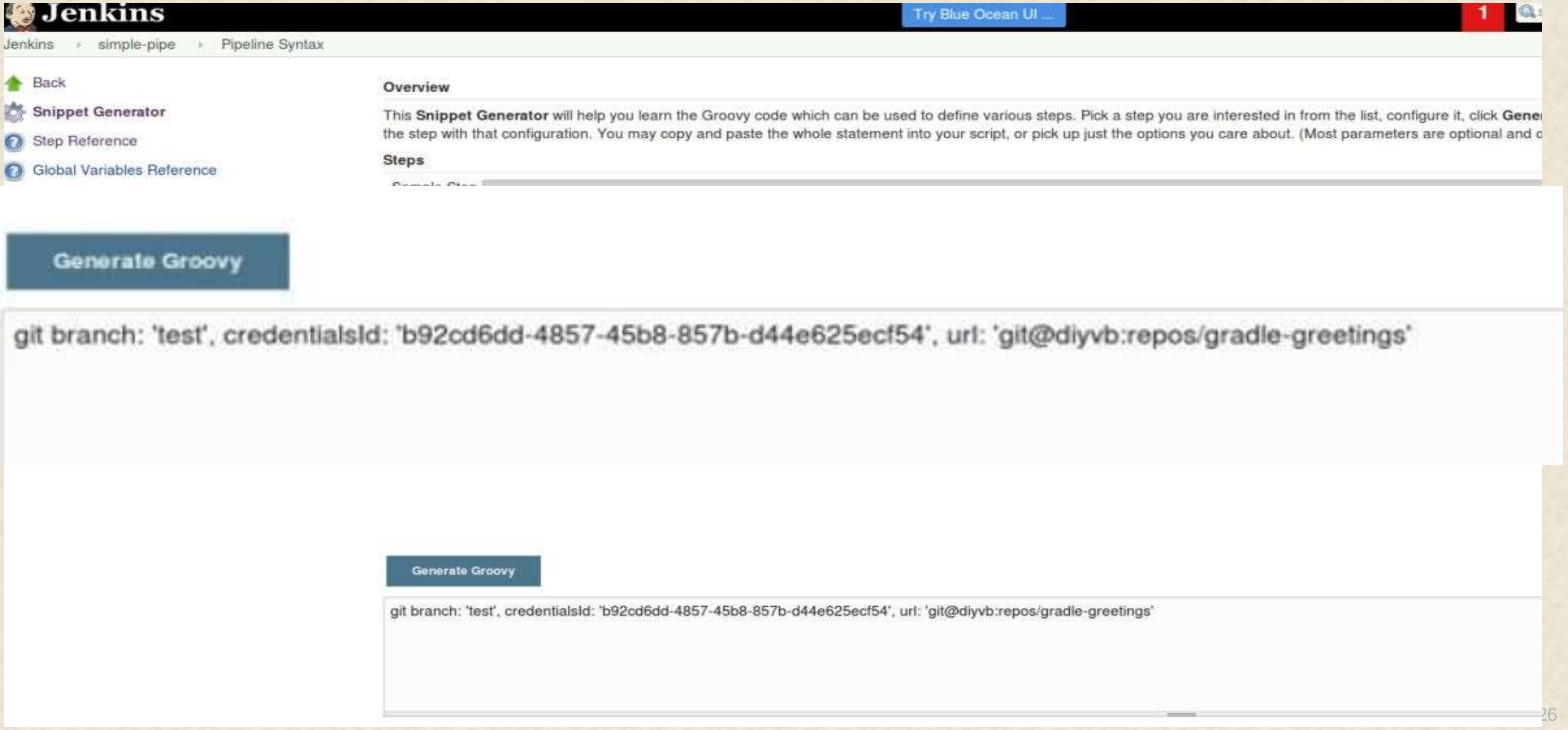
☑ Include in polling?

☑ Include in changelog?

```
git branch: 'test', credentialsId: 'b92cd6dd-4857-45b8-857b-d44e625ecf54', url: 'git@diyvb:repos/gradle-greetings'
```

Automatic DSL - Snippet Generator

- Facilitates generating Groovy code for typical actions
- Select the operation
- Fill in the arguments/parameters
- Push the button
- Copy and paste



Overview

This **Snippet Generator** will help you learn the Groovy code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Gene**
the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and c

Steps

```
git branch: 'test', credentialsId: 'b92cd6dd-4857-45b8-857b-d44e625ecf54', url: 'git@diyvb:repos/gradle-greetings'
```

```
git branch: 'test', credentialsId: 'b92cd6dd-4857-45b8-857b-d44e625ecf54', url: 'git@diyvb:repos/gradle-greetings'
```

Scripted Pipelines – Nodes and Stages

- **Nodes**
  - Tells which system (agent) to run code on
  - Code between {} forms program to run
  - A particular agent can be specified in node(agent)
  - Creates an associated workspace and schedules code steps to run in build queue
  - specific node

- **Stages**
- Aggregates build steps into sections
- Stages are inside of a node block
- Stages take a name (usually corresponding to the function)

```
node {
  // stages
}

node ('agent_1') {
  //  stages
}
```

```
node ('worker_node1') {
   def gradleHome
   stage('Source') { // for display purposes
      // Get some code from our Git repository
      git 'jenkins@localhost:8091/projects/gradle-greetings.git'
   }
```

```
parallel (
            win: { node ('win64'){
            ...
            }},
            linux: { node ('ubuntu'){
            ...
            }},
         )
```

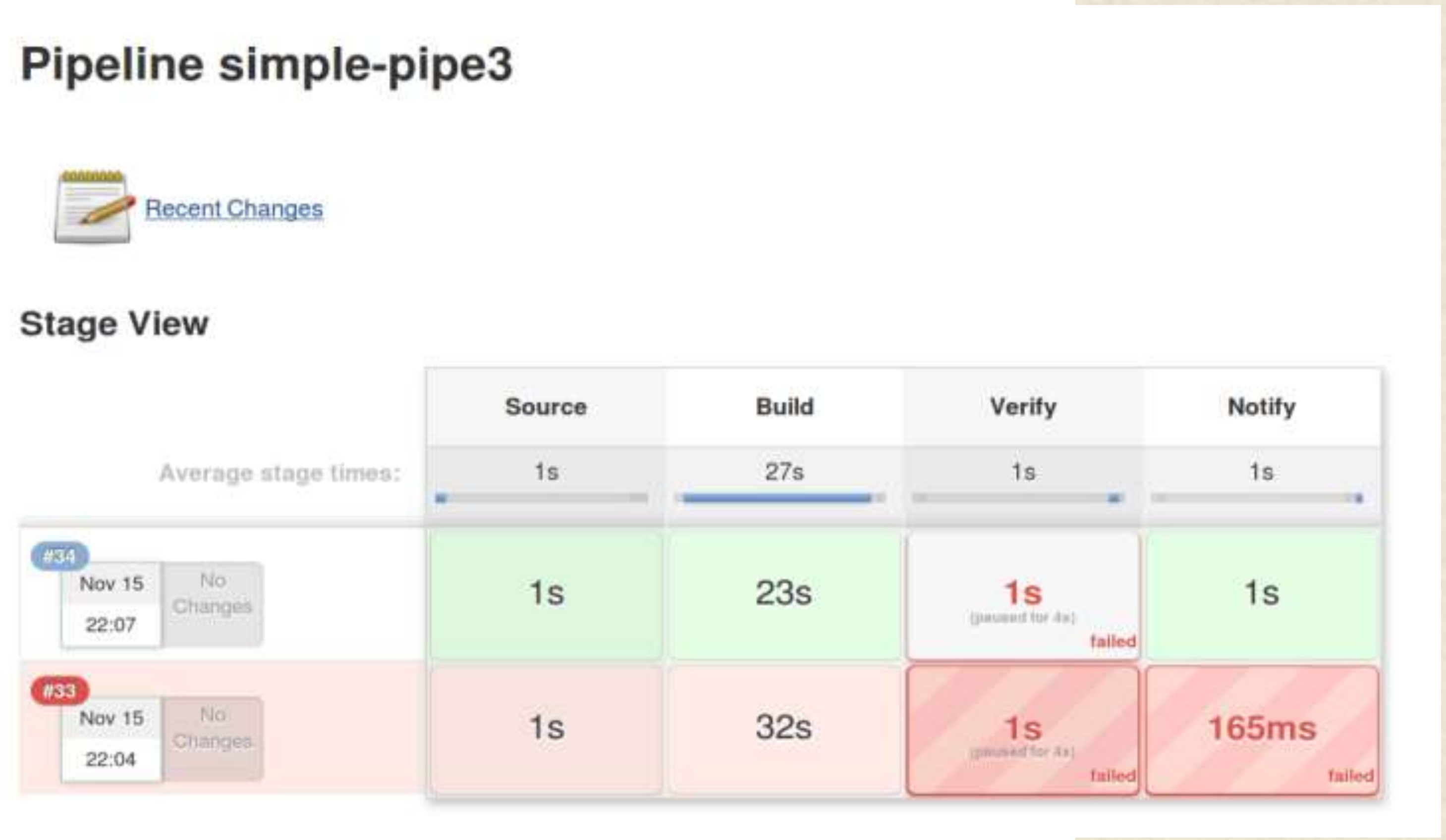
```
   stage('Results') {
      junit '**/target/surefire-reports/TEST-*.xml'
      archive 'target/*.jar'
   }
```

Stages and output – the Stage View

```
import static org.foo.Utilities.*
node ('worker_node1') {
try {
   def gradleHome
   stage('Source') { // for display purposes
      // Get some code from our Git repository
      git 'git@diyvb:repos/gradle-greetings.git'
   }
   stage('Build') {
      // Run the gradle build
      gbuild this, 'clean build'
   }
   stage ('Verify') {
      // Now load 'verify.groovy'.
      def verifyCall = load("/home/diyuser/shared_libraries/src/verify.groovy")
      timeout(time: 5, unit: 'SECONDS') {
        verifyCall("Please Verify the build")
      }
   }  // end verify
   }  // end try
   catch (err) {
        echo "Caught: ${err}"
   }
   stage ('Notify') {
     mailUser('user@domain', getBuildInfo())
   }
}
```

Stages and output – the Stage View

```
import static org.foo.Utilities.*
node ('worker_node1') {
try {
   def gradleHome
   stage('Source') {
      // Get some cod
      git 'git@diyvb:
   }
   stage('Build') {
      // Run the gradl
      gbuild this, 'c
   }
   stage ('Verify') {
      // Now load 've
      def verifyCall =
      timeout(time: 5,
        verifyCall("P
      }
   }  // end verify
   }  // end try
   catch (err) {
        echo "Caught:
   }
   stage ('Notify') {
     mailUser('user@domain', getBuildInfo())
   }
}
```



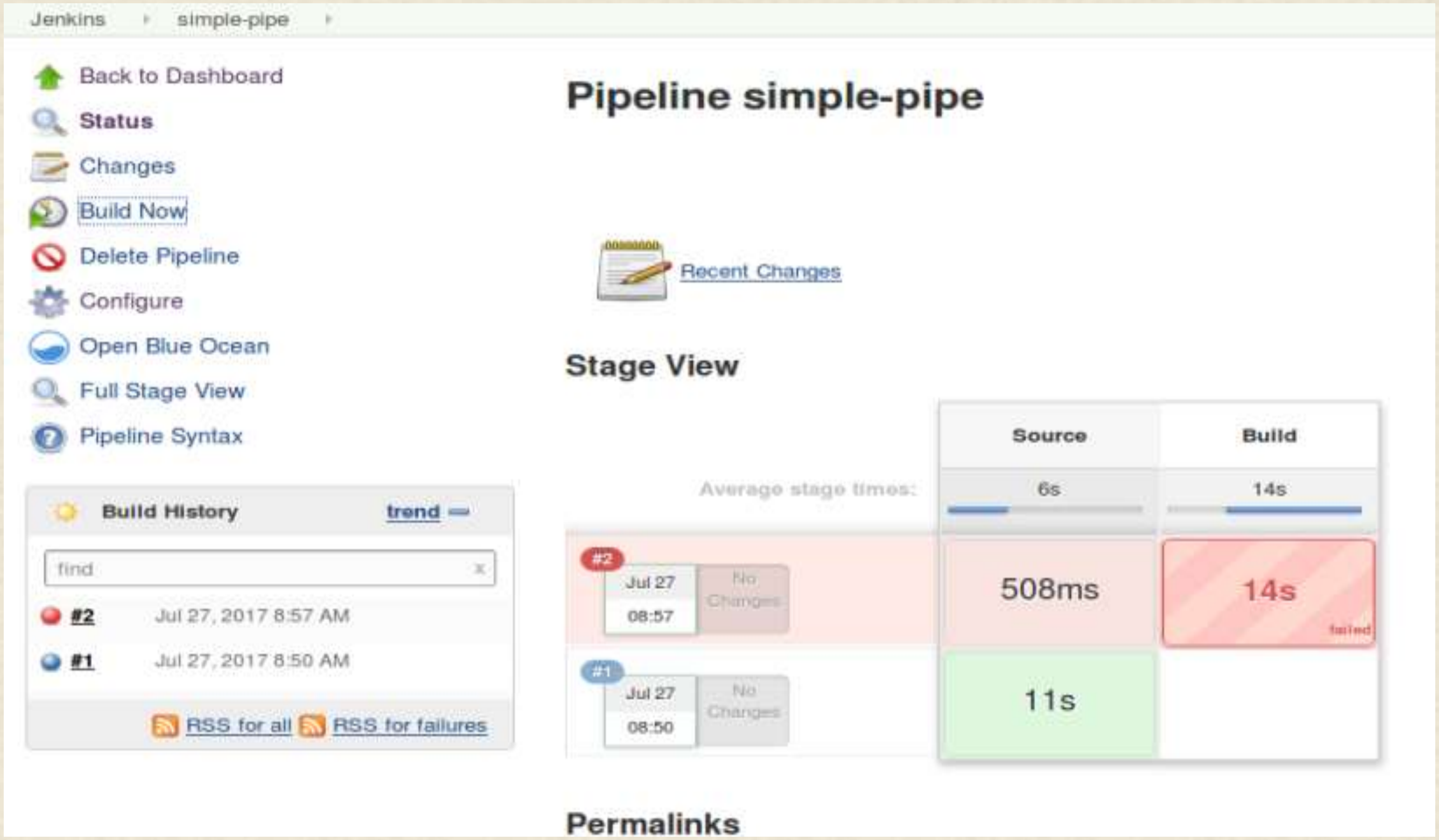
Pipeline simple-pipe3

Recent Changes

Stage View

	Source	Build	Verify	Notify
Average stage times:	1s	27s	1s	1s
#34 Nov 15 22:07 No Changes	1s	23s	1s (paused for 4s) failed	1s
#33 Nov 15 22:04 No Changes	1s	32s	1s (paused for 4s) failed	165ms failed

Stage View and Logs


Jenkins › simple-pipe ›

- Back to Dashboard
- Status
- Changes
- Build Now
- Delete Pipeline
- Configure
- Open Blue Ocean
- Full Stage View
- Pipeline Syntax

Build History trend

find

#2	Jul 27, 2017 8:57 AM
#1	Jul 27, 2017 8:50 AM

RSS for all RSS for failures

Pipeline simple-pipe

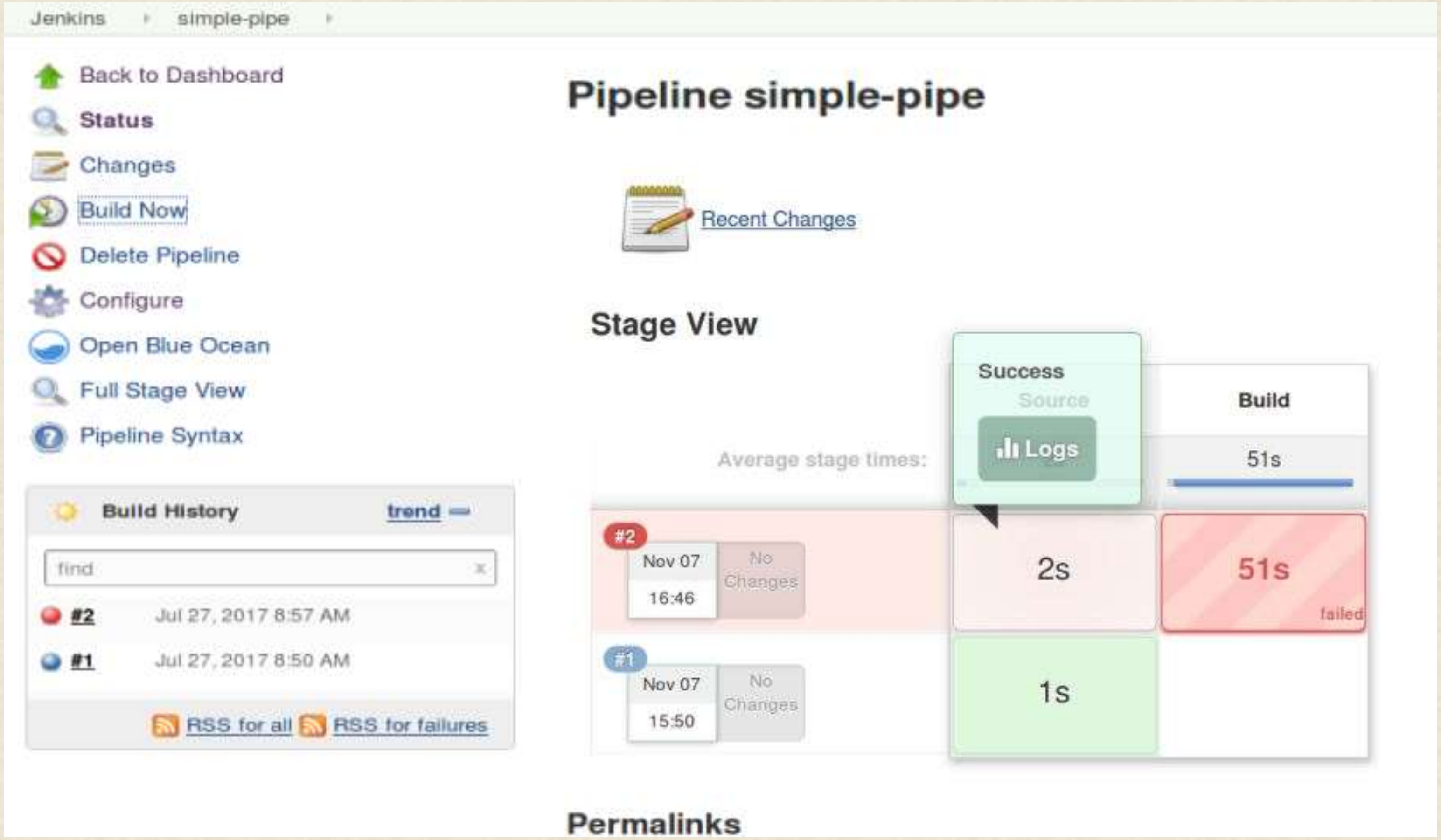
Recent Changes

Stage View

	Source	Build
Average stage times:	6s	14s
#2 Jul 27 08:57 No Changes	508ms	14s failed
#1 Jul 27 08:50 No Changes	11s	

Permalinks

Stage View and Logs



Pipeline simple-pipe

Stage View

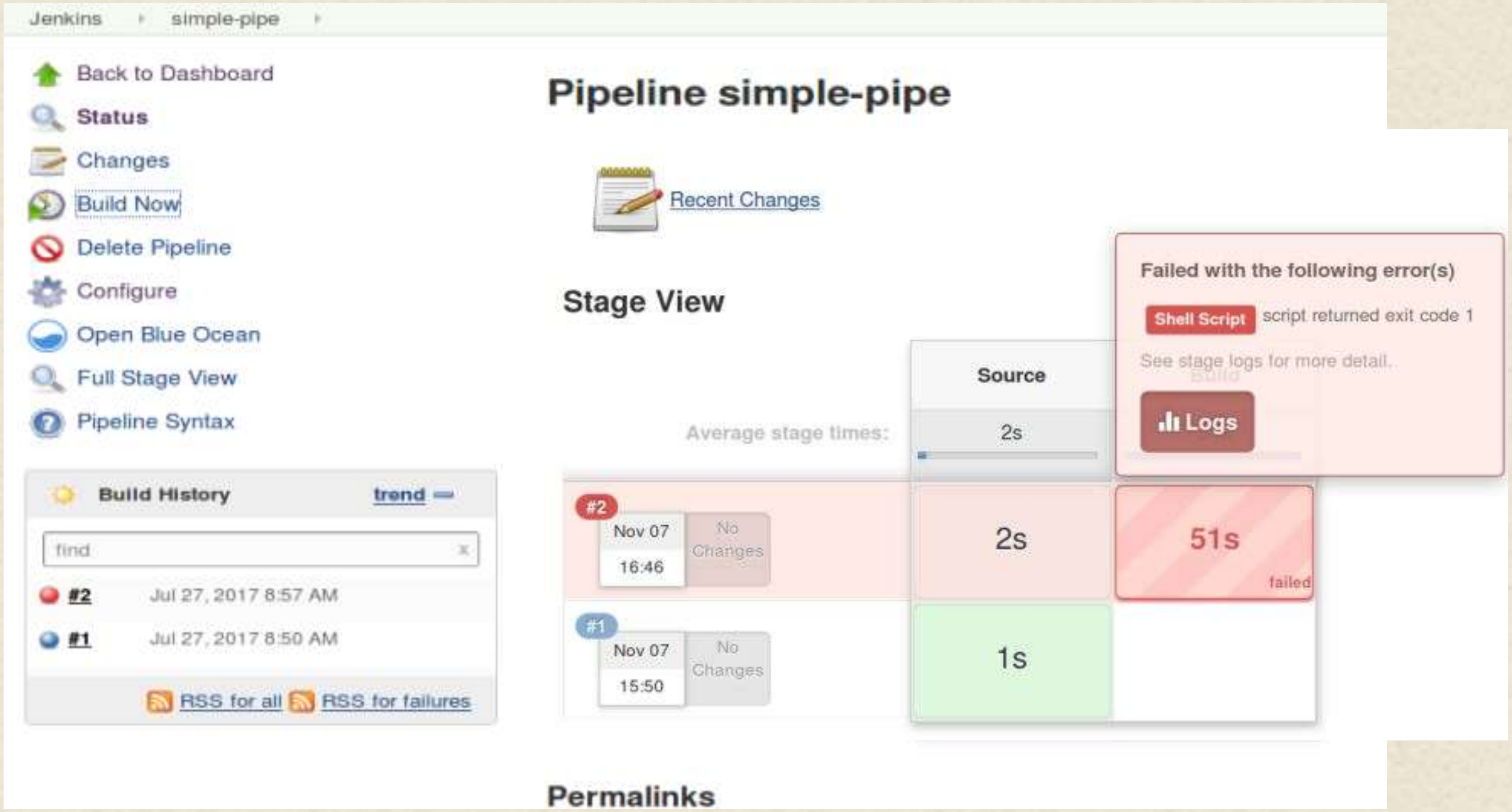
	Source	Build
Average stage times:		51s
#2 Nov 07 16:46 No Changes	2s	51s failed
#1 Nov 07 15:50 No Changes	1s	

Success

#2	Jul 27, 2017 8:57 AM
#1	Jul 27, 2017 8:50 AM

Permalinks

Stage View and Logs



Jenkins › simple-pipe ›

- Back to Dashboard
- Status
- Changes
- Build Now
- Delete Pipeline
- Configure
- Open Blue Ocean
- Full Stage View
- Pipeline Syntax

Build History trend

find

#2	Jul 27, 2017 8:57 AM
#1	Jul 27, 2017 8:50 AM

RSS for all RSS for failures

Pipeline simple-pipe

Recent Changes

Stage View

Failed with the following error(s)

Shell Script script returned exit code 1

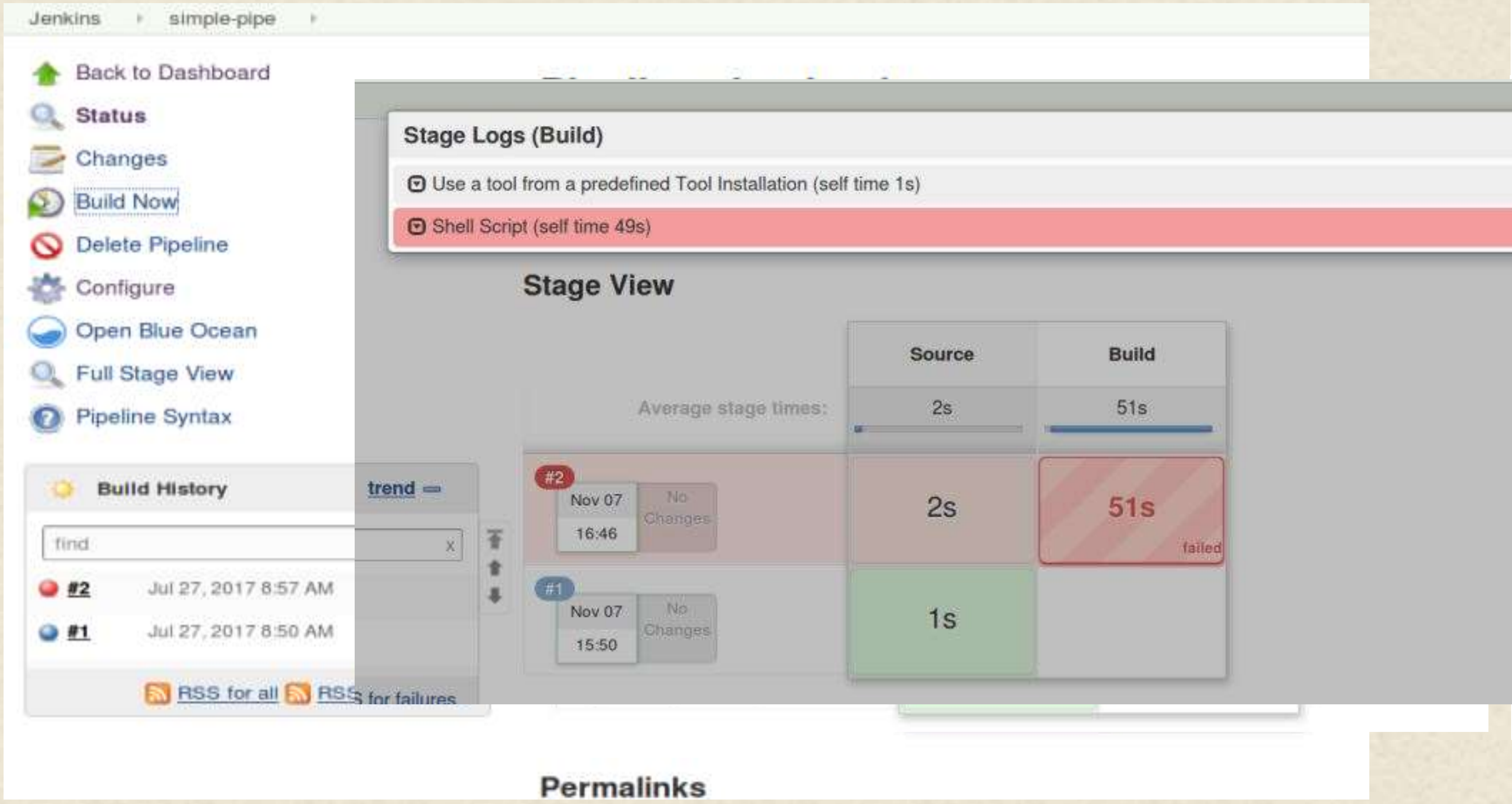
See stage logs for more detail.

Logs

		Source	Build
Average stage times:		2s	
#2 Nov 07 16:46	No Changes	2s	51s failed
#1 Nov 07 15:50	No Changes	1s	

Permalinks

Stage View and Logs



#2	Jul 27, 2017 8:57 AM	
#1	Jul 27, 2017 8:50 AM	

Stage Logs (Build)

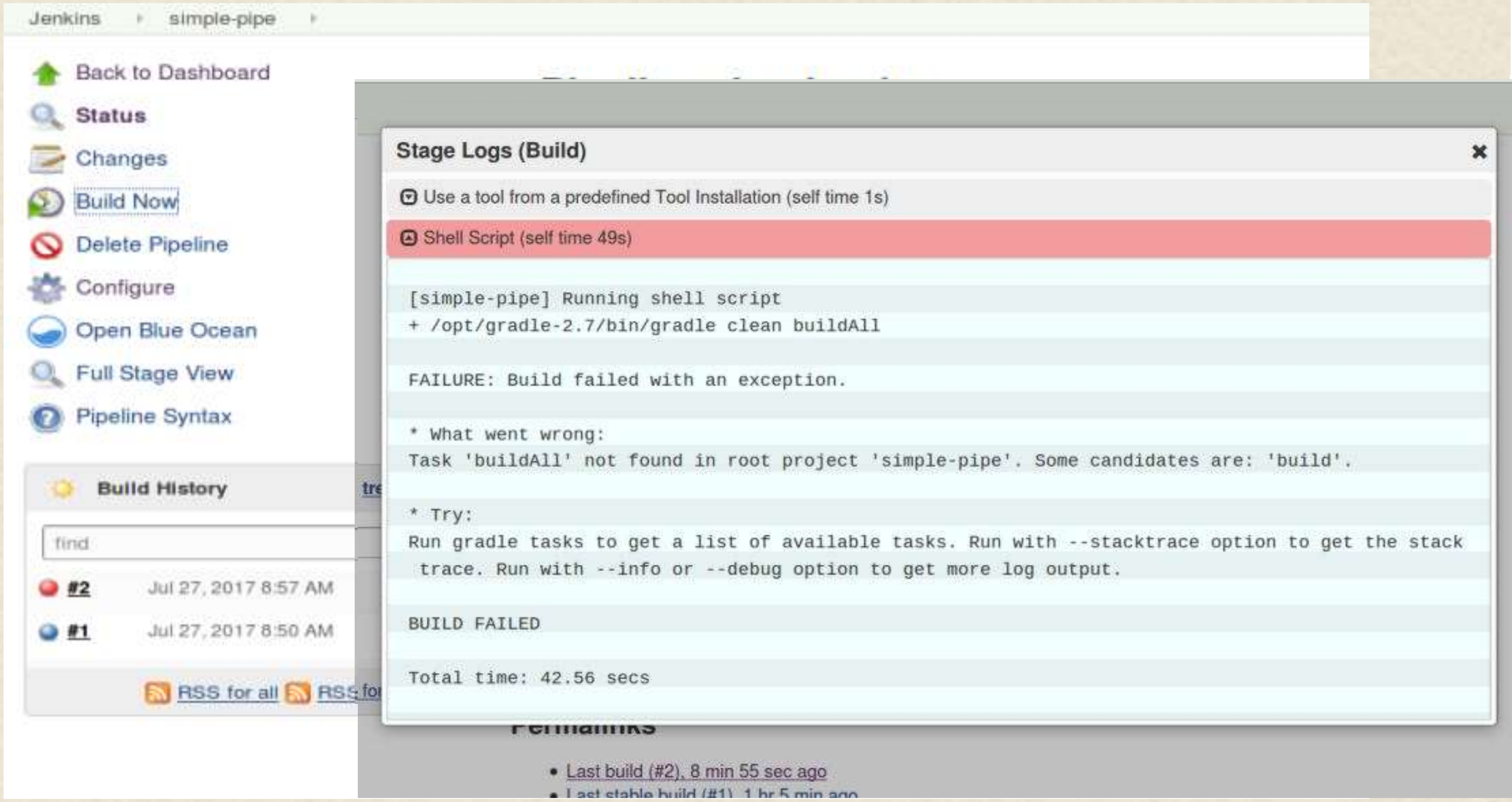
Use a tool from a predefined Tool Installation (self time 1s)

Shell Script (self time 49s)

Stage View

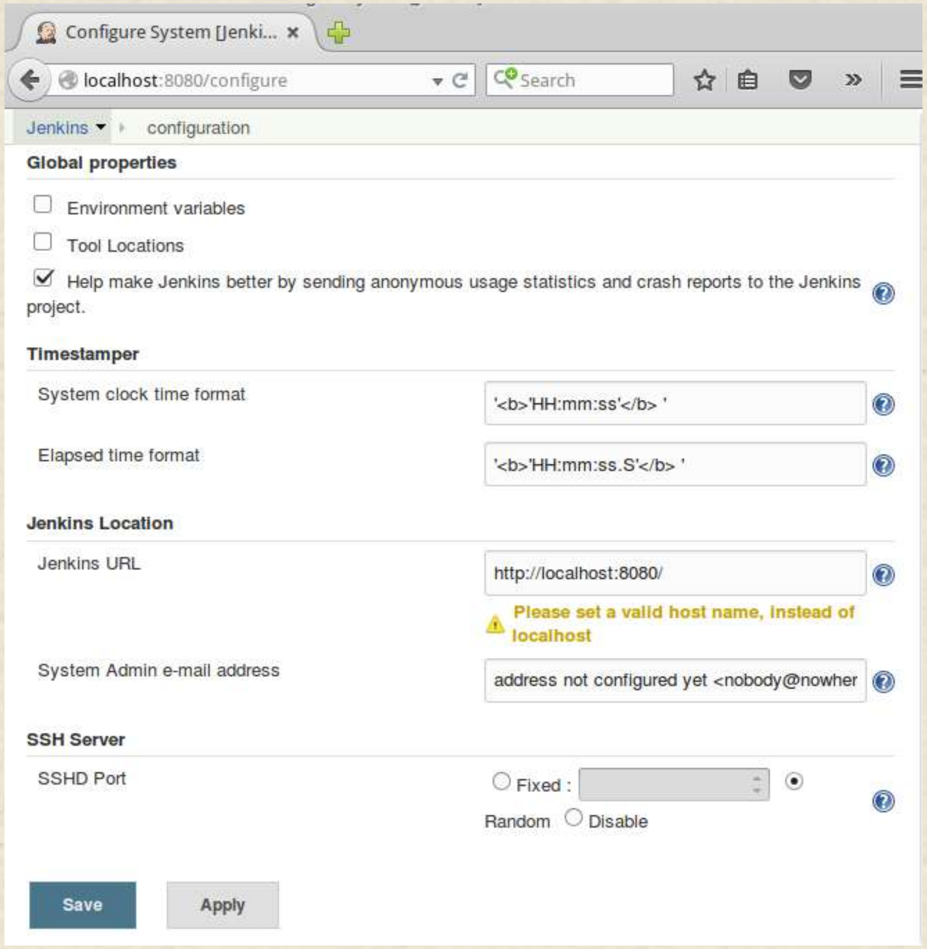
	Source	Build
Average stage times:	2s	51s
#2 Nov 07 16:46 No Changes	2s	51s failed
#1 Nov 07 15:50 No Changes	1s	

Stage View and Logs



```
[simple-pipe] Running shell script
+ /opt/gradle-2.7/bin/gradle clean buildAll

FAILURE: Build failed with an exception.

* What went wrong:
Task 'buildAll' not found in root project 'simple-pipe'. Some candidates are: 'build'.

* Try:
Run gradle tasks to get a list of available tasks. Run with --stacktrace option to get the stack
 trace. Run with --info or --debug option to get more log output.

BUILD FAILED

Total time: 42.56 secs
```

Global Configuration for Pipelines



Configure System [Jenki... ×

localhost:8080/configure

Jenkins ▸ configuration

Global properties

☐ Environment variables

☐ Tool Locations

☑ Help make Jenkins better by sending anonymous usage statistics and crash reports to the Jenkins project.

Timestamper

System clock time format	'<b>'HH:mm:ss'</b> '
Elapsed time format	'<b>'HH:mm:ss.S'</b> '

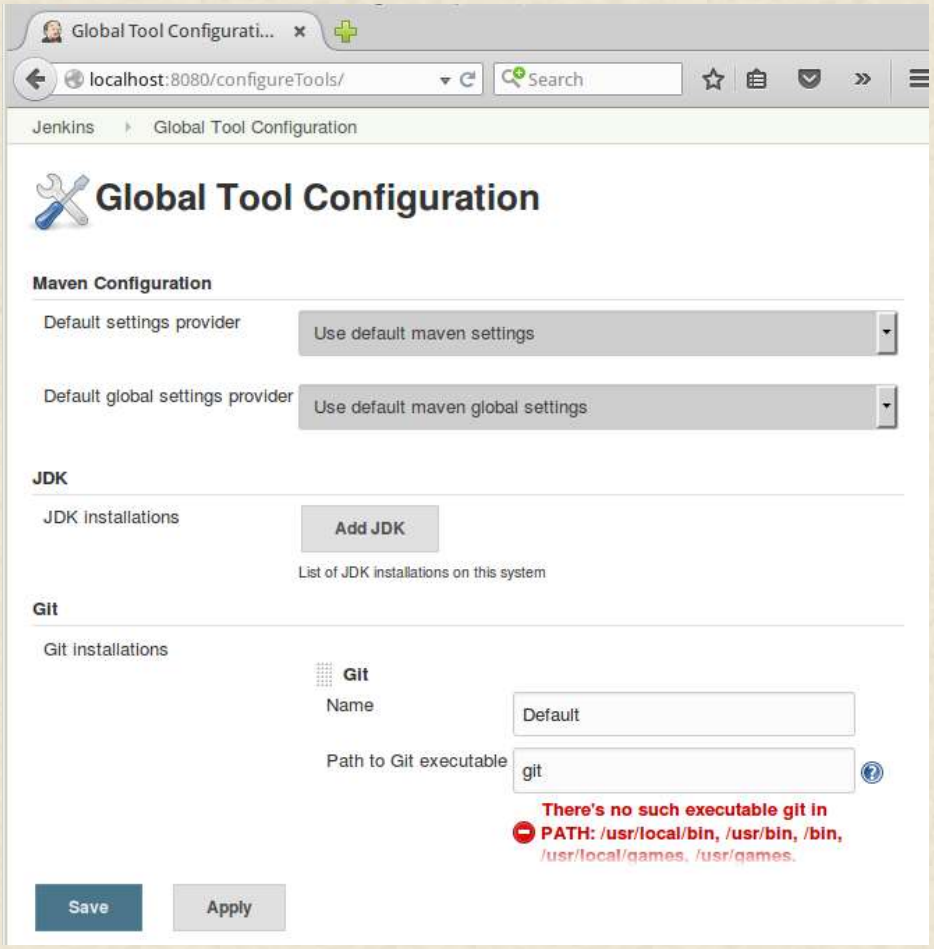
Jenkins Location

Jenkins URL	http://localhost:8080/
	⚠ Please set a valid host name, instead of localhost
System Admin e-mail address	address not configured yet <nobody@nowher

SSH Server

SSHD Port ○ Fixed : [✓ Random] ○ Disable

Save Apply



Global Tool Configurati... ×

localhost:8080/configureTools/

Jenkins ▸ Global Tool Configuration

Global Tool Configuration

Maven Configuration

Default settings provider	Use default maven settings
Default global settings provider	Use default maven global settings

JDK

JDK installations: Add JDK

List of JDK installations on this system

Git

Git installations

Git

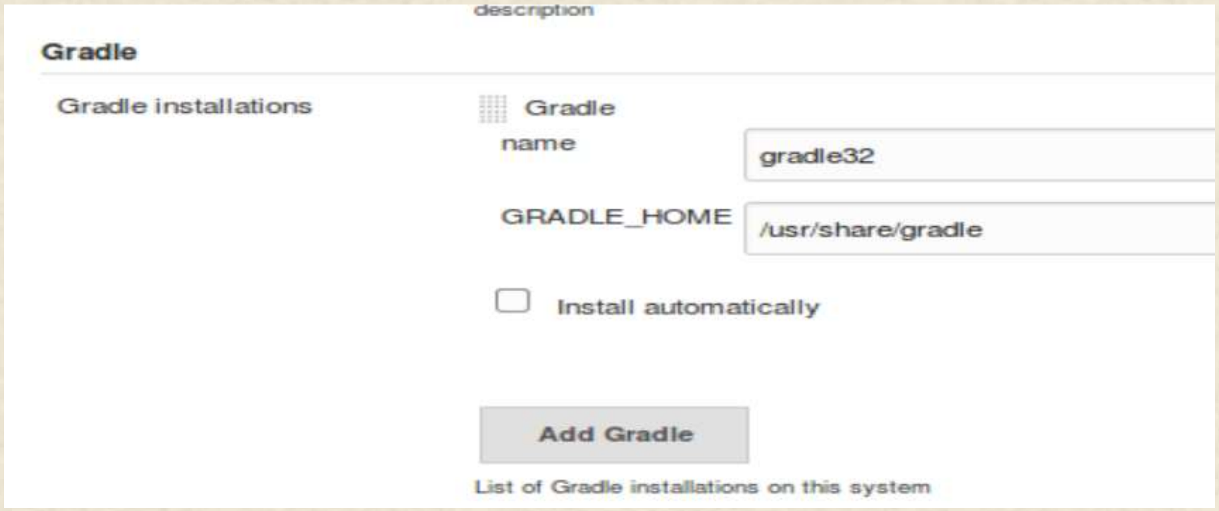
Name	Default
Path to Git executable	git

There's no such executable git in PATH: /usr/local/bin, /usr/bin, /bin, /usr/local/games, /usr/games.

Save Apply

Global Tools and Pipeline

- Tools defined in global configuration
- DSL keyword 'tool'
- In pipeline script
  - def variable for tool location
  - assign variable to "tool <toolname>" from global configuration
  - Use variable in place of location in script



description

Gradle

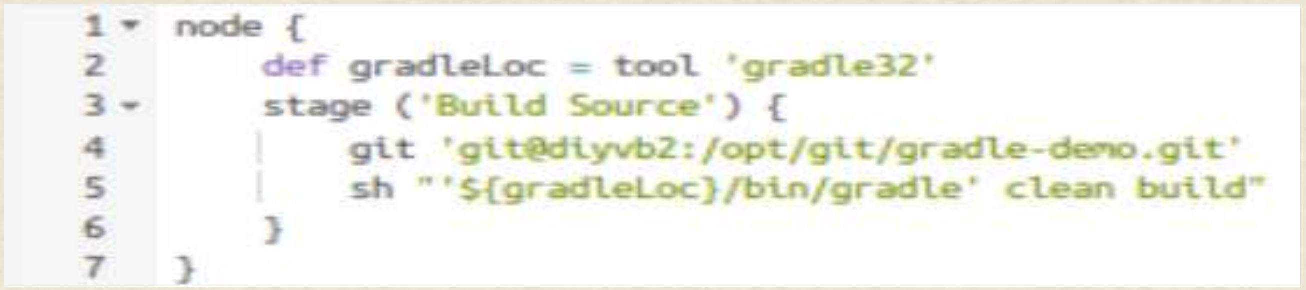
Gradle installations — Gradle

name: gradle32

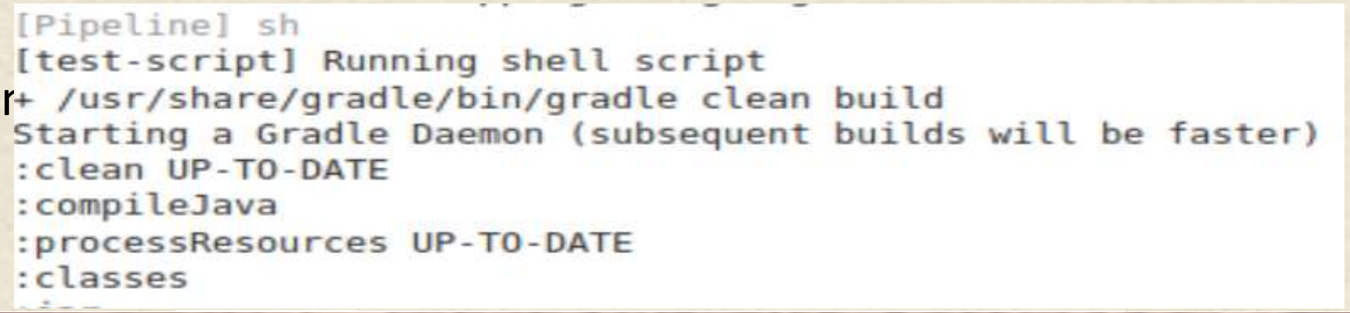
GRADLE_HOME: /usr/share/gradle

☐ Install automatically

List of Gradle installations on this system



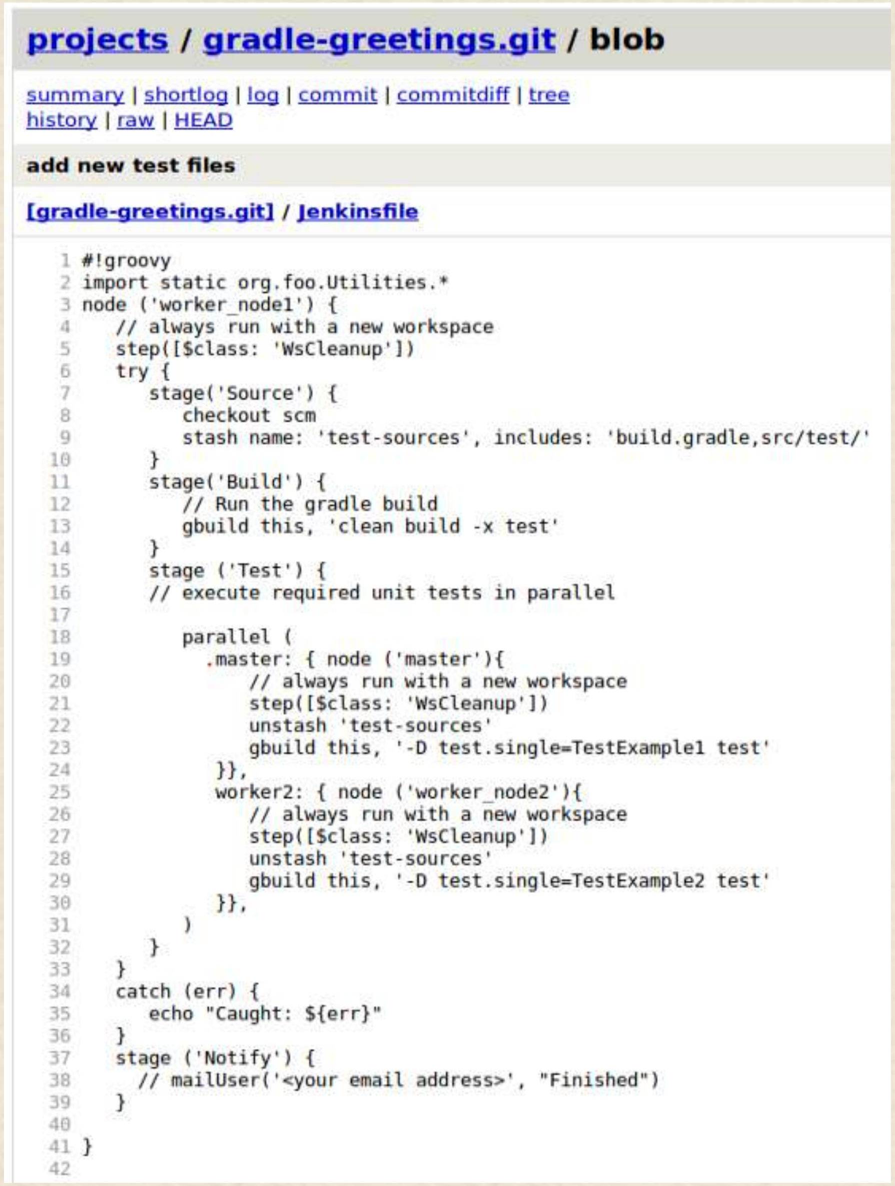
```
node {
    def gradleLoc = tool 'gradle32'
    stage ('Build Source') {
        git 'git@diyvb2:/opt/git/gradle-demo.git'
        sh "'${gradleLoc}/bin/gradle' clean build"
    }
}
```



```
[Pipeline] sh
[test-script] Running shell script
+ /usr/share/gradle/bin/gradle clean build
Starting a Gradle Daemon (subsequent builds will be faster)
:clean UP-TO-DATE
:compileJava
:processResources UP-TO-DATE
:classes
```

The Jenkinsfile

- Pipeline script stored in SCM
- Can develop in the job and then transfer to Jenkinsfile
- Granularity is per branch, per project
- Not required for Jenkins to build
- Best practice to add “#!groovy” at the top
- Used as marker for Jenkins to identify branches (including creation and deletion) in multibranch and organization projects



projects / gradle-greetings.git / blob

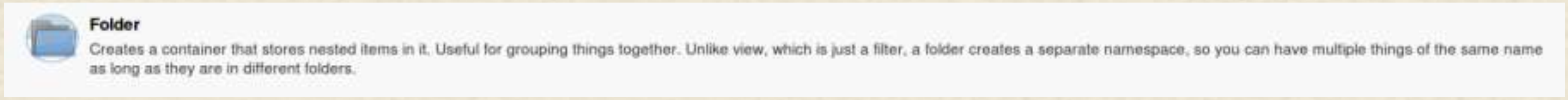
summary | shortlog | log | commit | commitdiff | tree
history | raw | HEAD

add new test files

[gradle-greetings.git] / Jenkinsfile

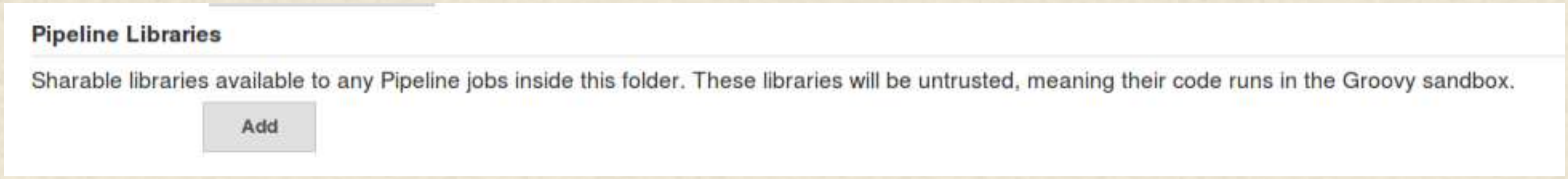
```
#!groovy
import static org.foo.Utilities.*
node ('worker_node1') {
    // always run with a new workspace
    step([$class: 'WsCleanup'])
    try {
        stage('Source') {
            checkout scm
            stash name: 'test-sources', includes: 'build.gradle,src/test/'
        }
        stage('Build') {
            // Run the gradle build
            gbuild this, 'clean build -x test'
        }
        stage ('Test') {
        // execute required unit tests in parallel

            parallel (
              .master: { node ('master'){
                    // always run with a new workspace
                    step([$class: 'WsCleanup'])
                    unstash 'test-sources'
                    gbuild this, '-D test.single=TestExample1 test'
                }},
                worker2: { node ('worker_node2'){
                    // always run with a new workspace
                    step([$class: 'WsCleanup'])
                    unstash 'test-sources'
                    gbuild this, '-D test.single=TestExample2 test'
                }},
            )
        }
    }
    catch (err) {
        echo "Caught: ${err}"
    }
    stage ('Notify') {
      // mailUser('<your email address>', "Finished")
    }

}
```

Folder project


Folder

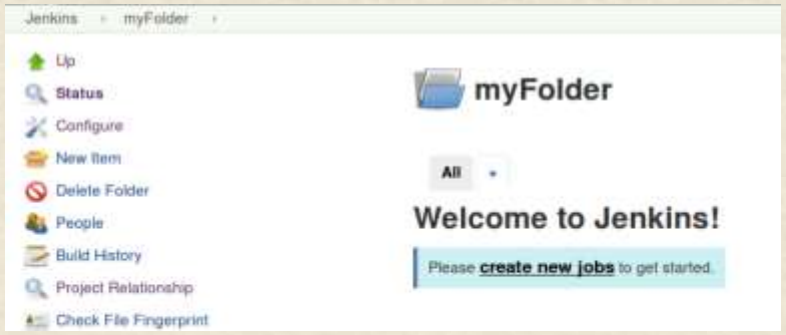
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

- Creates a high-level container for other projects
- Provides a separate namespace (not just viewing organization like views)
- Allows for pipeline libraries that can be shared among jobs in the folder


Pipeline Libraries

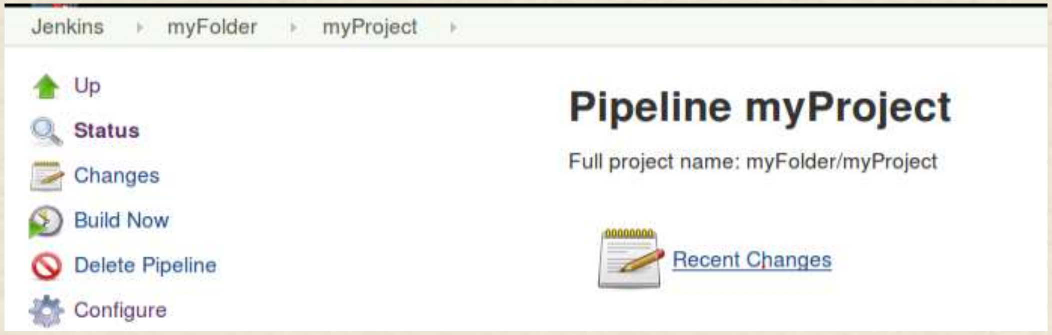
Sharable libraries available to any Pipeline jobs inside this folder. These libraries will be untrusted, meaning their code runs in the Groovy sandbox.

- Once folder project is created, interface is available to create other jobs inside of it
- Full name of items in folder are <Folder name>/<Item name>



Welcome to Jenkins!

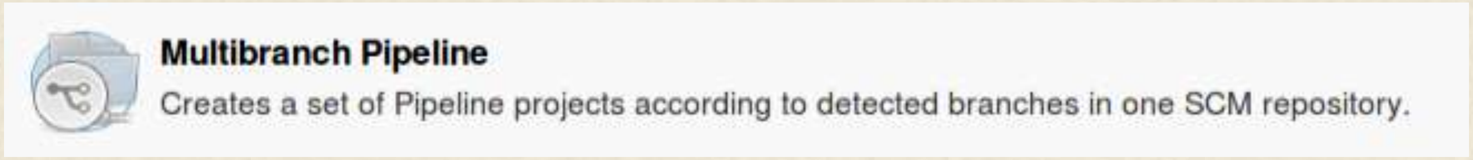
Please **create new jobs** to get started.



Pipeline myProject

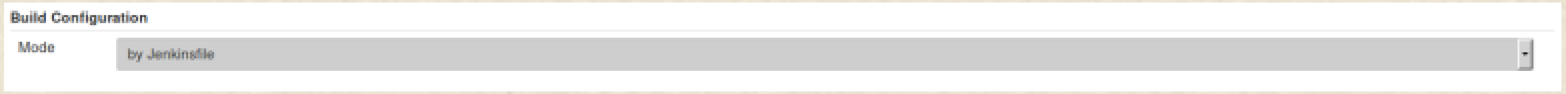
Full project name: myFolder/myProject

Multibranch Pipeline


Multibranch Pipeline

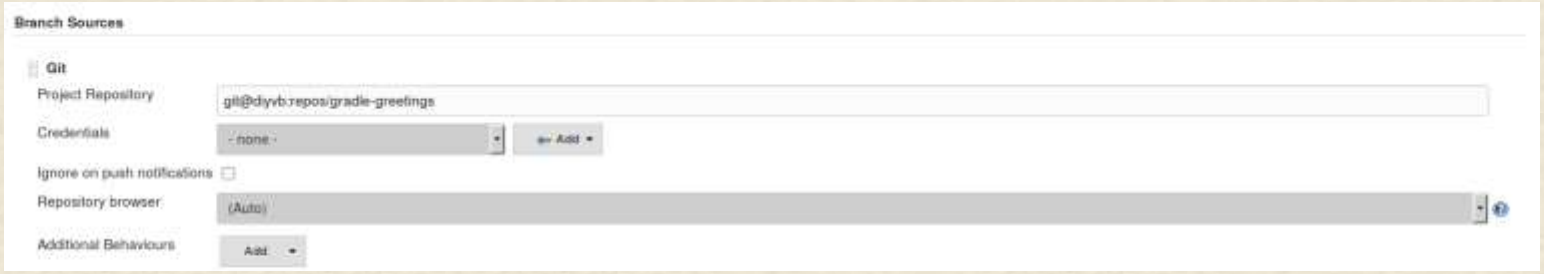
Creates a set of Pipeline projects according to detected branches in one SCM repository.

- Creates pipeline projects in Jenkins to correspond to branches in an SCM repository
- Marker for whether a branch should have a corresponding job is presence of a Jenkinsfile in branch


Build Configuration

Mode	by Jenkinsfile

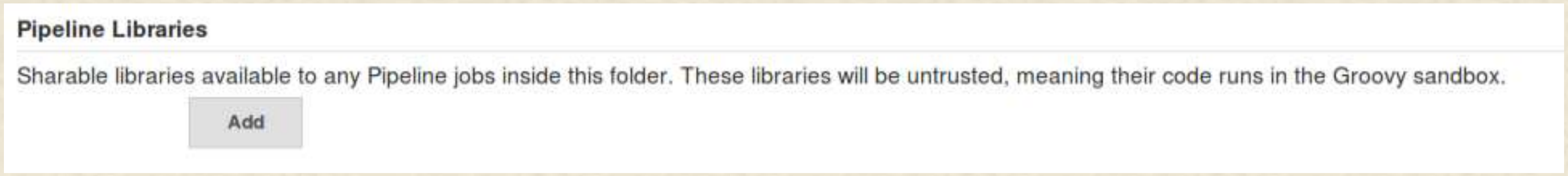
- Configure sources just like any other job - but don't specify branches - (except to include or exclude if needed)


Branch Sources

Git

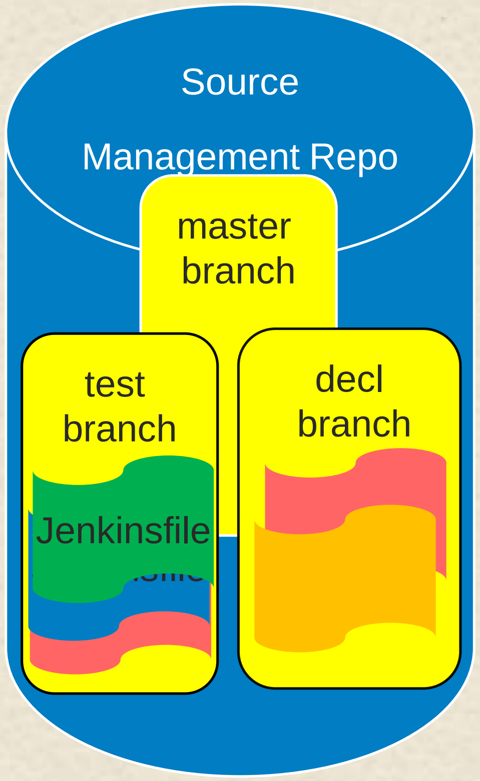
Project Repository	git@diyvb:repos/gradle-greetings
Credentials	- none -
Ignore on push notifications	☐
Repository browser	(Auto)
Additional Behaviours	Add

- Can also include Shared Pipeline Libraries just for this set

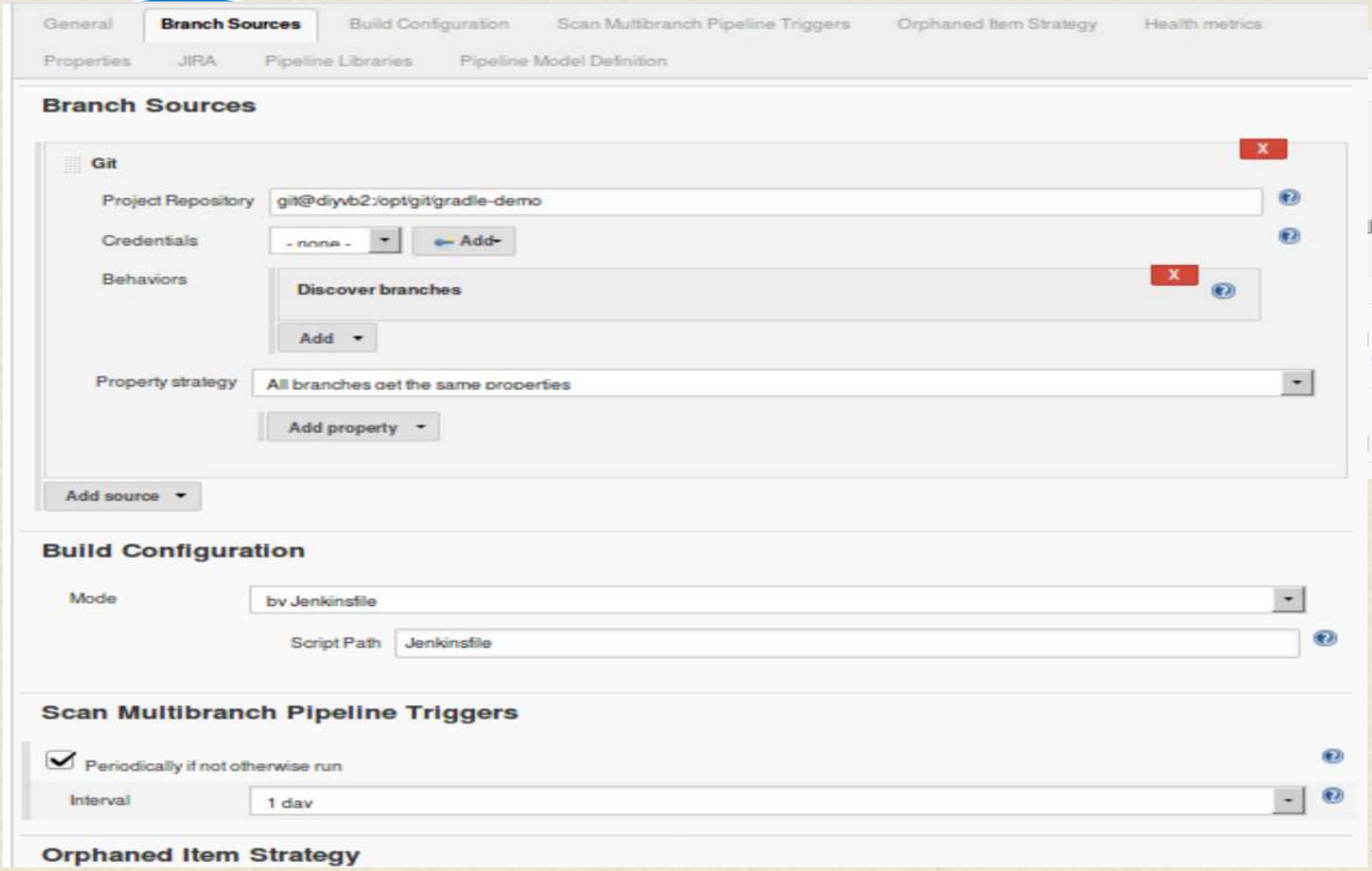

Pipeline Libraries

Sharable libraries available to any Pipeline jobs inside this folder. These libraries will be untrusted, meaning their code runs in the Groovy sandbox.

Multibranch Pipeline Scanning



Multibranch Pipeline Scanning



Branch Sources

Git

Project Repository: git@diyvb2:/opt/git/gradle-demo

Credentials: - none -

Behaviors: Discover branches

Property strategy: All branches get the same properties

Build Configuration

Mode: by Jenkinsfile

Script Path: Jenkinsfile

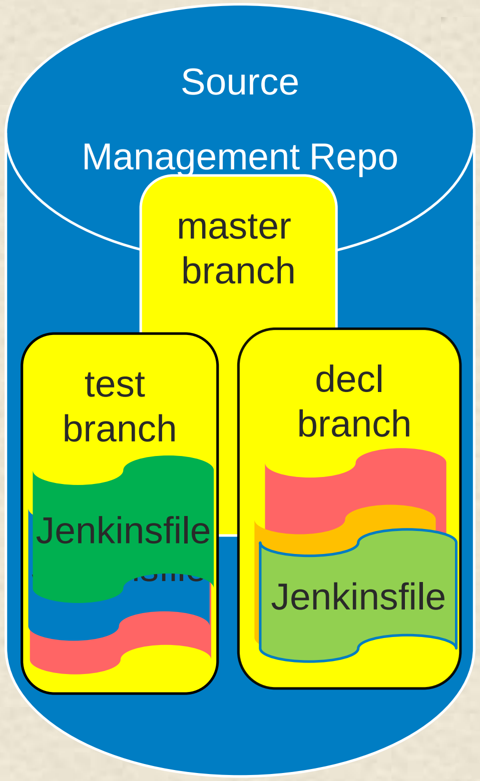
Scan Multibranch Pipeline Triggers

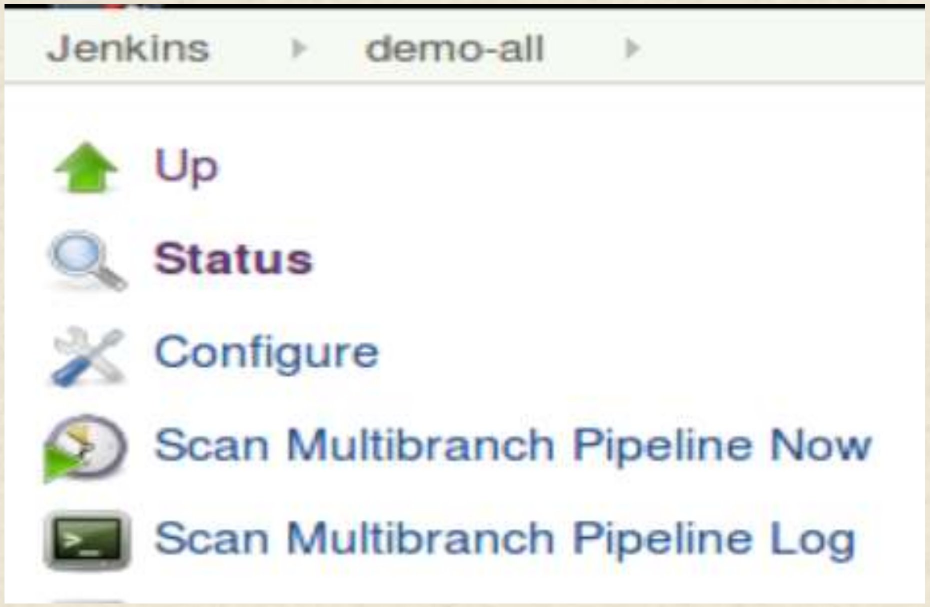
✓ Periodically if not otherwise run

Interval: 1 day

Orphaned Item Strategy

Multibranch Pipeline Scanning



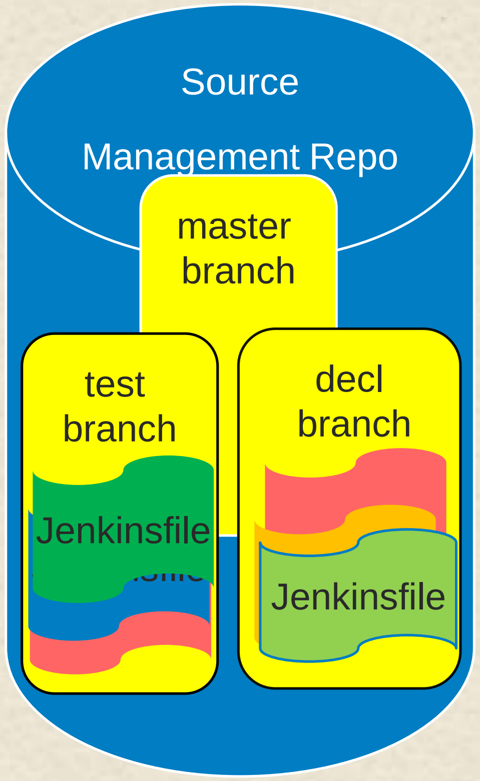


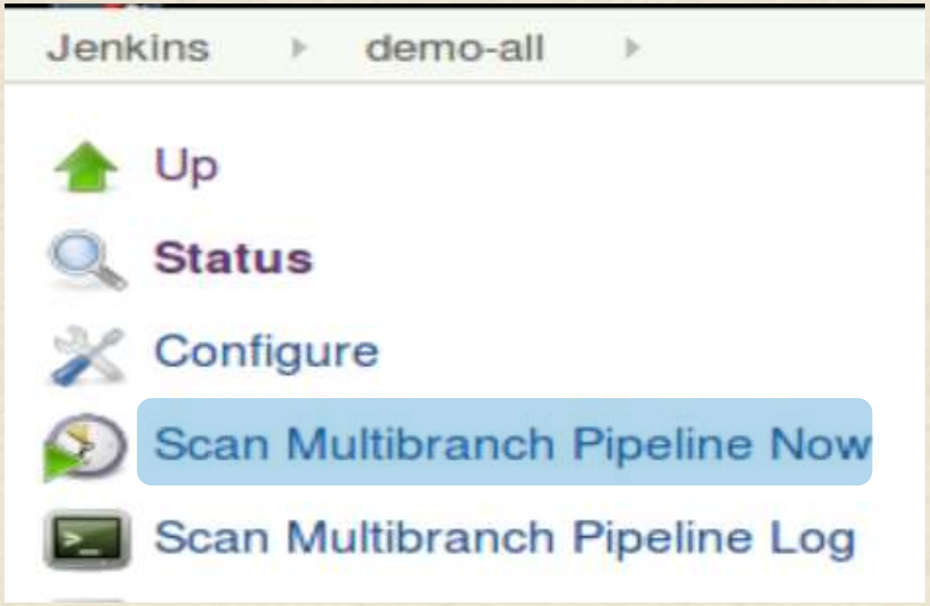
User creates and pushes a Jenkinsfile to desired branches in source control system.

User creates and configures a multibranch project.

Jenkins scans the project branches for ones with Jenkinsfiles.

Multibranch Pipeline Scanning



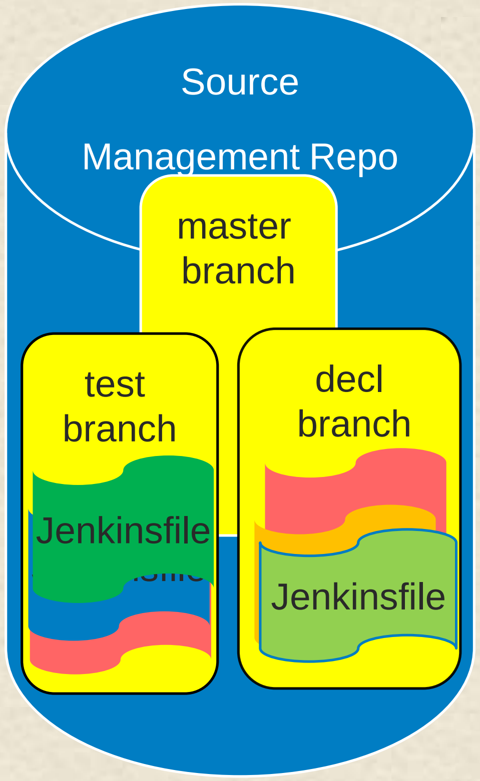


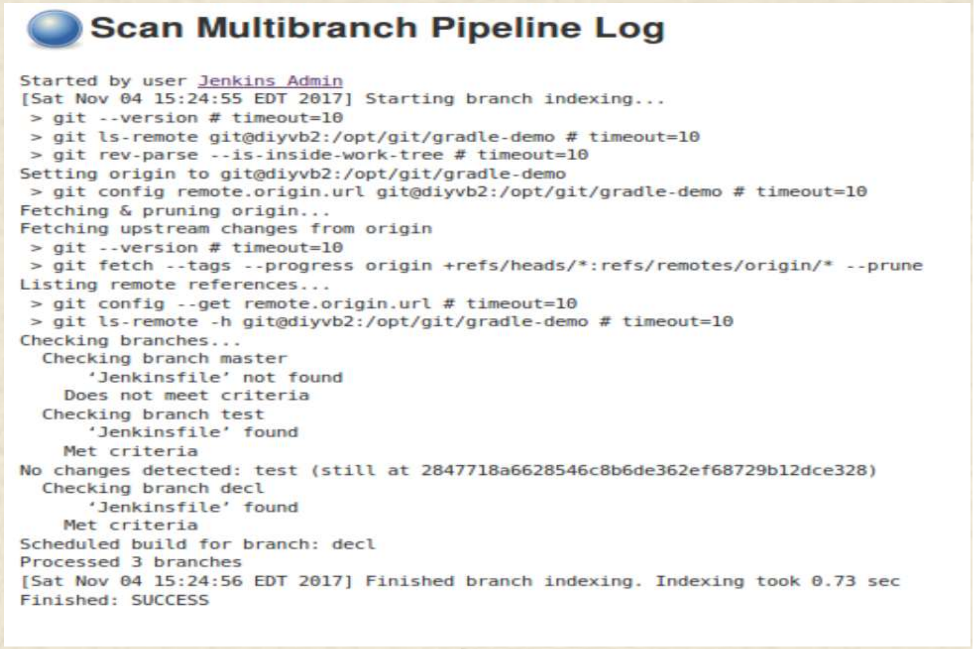
User creates and pushes a Jenkinsfile to desired branches in source control system.

User creates and configures a multibranch project.

Jenkins scans the project branches for ones with Jenkinsfiles.

Multibranch Pipeline Scanning





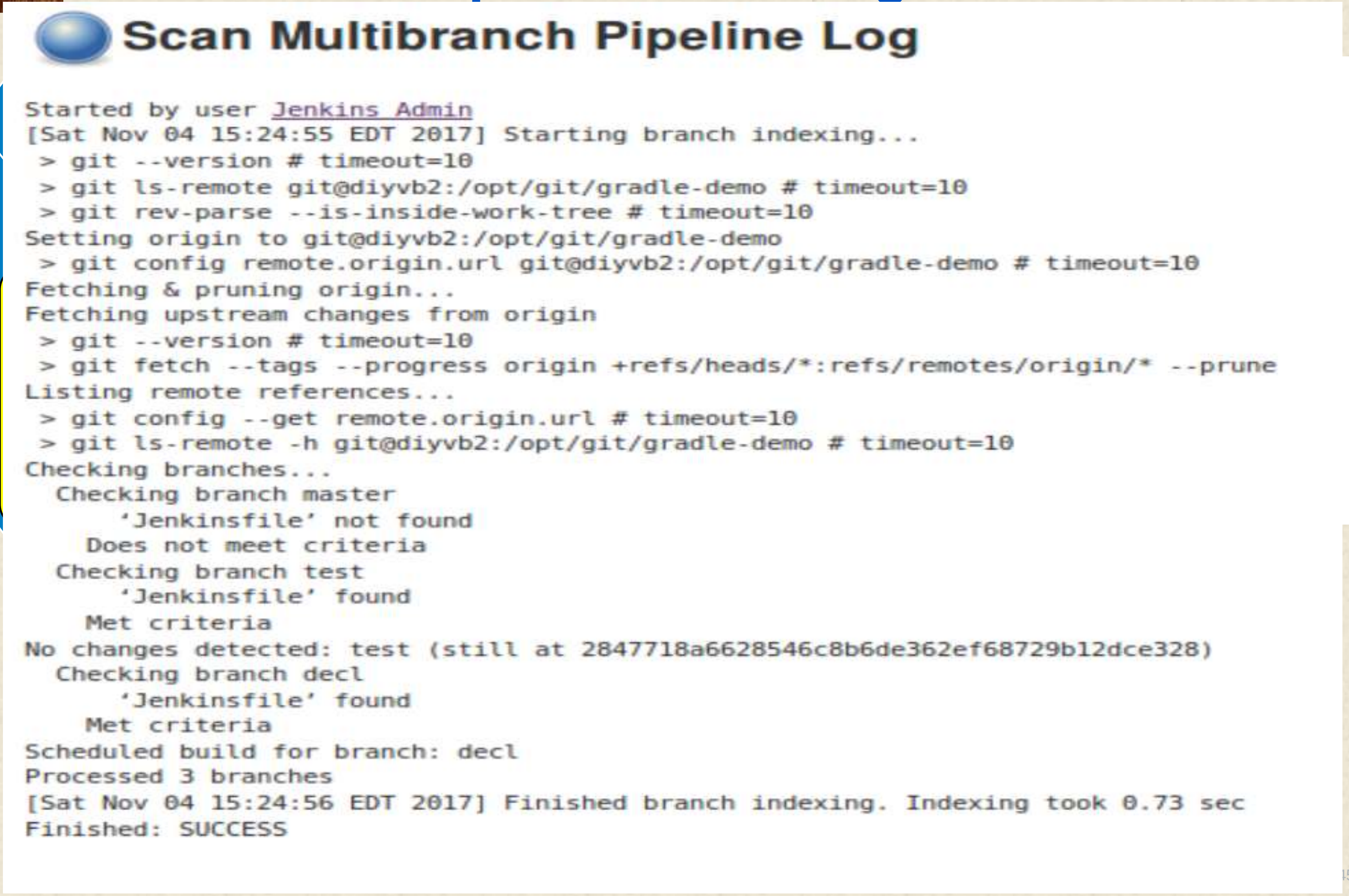
```
Started by user Jenkins Admin
[Sat Nov 04 15:24:55 EDT 2017] Starting branch indexing...
 > git --version # timeout=10
 > git ls-remote git@diyvb2:/opt/git/gradle-demo # timeout=10
 > git rev-parse --is-inside-work-tree # timeout=10
Setting origin to git@diyvb2:/opt/git/gradle-demo
 > git config remote.origin.url git@diyvb2:/opt/git/gradle-demo # timeout=10
Fetching & pruning origin...
Fetching upstream changes from origin
 > git --version # timeout=10
 > git fetch --tags --progress origin +refs/heads/*:refs/remotes/origin/* --prune
Listing remote references...
 > git config --get remote.origin.url # timeout=10
 > git ls-remote -h git@diyvb2:/opt/git/gradle-demo # timeout=10
Checking branches...
  Checking branch master
      'Jenkinsfile' not found
    Does not meet criteria
  Checking branch test
      'Jenkinsfile' found
    Met criteria
No changes detected: test (still at 2847718a6628546c8b6de362ef68729b12dce328)
  Checking branch decl
      'Jenkinsfile' found
    Met criteria
Scheduled build for branch: decl
Processed 3 branches
[Sat Nov 04 15:24:56 EDT 2017] Finished branch indexing. Indexing took 0.73 sec
Finished: SUCCESS
```

User creates and pushes a Jenkinsfile to desired branches in source control system.

User creates and configures a multibranch project.

Jenkins scans the project branches for ones with Jenkinsfiles.

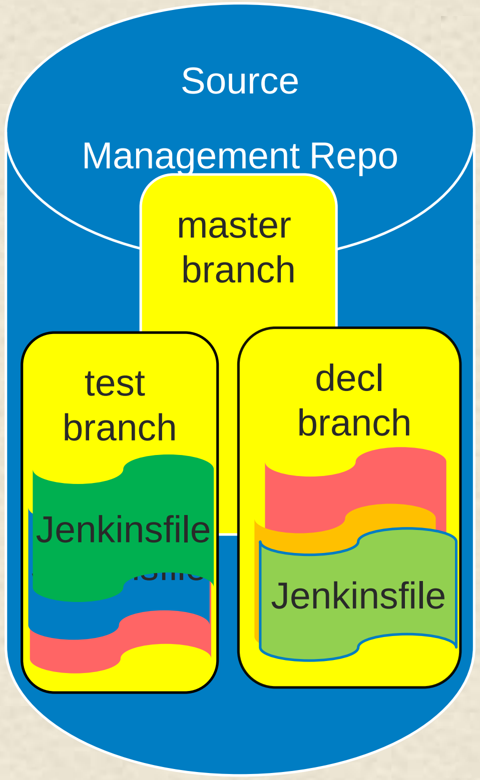
Multibranch Pipeline Scanning

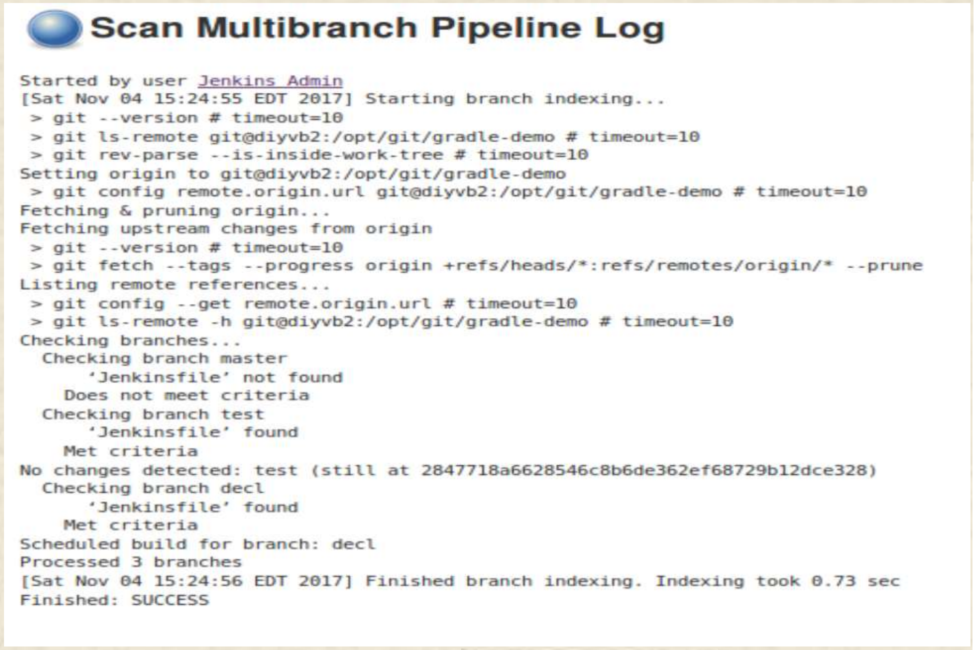


Scan Multibranch Pipeline Log

```
Started by user Jenkins Admin
[Sat Nov 04 15:24:55 EDT 2017] Starting branch indexing...
 > git --version # timeout=10
 > git ls-remote git@diyvb2:/opt/git/gradle-demo # timeout=10
 > git rev-parse --is-inside-work-tree # timeout=10
Setting origin to git@diyvb2:/opt/git/gradle-demo
 > git config remote.origin.url git@diyvb2:/opt/git/gradle-demo # timeout=10
Fetching & pruning origin...
Fetching upstream changes from origin
 > git --version # timeout=10
 > git fetch --tags --progress origin +refs/heads/*:refs/remotes/origin/* --prune
Listing remote references...
 > git config --get remote.origin.url # timeout=10
 > git ls-remote -h git@diyvb2:/opt/git/gradle-demo # timeout=10
Checking branches...
  Checking branch master
      'Jenkinsfile' not found
    Does not meet criteria
  Checking branch test
      'Jenkinsfile' found
    Met criteria
No changes detected: test (still at 2847718a6628546c8b6de362ef68729b12dce328)
  Checking branch decl
      'Jenkinsfile' found
    Met criteria
Scheduled build for branch: decl
Processed 3 branches
[Sat Nov 04 15:24:56 EDT 2017] Finished branch indexing. Indexing took 0.73 sec
Finished: SUCCESS
```

Multibranch Pipeline Scanning





Scan Multibranch Pipeline Log

```
Started by user Jenkins Admin
[Sat Nov 04 15:24:55 EDT 2017] Starting branch indexing...
 > git --version # timeout=10
 > git ls-remote git@diyvb2:/opt/git/gradle-demo # timeout=10
 > git rev-parse --is-inside-work-tree # timeout=10
Setting origin to git@diyvb2:/opt/git/gradle-demo
 > git config remote.origin.url git@diyvb2:/opt/git/gradle-demo # timeout=10
Fetching & pruning origin...
Fetching upstream changes from origin
 > git --version # timeout=10
 > git fetch --tags --progress origin +refs/heads/*:refs/remotes/origin/* --prune
Listing remote references...
 > git config --get remote.origin.url # timeout=10
 > git ls-remote -h git@diyvb2:/opt/git/gradle-demo # timeout=10
Checking branches...
  Checking branch master
      'Jenkinsfile' not found
    Does not meet criteria
  Checking branch test
      'Jenkinsfile' found
    Met criteria
No changes detected: test (still at 2847718a6628546c8b6de362ef68729b12dce328)
  Checking branch decl
      'Jenkinsfile' found
    Met criteria
Scheduled build for branch: decl
Processed 3 branches
[Sat Nov 04 15:24:56 EDT 2017] Finished branch indexing. Indexing took 0.73 sec
Finished: SUCCESS
```

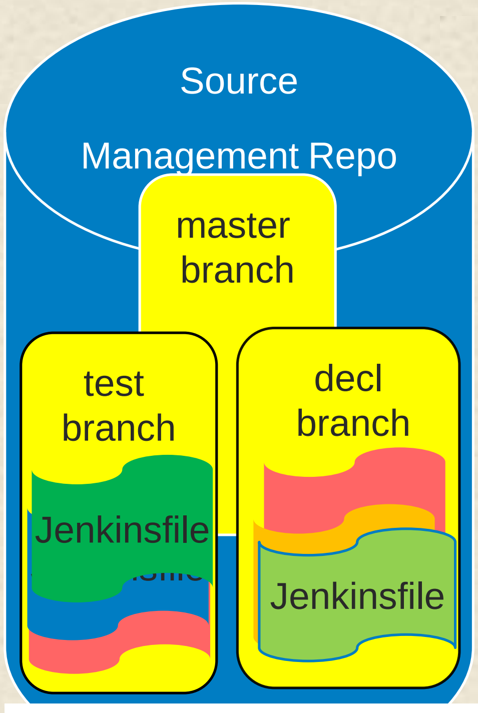
User creates and pushes a Jenkinsfile to desired branches in source control system.

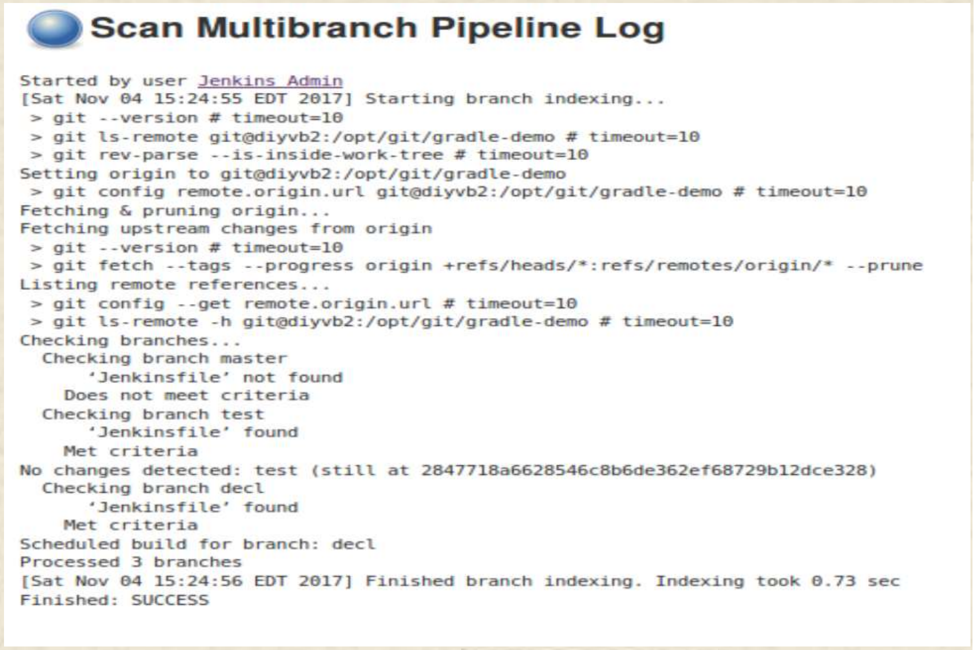
User creates and configures a multibranch project.

Jenkins scans the project branches for ones with Jenkinsfiles.

Jenkins creates new jobs for each branch having a Jenkinsfile and executes the jobs

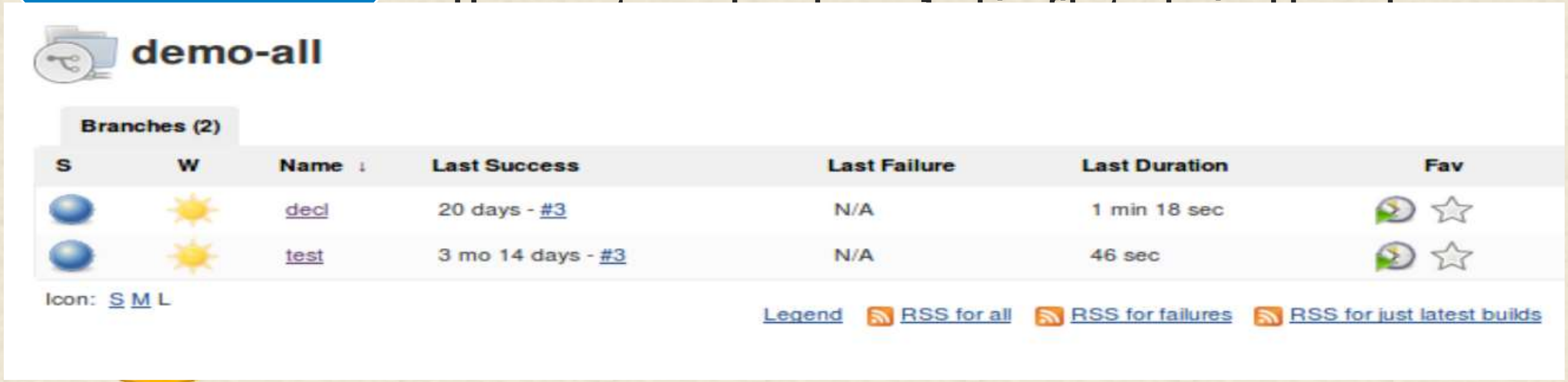
Multibranch Pipeline Scanning





Scan Multibranch Pipeline Log

```
Started by user Jenkins Admin
[Sat Nov 04 15:24:55 EDT 2017] Starting branch indexing...
 > git --version # timeout=10
 > git ls-remote git@diyvb2:/opt/git/gradle-demo # timeout=10
 > git rev-parse --is-inside-work-tree # timeout=10
Setting origin to git@diyvb2:/opt/git/gradle-demo
 > git config remote.origin.url git@diyvb2:/opt/git/gradle-demo # timeout=10
Fetching & pruning origin...
Fetching upstream changes from origin
 > git --version # timeout=10
 > git fetch --tags --progress origin +refs/heads/*:refs/remotes/origin/* --prune
Listing remote references...
 > git config --get remote.origin.url # timeout=10
 > git ls-remote -h git@diyvb2:/opt/git/gradle-demo # timeout=10
Checking branches...
  Checking branch master
      'Jenkinsfile' not found
    Does not meet criteria
  Checking branch test
      'Jenkinsfile' found
    Met criteria
No changes detected: test (still at 2847718a6628546c8b6de362ef68729b12dce328)
  Checking branch decl
      'Jenkinsfile' found
    Met criteria
Scheduled build for branch: decl
Processed 3 branches
[Sat Nov 04 15:24:56 EDT 2017] Finished branch indexing. Indexing took 0.73 sec
Finished: SUCCESS
```



demo-all

Branches (2)

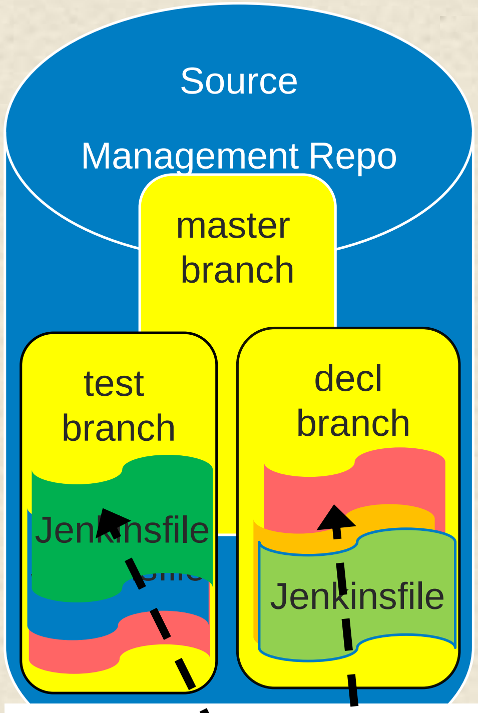
S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
●	☀	decl	20 days - #3	N/A	1 min 18 sec	☆
●	☀	test	3 mo 14 days - #3	N/A	46 sec	☆

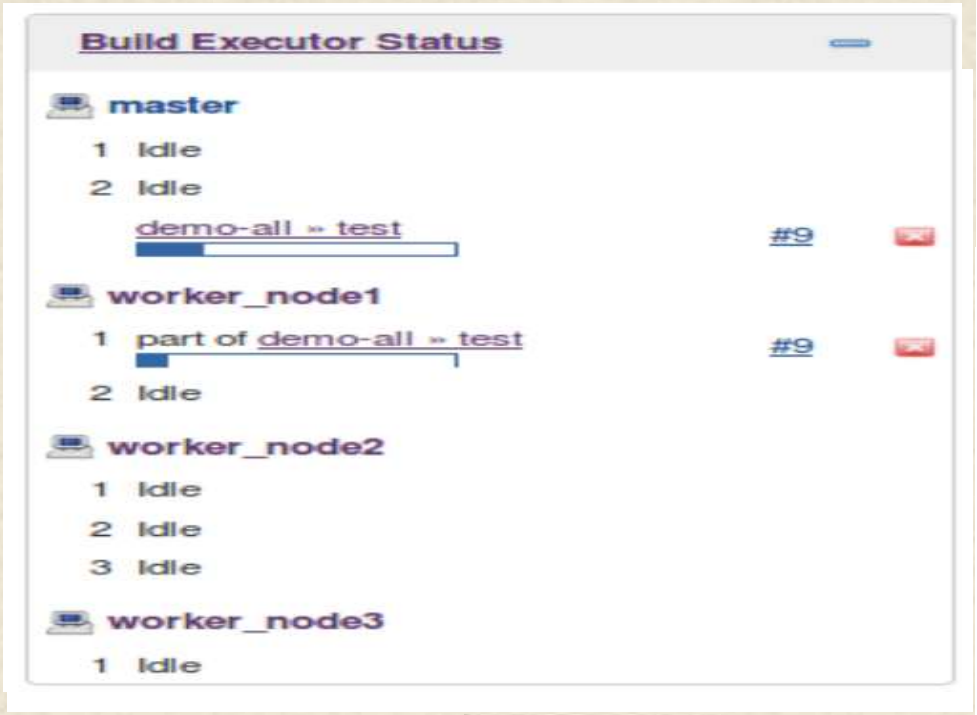
Icon: S M L

Legend   RSS for all   RSS for failures   RSS for just latest builds

Jenkinsfile and executes the jobs

Multibranch Pipeline Scanning





Build Executor Status

master

1	Idle
2	Idle
	demo-all » test #9

worker_node1

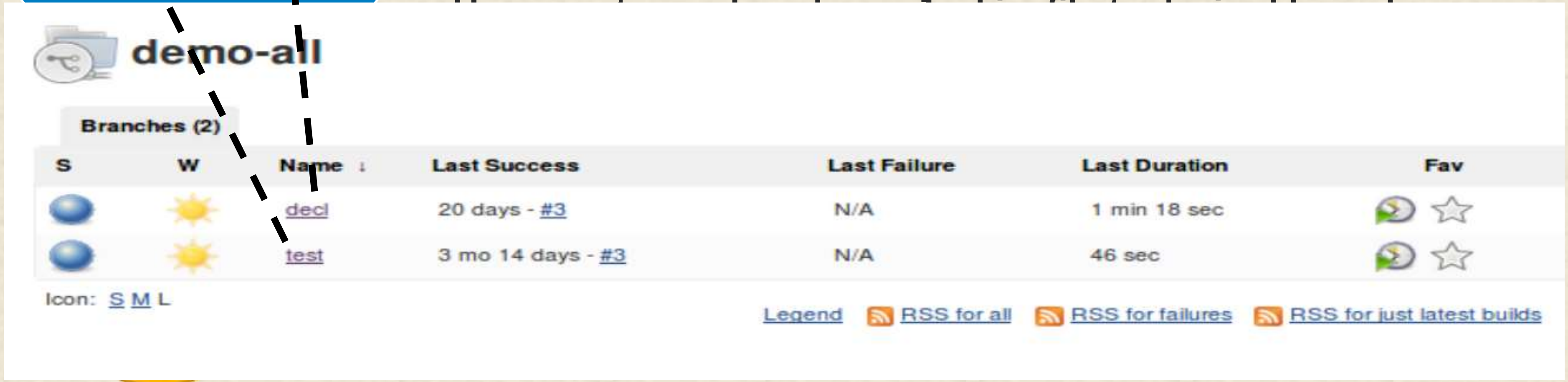
1	part of demo-all » test #9
2	Idle

worker_node2

1	Idle
2	Idle
3	Idle

worker_node3

1	Idle



demo-all

Branches (2)

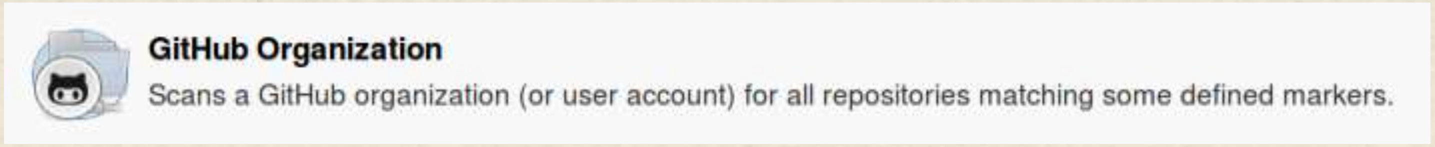
S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
●	☀	decl	20 days - #3	N/A	1 min 18 sec	☆
●	☀	test	3 mo 14 days - #3	N/A	46 sec	☆

Icon: S M L

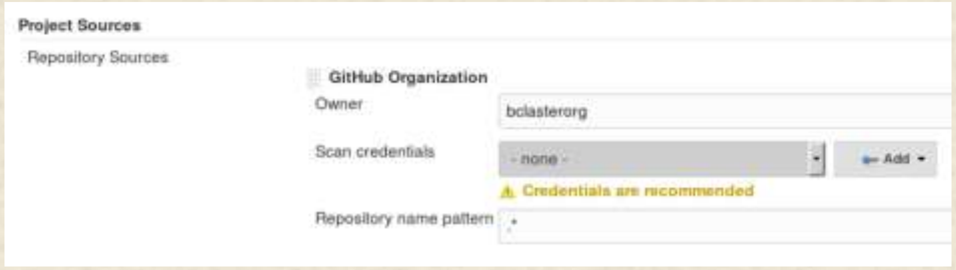
Legend RSS for all RSS for failures RSS for just latest builds

Jenkinsfile and executes the jobs

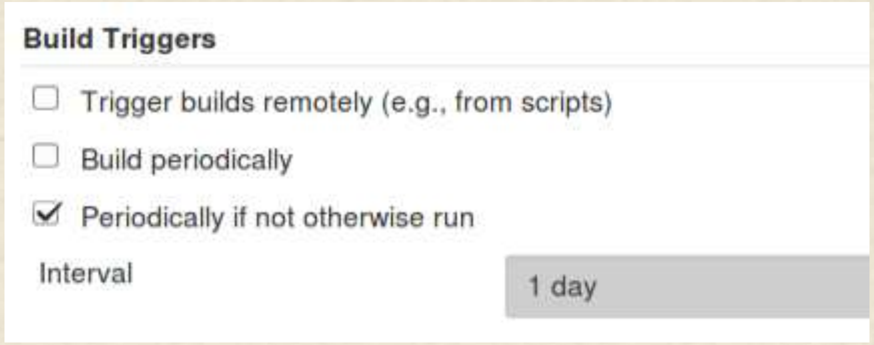
Github Organization Project


GitHub Organization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

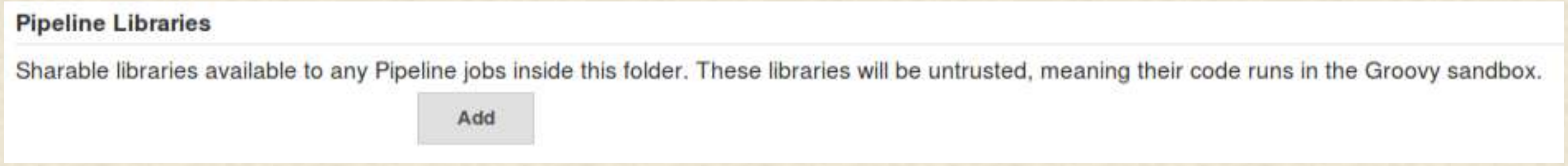
- Automatically detects repositories in organization
- Creates multibranch projects for each repository
- Monitors activity in projects/branches with Jenkinsfile
- Automatically sets up “organization webhook” on Github and react to webhook posting back to Jenkins system
- Supports shared pipeline libraries for projects in organization


Project Sources
Repository Sources
GitHub Organization
Owner: bclasterorg
Scan credentials: - none -
Credentials are recommended
Repository name pattern: .*

Project Recognizers
Pipeline Jenkinsfile

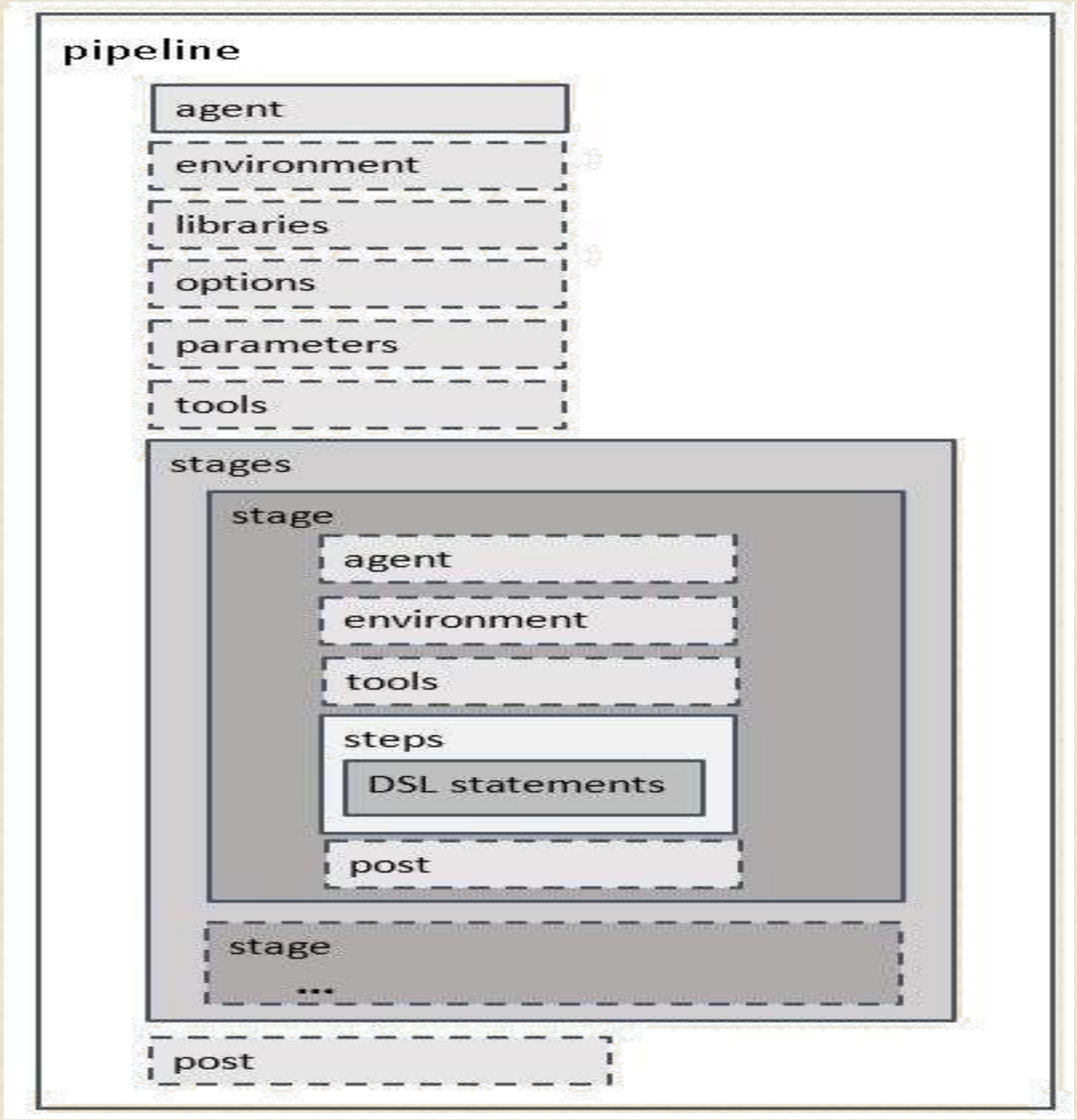

Build Triggers
- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build periodically
- [x] Periodically if not otherwise run

Interval: 1 day


Pipeline Libraries
Sharable libraries available to any Pipeline jobs inside this folder. These libraries will be untrusted, meaning their code runs in the Groovy sandbox.

Declarative Pipelines – Motivation and Form

- Scripted DSL is good, but not as intuitive
- Scripted DSL feels like you need to know Groovy
- Extra processing required for things we used to get for free
- Still part of pipeline - can be entered in script window or Jenkinsfiles
- Simpler syntax
- Improved error checking/code validation
- Set of declarative “directives” and “sections” for various components (current list)



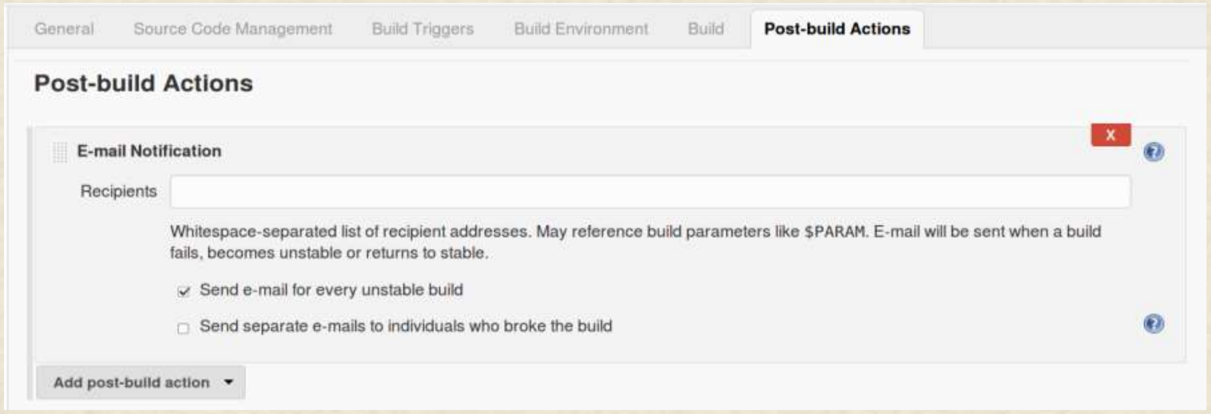
Scripted vs. Declarative Syntax

```
#!groovy
@Library('Utilities@1.5')_
node ('worker_node1') {
try {
     stage('Source') {
        // always run with a new workspace
        cleanupWs()
        checkout scm
        stash name: 'test-sources', includes: 'build.gradle,src/test/'
     }
     stage('Build') {
        // Run the gradle build
        gbuild2 'clean build -x test'
     }
}
 catch (err) {
     echo "Caught: ${err}"
  }
  stage ('Notify') {
     // mailUser('<your email address>', "Finished")
  }
}
```

```
#!groovy
pipeline {
   agent{ label 'worker_node1'}
   libraries {
      lib('Utilities@1.5')
  }
  stages {
     stage('Source') {
        steps {
           cleanWs()
           checkout scm
           stash name: 'test-sources', includes: 'build.gradle,src/test/'
        }
     }
   stage('Build') {
        // Run the gradle build
        steps {
            gbuild2 'clean build -x test'
         }
      }
} // end stages
 post {
      always {
        echo "Build stage complete"
      }
      failure{
        echo "Build failed"
        mail body: 'build failed', subject: 'Build failed!', to: '<your email address>'
      }
      success {
         echo "Build succeeded"
         mail body: 'build succeeded', subject: 'Build Succeeded', to: '<your email address>'
      }
   }
} // end pipeline
```

FreeStyle vs. Scripted vs Declarative

- Example: post processing – such as always sending mail



Post-build Actions

E-mail Notification

Recipients

Whitespace-separated list of recipient addresses. May reference build parameters like $PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

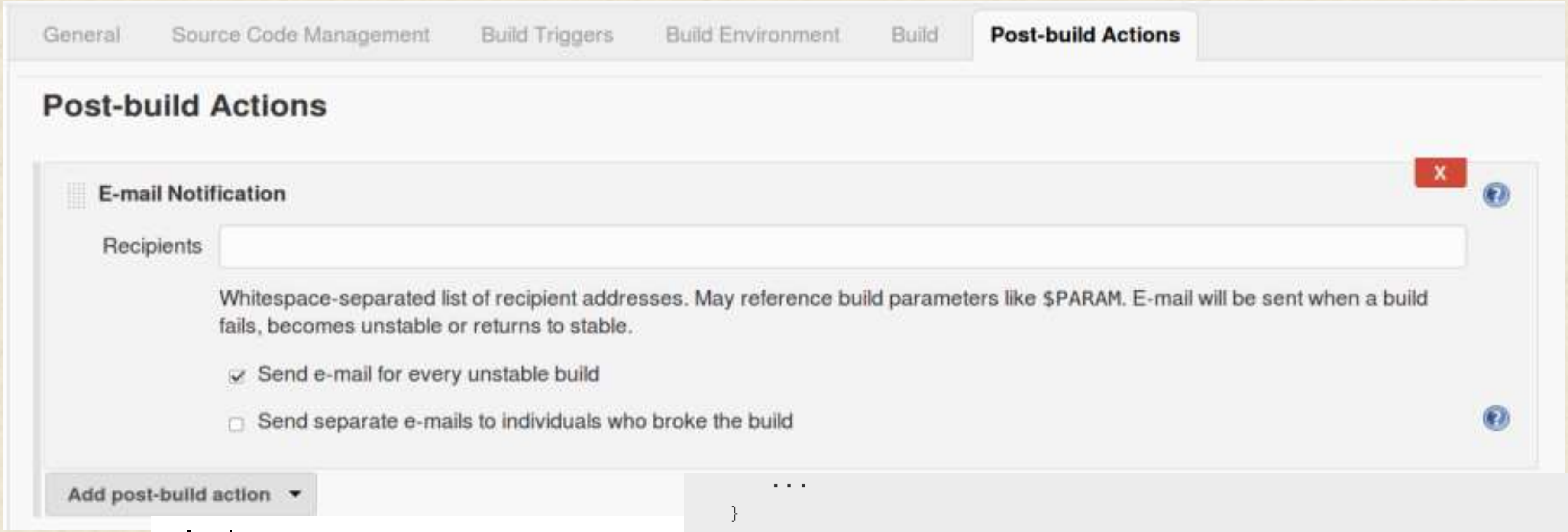
☑ Send e-mail for every unstable build

☐ Send separate e-mails to individuals who broke the build

```
node {
  try {
    sendEmailStarted()
    stage('Source') {...}
    stage('Build')  {...}
    ...
    sendEmailSuccess()
  } catch (err) {
    currentBuild.result = "FAILED"
    sendEmailFail()
    throw err
  }
}
```

```
      ...
   }
   post {
        always {
          echo "Build stage complete"
        }
        // changed means when the build status is different than the p
       failure{
          echo "Build failed"
          mail body: <some text>, subject: 'Build failed!', to: 'devop
        }
        success {
          echo "Build succeeded"
          archiveArtifacts '**/*'
        }
      }
 }
```

FreeStyle vs. Scripted vs Declarative


General | Source Code Management | Build Triggers | Build Environment | Build | **Post-build Actions**

Post-build Actions

E-mail Notification

Recipients

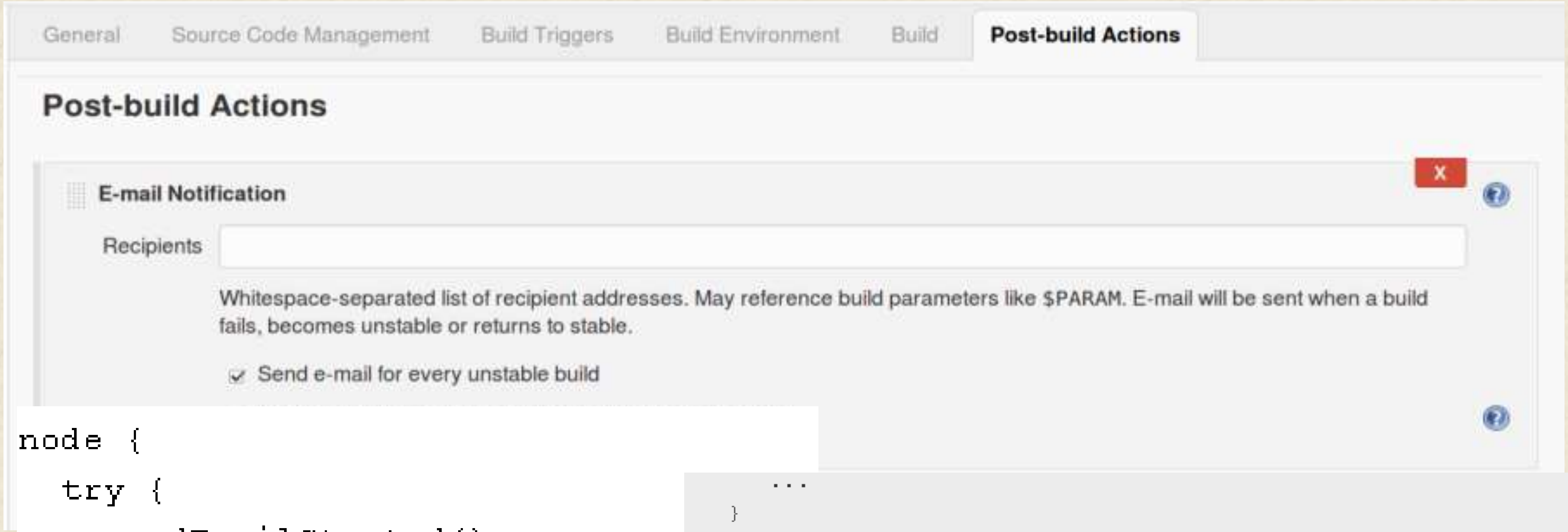
Whitespace-separated list of recipient addresses. May reference build parameters like $PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

- [✓] Send e-mail for every unstable build
- [ ] Send separate e-mails to individuals who broke the build

```
node {
  try {
    sendEmailStarted()
    stage('Source') {...}
    stage('Build')  {...}
    ...
    sendEmailSuccess()
  } catch (err) {
    currentBuild.result = "FAILED"
    sendEmailFail()
    throw err
  }
}
```

```
      ...
  }
  post {
       always {
         echo "Build stage complete"
       }
       // changed means when the build status is different than the p
      failure{
         echo "Build failed"
         mail body: <some text>, subject: 'Build failed!', to: 'devop
       }
       success {
         echo "Build succeeded"
         archiveArtifacts '**/*'
       }
     }
}
```

FreeStyle vs. Scripted vs Declarative



General | Source Code Management | Build Triggers | Build Environment | Build | **Post-build Actions**

Post-build Actions

E-mail Notification

Recipients

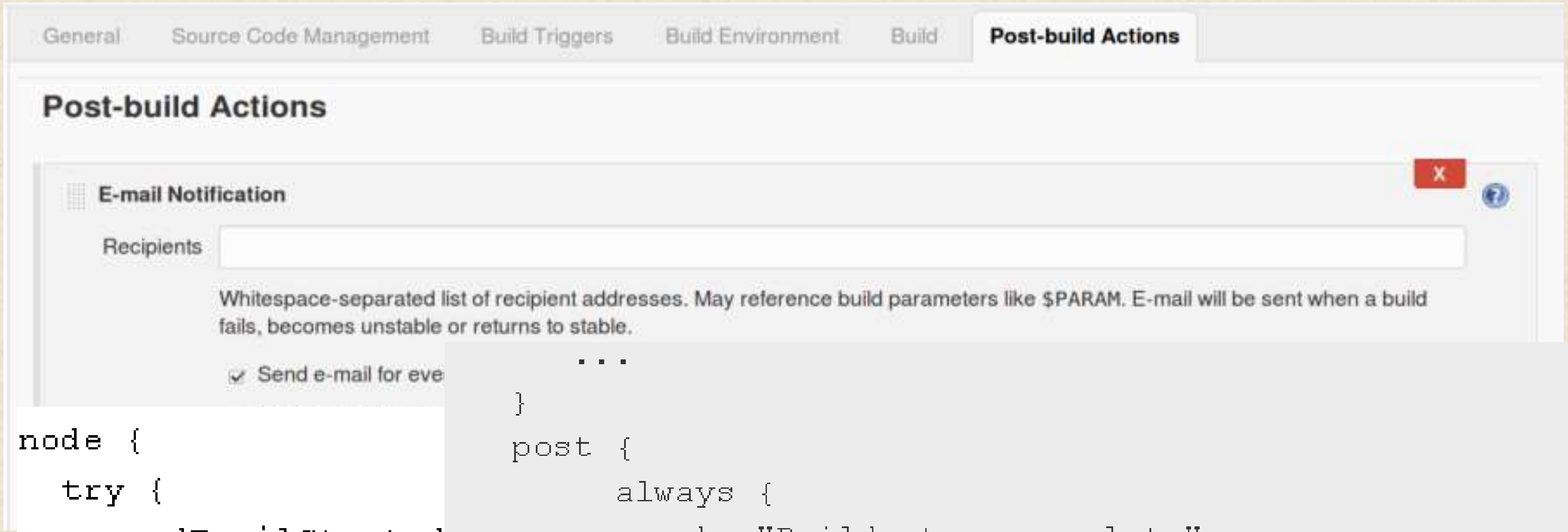
Whitespace-separated list of recipient addresses. May reference build parameters like $PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

[✓] Send e-mail for every unstable build

```
node {
  try {
    sendEmailStarted()
    stage('Source') {...}
    stage('Build')  {...}
    ...
    sendEmailSuccess()
  } catch (err) {
    currentBuild.result = "FAIL
    sendEmailFail()
    throw err
  }
}
```

```
        ...
    }
    post {
          always {
            echo "Build stage complete"
          }
          // changed means when the build status is different than the p
         failure{
            echo "Build failed"
            mail body: <some text>, subject: 'Build failed!', to: 'devop
          }
          success {
            echo "Build succeeded"
            archiveArtifacts '**/*'
          }
        }
  }
```

FreeStyle vs. Scripted vs Declarative



General | Source Code Management | Build Triggers | Build Environment | Build | **Post-build Actions**

Post-build Actions

E-mail Notification

Recipients

Whitespace-separated list of recipient addresses. May reference build parameters like $PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

✓ Send e-mail for eve

```
node {
  try {
    sendEmailStarted
    stage('Source')
    stage('Build')
    ...
    sendEmailSuccess
  } catch (err) {
    currentBuild.res
    sendEmailFail()
    throw err
  }
}
```

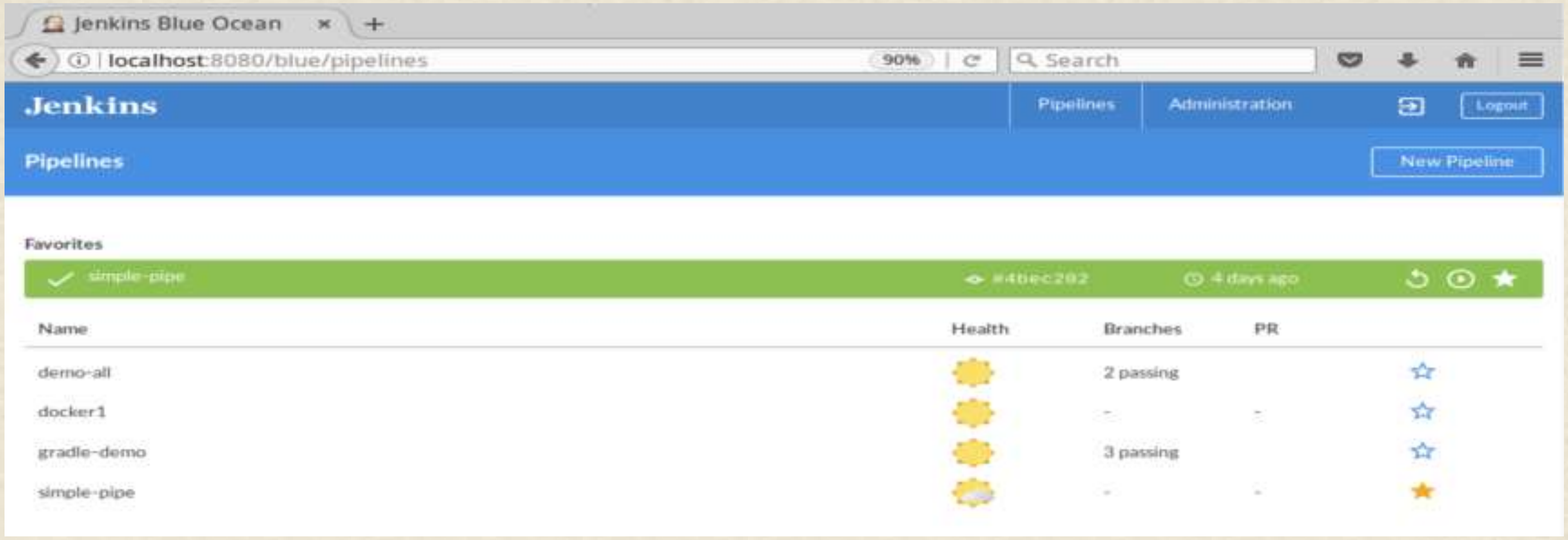
```
        ...
    }
    post {
        always {
          echo "Build stage complete"
        }
        // changed means when the build status is different
       failure{
          echo "Build failed"
          mail body: <some text>, subject: 'Build failed!',
        }
        success {
          echo "Build succeeded"
          archiveArtifacts '**/*'
        }
      }
}
```

Blue Ocean

- New Jenkins interface
- Support for graphically representing pipelines
- Based on stages definitions
- Full functionality requires declarative pipeline
- Shows processing, success, failure of stages
- Can view logs segmented by steps
- Can be invoked by left menu item or http:<jenkins url>/blue
- Visual pipeline creator/editor – allows for creating pipeline around existing code base

Pipelines page

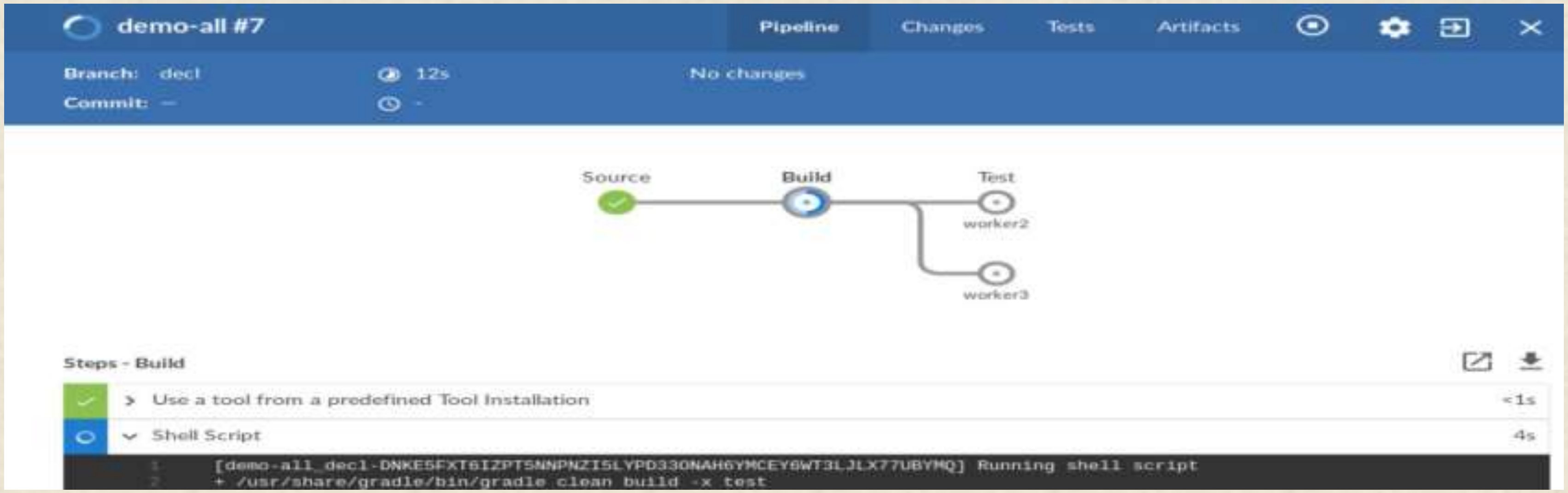
- List all pipelines and health, etc.
- Like Jenkins dashboard page
- Can drill in from here
- Also can select favorites



Name	Health	Branches	PR	
demo-all	☀	2 passing		☆
docker1	☀	-	-	☆
gradle-demo	☀	3 passing		☆
simple-pipe	☀	-	-	★

Run in progress

- Green checks indicate successfully completed stages
- Partially filled/outlined circles represent stages in progress
- Empty circles represent stages waiting to be done

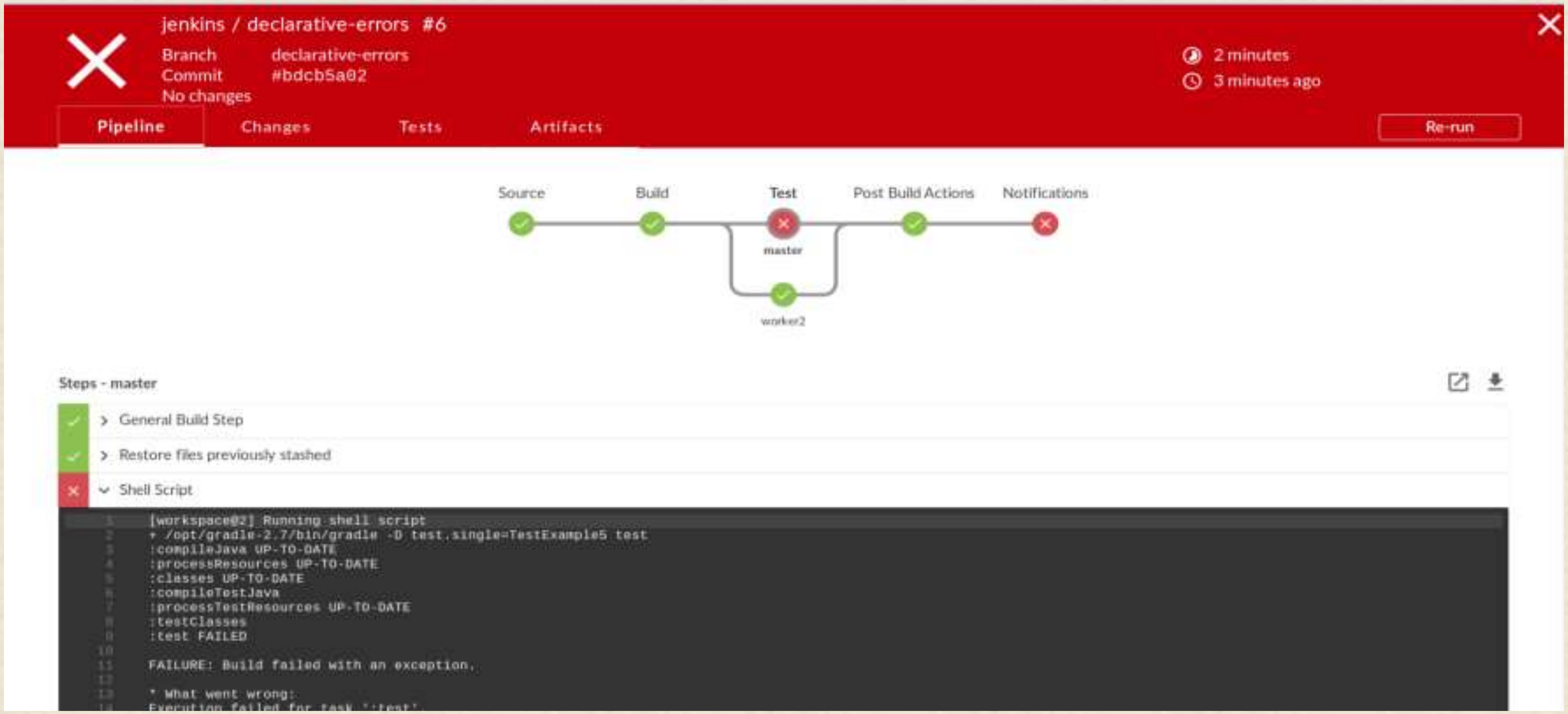


✓	> Use a tool from a predefined Tool Installation	<1s
○	⌄ Shell Script	4s

```
[demo-all_decl-DNKE5FXT6IZPT5NNPNZI5LYPD33ONAH6YMCEY6WT3LJLX77UBYMQ] Running shell script
+ /usr/share/gradle/bin/gradle clean build -x test
```

Failed Build Step

- Individual steps marked as failed
- Logs accessible
- Note "Re-run" button



jenkins / declarative-errors #6

Branch declarative-errors
Commit #bdcb5a02
No changes

2 minutes
3 minutes ago

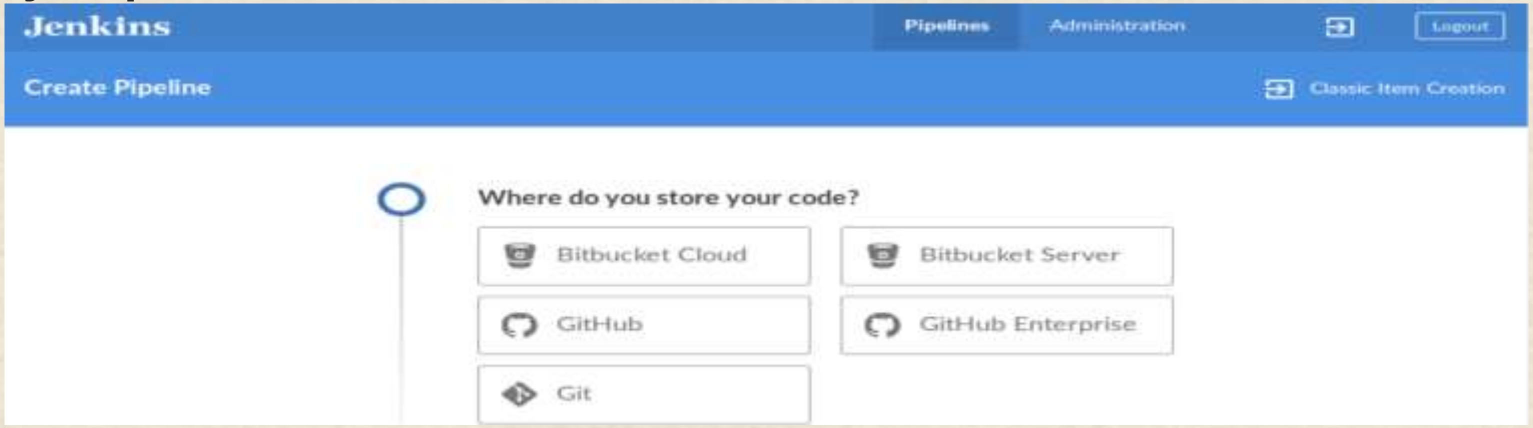
Steps - master

- ✓ General Build Step
- ✓ Restore files previously stashed
- ✗ Shell Script

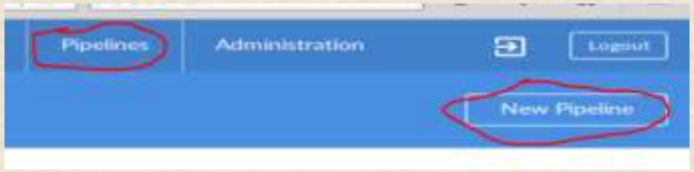
```
[workspace@2] Running shell script
+ /opt/gradle-2.7/bin/gradle -D test.single=TestExample5 test
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:compileTestJava
:processTestResources UP-TO-DATE
:testClasses
:test FAILED

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':test'.
```

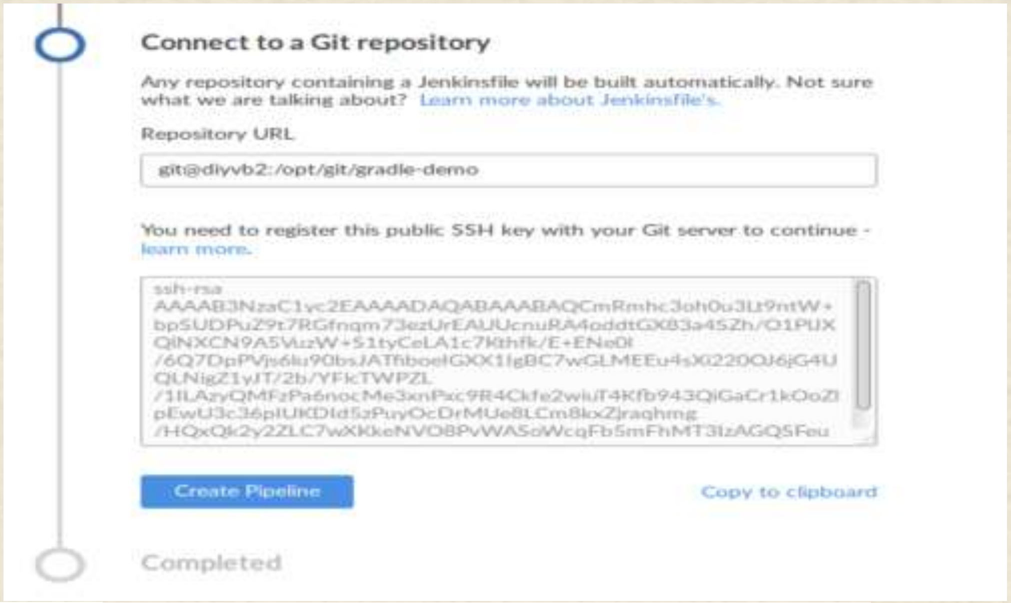
New Pipeline

- Allows you to create a new pipeline around existing code base
- Easy for existing multibranch pipelines
- May require authorization



Where do you store your code?





Connect to a Git repository

Any repository containing a Jenkinsfile will be built automatically. Not sure what we are talking about? Learn more about Jenkinsfile's.

Repository URL

git@diyvb2:/opt/git/gradle-demo

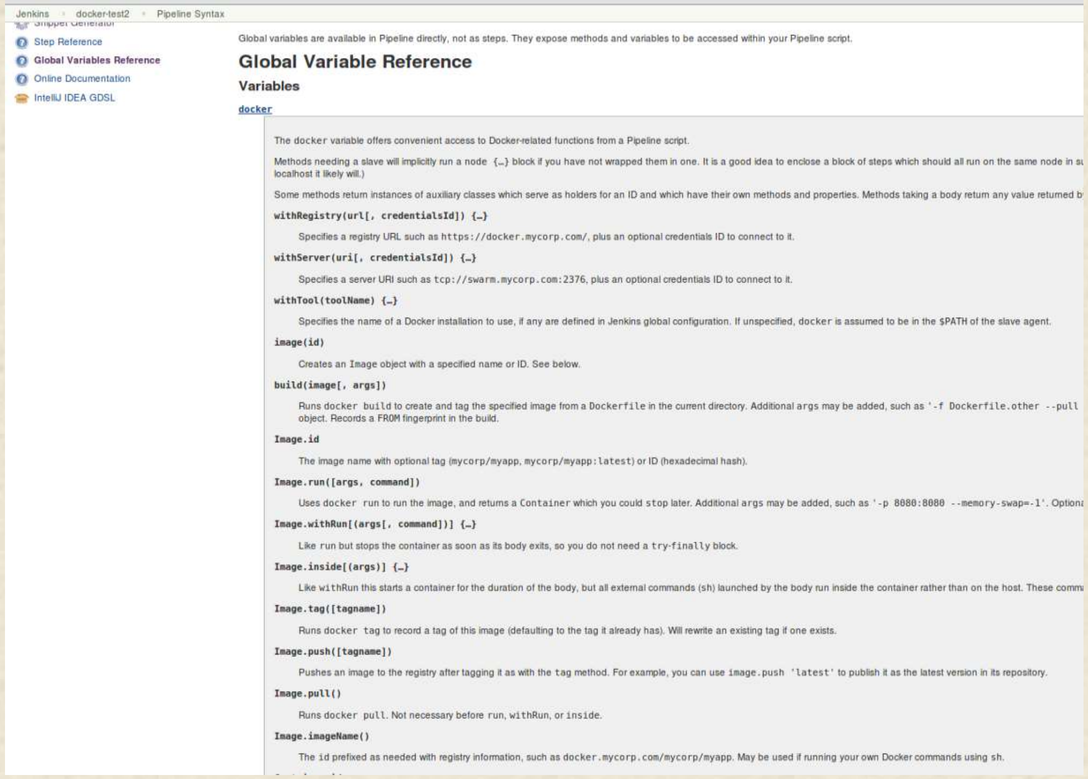
You need to register this public SSH key with your Git server to continue - learn more.

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCmRmhc3oh0u3Ut9ntW+
bpSUDPuZ9t7RGfnqm73ezUrEAUUcnuRA4oddtGX83a45Zh/O1PUX
QINXCN9A5VuzW+51tyCeLA1c7Kthfk/E+ENeDI
/6Q7DpPVjs6lu90bsJATfiboeIGXX1IgBC7wGLMEEu4sX0220OJ6jG4U
QLNigZ1yJT/2b/YFkTWPZL
/1ILAzyQMFzPa6nocMe3xnPxc9R4Ckfe2wiuT4Kfb943QiGaCr1kOoZI
pEwU3c36pIUKDId5zPuyOcDrMUe8LCm8kxZjraqhmg
/HQxQk2y2ZLC7wXKkeNVO8PvWASoWcqFb5mFhMT3IzAGQ5Feu
```

Jenkins 2 and Docker

- 4 ways of running Docker via Jenkins
  - Configured as a “cloud” (via Docker plugin)
    - » Global configuration points Jenkins to a Docker image that works as a node
    - » Jenkins can start/stop containers based on the image automatically as needed to do work (dynamically start up nodes)
  - Use global variable “docker” (via Docker pipeline plugin)
    - » Point to an image – invoke “inside” method
    - » Get image (if not already there), spins it up, provides access to workspace), performs tasks, and gets rid of it
  - As “agent” for declarative pipelines
    - » Can be pointed to input file (Dockerfile) and automatically “build” a docker image to run on
  - Directly vs shell call
    - » Uses “sh” shell call to run docker executable

Jenkins 2.0 and Docker – Inside example



Jenkins › docker-test2 › Pipeline Syntax

Snippet Generator
Step Reference
Global Variables Reference
Online Documentation
IntelliJ IDEA GDSL

Global variables are available in Pipeline directly, not as steps. They expose methods and variables to be accessed within your Pipeline script.

Global Variable Reference

Variables

docker

The docker variable offers convenient access to Docker-related functions from a Pipeline script.

Methods needing a slave will implicitly run a node {…} block if you have not wrapped them in one. It is a good idea to enclose a block of steps which should all run on the same node in su localhost it likely will.)

Some methods return instances of auxiliary classes which serve as holders for an ID and which have their own methods and properties. Methods taking a body return any value returned b

withRegistry(url[, credentialsId]) {…}

Specifies a registry URL such as `https://docker.mycorp.com/`, plus an optional credentials ID to connect to it.

withServer(uri[, credentialsId]) {…}

Specifies a server URI such as `tcp://swarm.mycorp.com:2376`, plus an optional credentials ID to connect to it.

withTool(toolName) {…}

Specifies the name of a Docker installation to use, if any are defined in Jenkins global configuration. If unspecified, `docker` is assumed to be in the `$PATH` of the slave agent.

image(id)

Creates an `Image` object with a specified name or ID. See below.

build(image[, args])

Runs `docker build` to create and tag the specified image from a `Dockerfile` in the current directory. Additional `args` may be added, such as `'-f Dockerfile.other --pull` object. Records a `FROM` fingerprint in the build.

Image.id

The image name with optional tag (`mycorp/myapp`, `mycorp/myapp:latest`) or ID (hexadecimal hash).

Image.run([args, command])

Uses `docker run` to run the image, and returns a `Container` which you could `stop` later. Additional `args` may be added, such as `'-p 8080:8080 --memory-swap=-1'`. Optiona

Image.withRun[(args[, command])] {…}

Like `run` but stops the container as soon as its body exits, so you do not need a `try-finally` block.

Image.inside[(args)] {…}

Like `withRun` this starts a container for the duration of the body, but all external commands (sh) launched by the body run inside the container rather than on the host. These comm

Image.tag([tagname])

Runs `docker tag` to record a tag of this image (defaulting to the tag it already has). Will rewrite an existing tag if one exists.

Image.push([tagname])

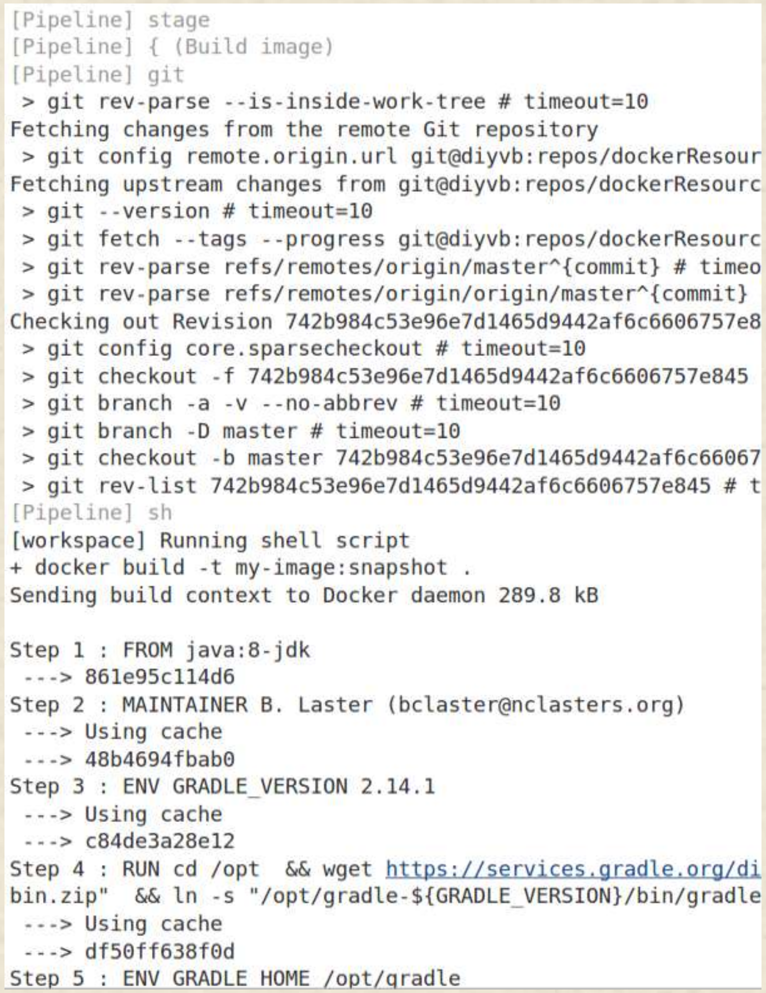
Pushes an image to the registry after tagging it as with the `tag` method. For example, you can use `image.push 'latest'` to publish it as the latest version in its repository.

Image.pull()

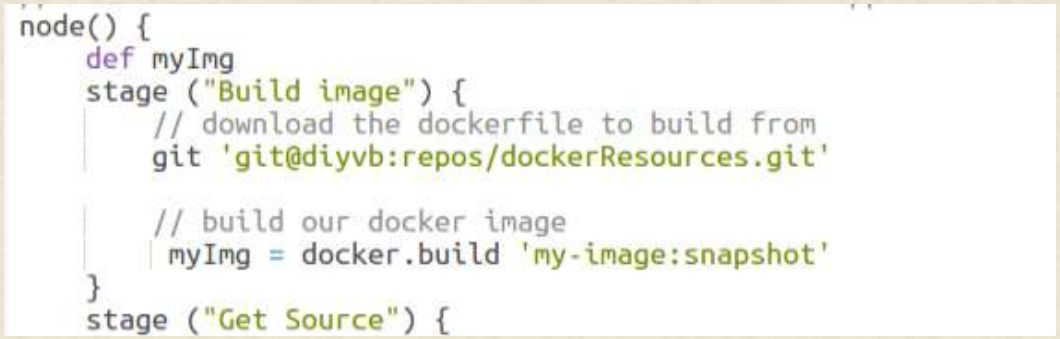
Runs `docker pull`. Not necessary before `run`, `withRun`, or `inside`.

Image.imageName()

The `id` prefixed as needed with registry information, such as `docker.mycorp.com/mycorp/myapp`. May be used if running your own Docker commands using `sh`.

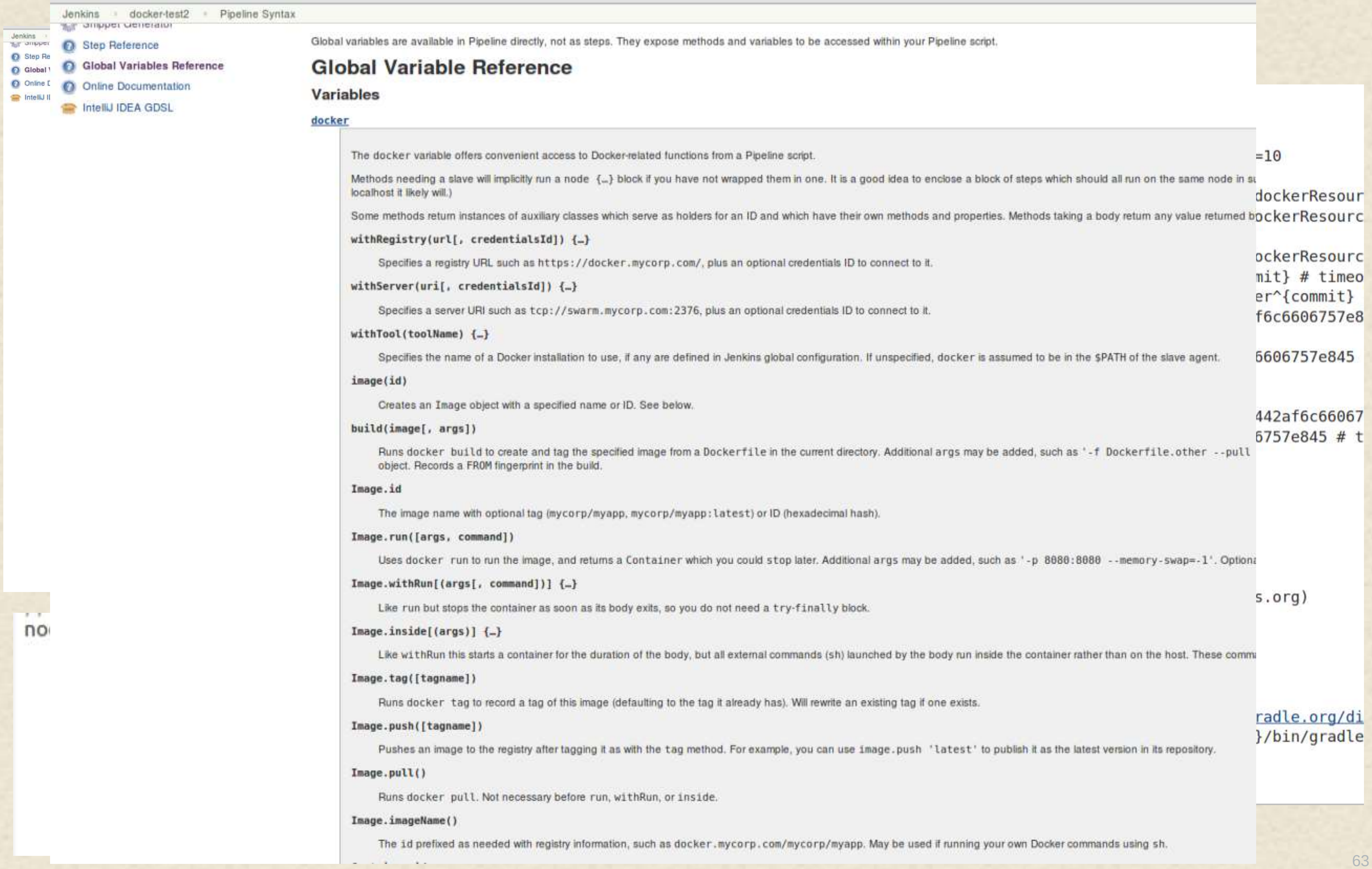


```
[Pipeline] stage
[Pipeline] { (Build image)
[Pipeline] git
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url git@diyvb:repos/dockerResour
Fetching upstream changes from git@diyvb:repos/dockerResourc
 > git --version # timeout=10
 > git fetch --tags --progress git@diyvb:repos/dockerResourc
 > git rev-parse refs/remotes/origin/master^{commit} # timeo
 > git rev-parse refs/remotes/origin/origin/master^{commit}
Checking out Revision 742b984c53e96e7d1465d9442af6c6606757e8
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 742b984c53e96e7d1465d9442af6c6606757e845
 > git branch -a -v --no-abbrev # timeout=10
 > git branch -D master # timeout=10
 > git checkout -b master 742b984c53e96e7d1465d9442af6c66067
 > git rev-list 742b984c53e96e7d1465d9442af6c6606757e845 # t
[Pipeline] sh
[workspace] Running shell script
+ docker build -t my-image:snapshot .
Sending build context to Docker daemon 289.8 kB

Step 1 : FROM java:8-jdk
 ---> 861e95c114d6
Step 2 : MAINTAINER B. Laster (bclaster@nclasters.org)
 ---> Using cache
 ---> 48b4694fbab0
Step 3 : ENV GRADLE_VERSION 2.14.1
 ---> Using cache
 ---> c84de3a28e12
Step 4 : RUN cd /opt   && wget https://services.gradle.org/di
bin.zip"   && ln -s "/opt/gradle-${GRADLE_VERSION}/bin/gradle
 ---> Using cache
 ---> df50ff638f0d
Step 5 : ENV GRADLE_HOME /opt/gradle
```



```
node() {
    def myImg
    stage ("Build image") {
        // download the dockerfile to build from
        git 'git@diyvb:repos/dockerResources.git'

        // build our docker image
         myImg = docker.build 'my-image:snapshot'
    }
    stage ("Get Source") {
```

Jenkins 2.0 and Docker – Inside example



Global variables are available in Pipeline directly, not as steps. They expose methods and variables to be accessed within your Pipeline script.

Global Variable Reference

Variables

docker

The docker variable offers convenient access to Docker-related functions from a Pipeline script.

Methods needing a slave will implicitly run a node {…} block if you have not wrapped them in one. It is a good idea to enclose a block of steps which should all run on the same node in su
localhost it likely will.)

Some methods return instances of auxiliary classes which serve as holders for an ID and which have their own methods and properties. Methods taking a body return any value returned b

`withRegistry(url[, credentialsId]) {…}`

Specifies a registry URL such as `https://docker.mycorp.com/`, plus an optional credentials ID to connect to it.

`withServer(uri[, credentialsId]) {…}`

Specifies a server URI such as `tcp://swarm.mycorp.com:2376`, plus an optional credentials ID to connect to it.

`withTool(toolName) {…}`

Specifies the name of a Docker installation to use, if any are defined in Jenkins global configuration. If unspecified, `docker` is assumed to be in the `$PATH` of the slave agent.

`image(id)`

Creates an `Image` object with a specified name or ID. See below.

`build(image[, args])`

Runs `docker build` to create and tag the specified image from a `Dockerfile` in the current directory. Additional `args` may be added, such as `'-f Dockerfile.other --pull`
object. Records a `FROM` fingerprint in the build.

`Image.id`

The image name with optional tag (`mycorp/myapp`, `mycorp/myapp:latest`) or ID (hexadecimal hash).

`Image.run([args, command])`

Uses `docker run` to run the image, and returns a `Container` which you could stop later. Additional `args` may be added, such as `'-p 8080:8080 --memory-swap=-1'`. Optiona

`Image.withRun[(args[, command])] {…}`

Like `run` but stops the container as soon as its body exits, so you do not need a `try-finally` block.

`Image.inside[(args)] {…}`

Like `withRun` this starts a container for the duration of the body, but all external commands (`sh`) launched by the body run inside the container rather than on the host. These comm

`Image.tag([tagname])`

Runs `docker tag` to record a tag of this image (defaulting to the tag it already has). Will rewrite an existing tag if one exists.

`Image.push([tagname])`

Pushes an image to the registry after tagging it as with the `tag` method. For example, you can use `image.push 'latest'` to publish it as the latest version in its repository.

`Image.pull()`

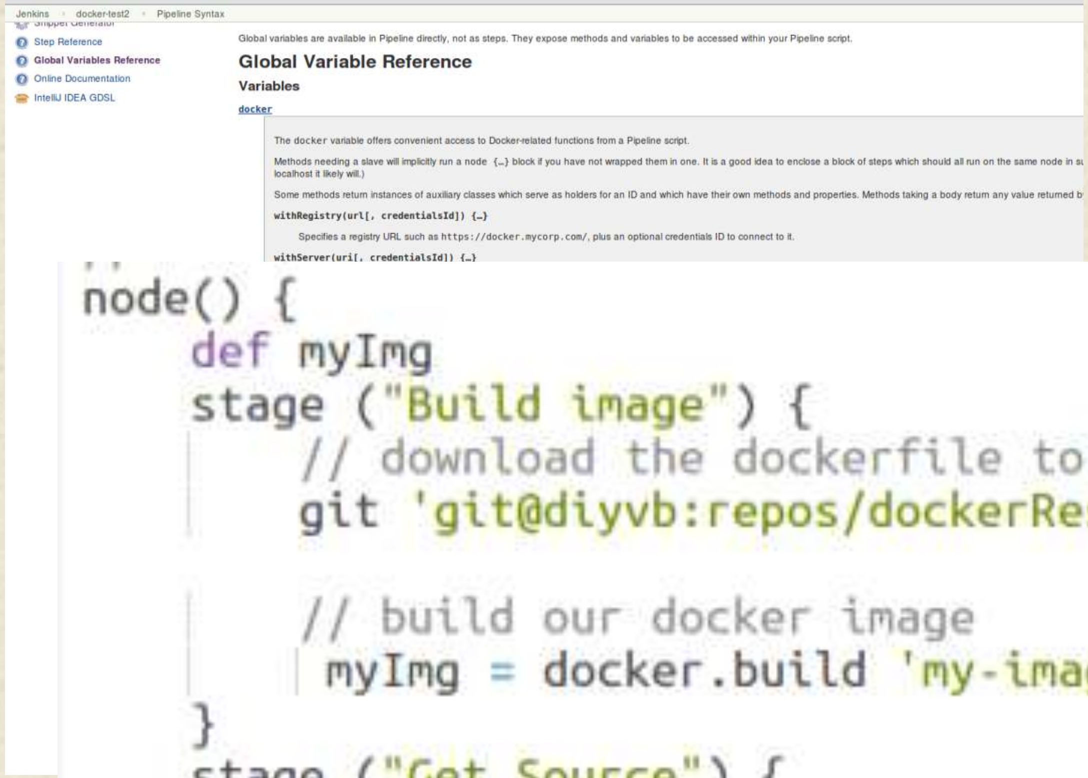
Runs `docker pull`. Not necessary before `run`, `withRun`, or `inside`.

`Image.imageName()`

The id prefixed as needed with registry information, such as `docker.mycorp.com/mycorp/myapp`. May be used if running your own Docker commands using `sh`.

```
=10
dockerResour
ockerResourc
ockerResourc
mit} # timeo
er^{commit}
f6c6606757e8
6606757e845
442af6c66067
6757e845 # t
s.org)
radle.org/di
}/bin/gradle
```

Jenkins 2.0 and Docker – Inside example



Jenkins › docker-test2 › Pipeline Syntax

- Snippet Generator
- Step Reference
- Global Variables Reference
- Online Documentation
- IntelliJ IDEA GDSL

Global variables are available in Pipeline directly, not as steps. They expose methods and variables to be accessed within your Pipeline script.

Global Variable Reference

Variables

docker

The docker variable offers convenient access to Docker-related functions from a Pipeline script.

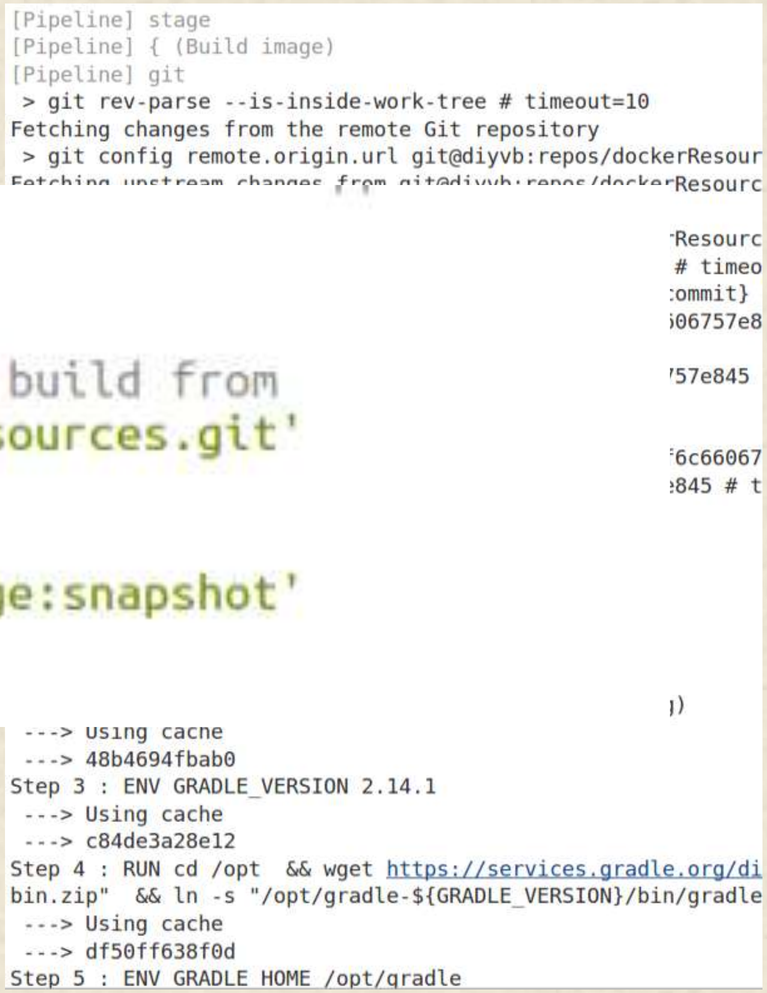
Methods needing a slave will implicitly run a node {…} block if you have not wrapped them in one. It is a good idea to enclose a block of steps which should all run on the same node in su
localhost it likely will.)

Some methods return instances of auxiliary classes which serve as holders for an ID and which have their own methods and properties. Methods taking a body return any value returned b

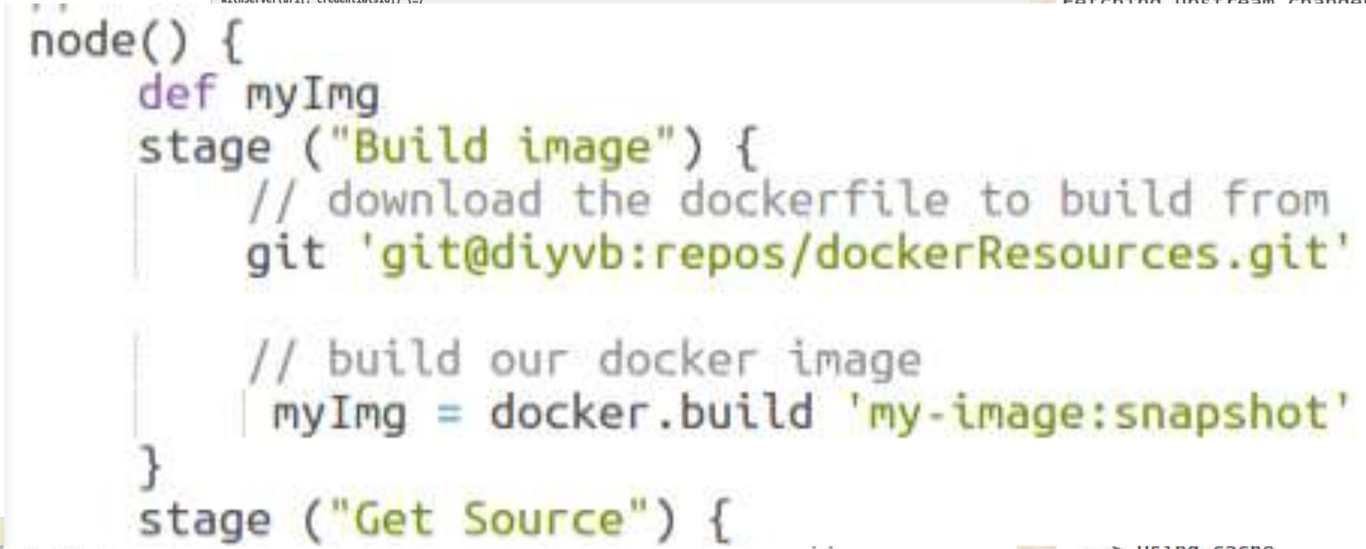
`withRegistry(url[, credentialsId]) {…}`

Specifies a registry URL such as `https://docker.mycorp.com/`, plus an optional credentials ID to connect to it.

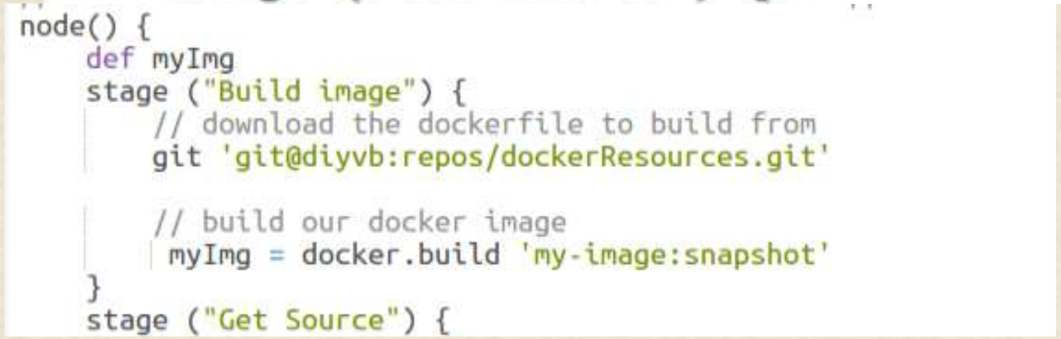
`withServer(uri[, credentialsId]) {…}`



```
[Pipeline] stage
[Pipeline] { (Build image)
[Pipeline] git
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url git@diyvb:repos/dockerResour
Fetching upstream changes from git@diyvb:repos/dockerResourc
                                                       Resourc
                                                       # timeo
                                                       ommit}
                                                       06757e8

                                                       57e845


                                                       6c66067
                                                       845 # t






                                                       g)
 ---> Using cache
 ---> 48b4694fbab0
Step 3 : ENV GRADLE_VERSION 2.14.1
 ---> Using cache
 ---> c84de3a28e12
Step 4 : RUN cd /opt   && wget https://services.gradle.org/di
bin.zip"   && ln -s "/opt/gradle-${GRADLE_VERSION}/bin/gradle
 ---> Using cache
 ---> df50ff638f0d
Step 5 : ENV GRADLE_HOME /opt/gradle
```

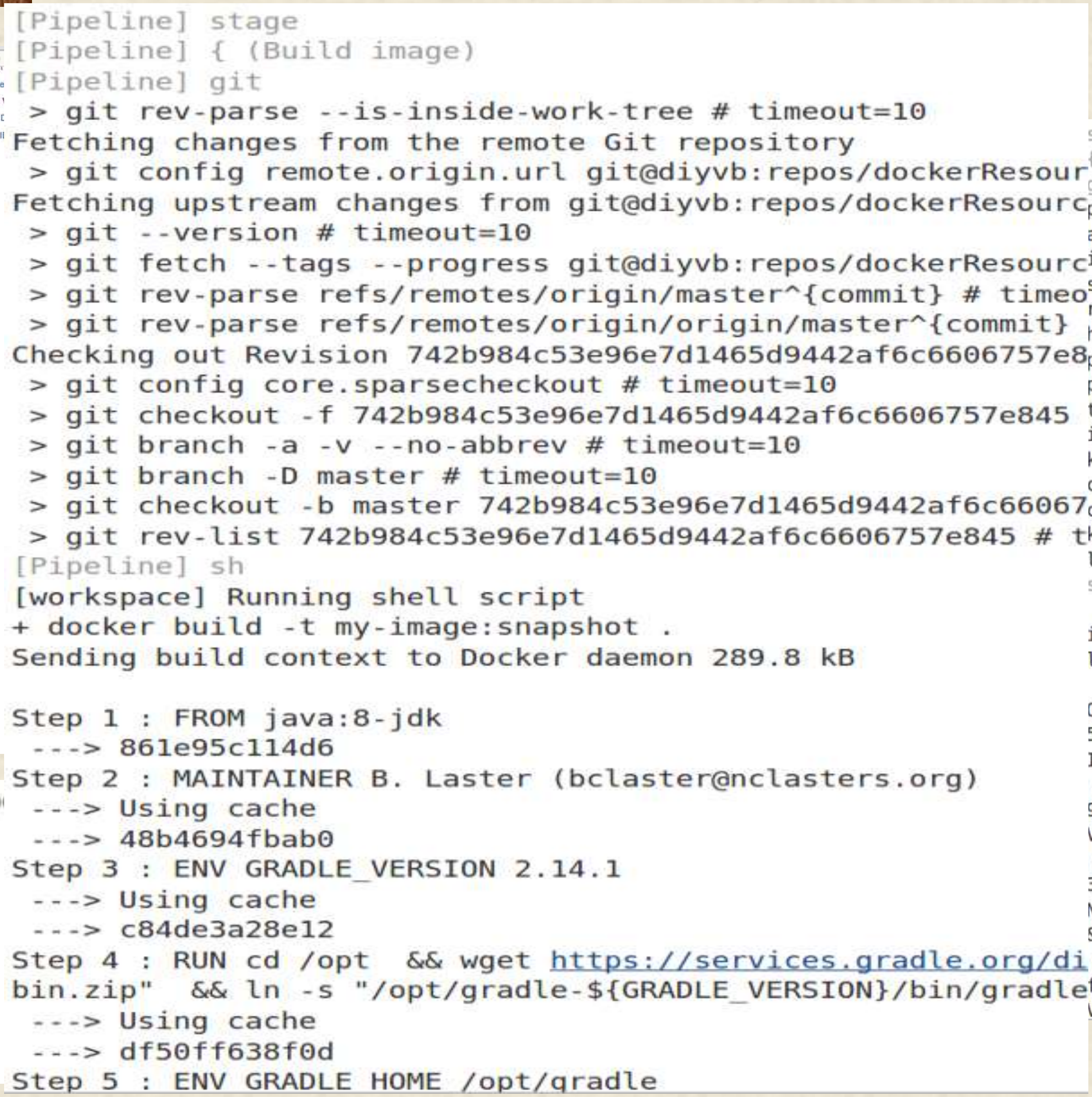


```
node() {
    def myImg
    stage ("Build image") {
        // download the dockerfile to build from
        git 'git@diyvb:repos/dockerResources.git'

        // build our docker image
         myImg = docker.build 'my-image:snapshot'
    }
    stage ("Get Source") {
```

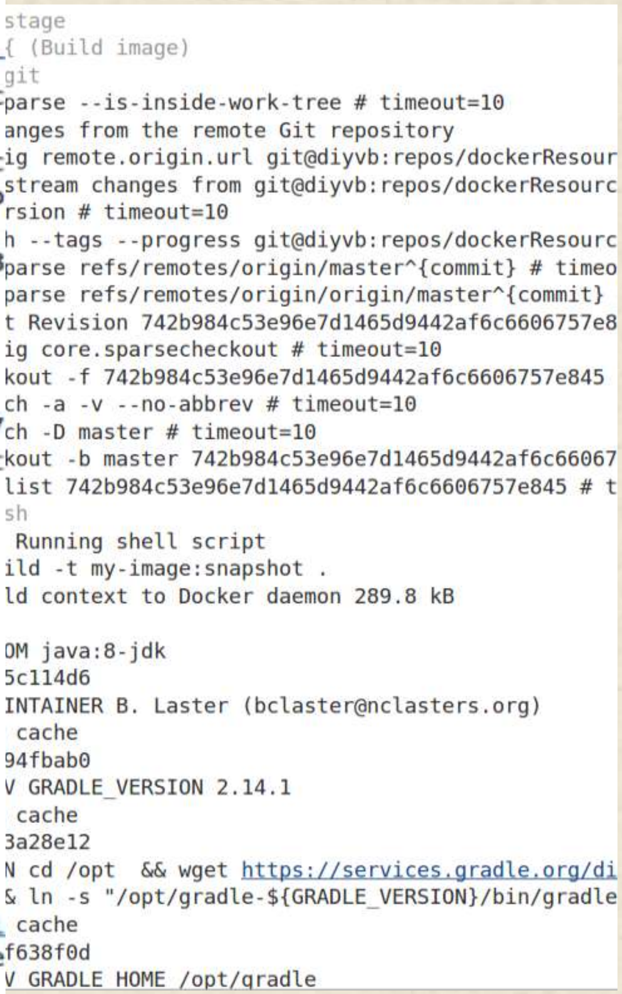


```
node() {
    def myImg
    stage ("Build image") {
        // download the dockerfile to build from
        git 'git@diyvb:repos/dockerResources.git'

        // build our docker image
         myImg = docker.build 'my-image:snapshot'
    }
    stage ("Get Source") {
```

Jenkins 2.0 and Docker – Inside example

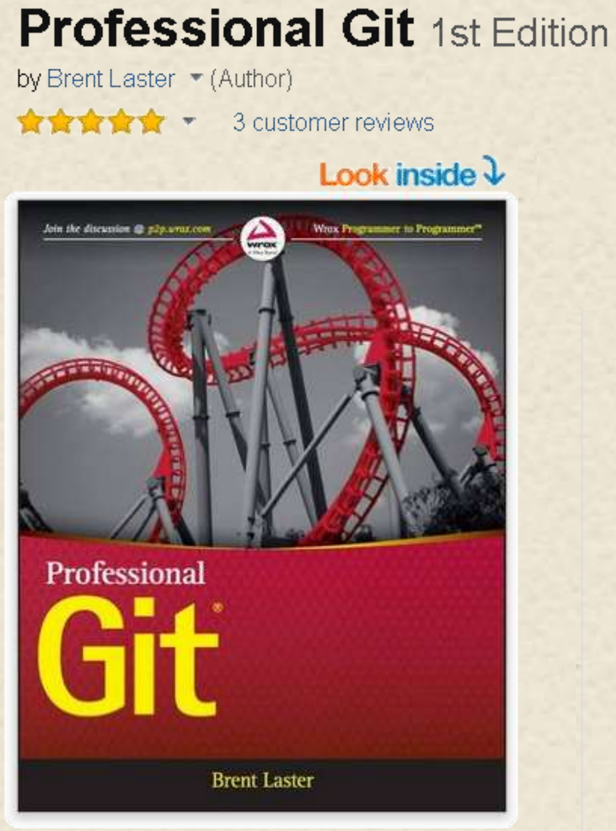


```
[Pipeline] stage
[Pipeline] { (Build image)
[Pipeline] git
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url git@diyvb:repos/dockerResour
Fetching upstream changes from git@diyvb:repos/dockerResourc
 > git --version # timeout=10
 > git fetch --tags --progress git@diyvb:repos/dockerResourc
 > git rev-parse refs/remotes/origin/master^{commit} # timeo
 > git rev-parse refs/remotes/origin/origin/master^{commit}
Checking out Revision 742b984c53e96e7d1465d9442af6c6606757e8
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 742b984c53e96e7d1465d9442af6c6606757e845
 > git branch -a -v --no-abbrev # timeout=10
 > git branch -D master # timeout=10
 > git checkout -b master 742b984c53e96e7d1465d9442af6c66067
 > git rev-list 742b984c53e96e7d1465d9442af6c6606757e845 # t
[Pipeline] sh
[workspace] Running shell script
+ docker build -t my-image:snapshot .
Sending build context to Docker daemon 289.8 kB

Step 1 : FROM java:8-jdk
 ---> 861e95c114d6
Step 2 : MAINTAINER B. Laster (bclaster@nclasters.org)
 ---> Using cache
 ---> 48b4694fbab0
Step 3 : ENV GRADLE_VERSION 2.14.1
 ---> Using cache
 ---> c84de3a28e12
Step 4 : RUN cd /opt   && wget https://services.gradle.org/di
bin.zip"   && ln -s "/opt/gradle-${GRADLE_VERSION}/bin/gradle
 ---> Using cache
 ---> df50ff638f0d
Step 5 : ENV GRADLE_HOME /opt/gradle
```



```
stage
{ (Build image)
git
parse --is-inside-work-tree # timeout=10
anges from the remote Git repository
ig remote.origin.url git@diyvb:repos/dockerResour
stream changes from git@diyvb:repos/dockerResourc
rsion # timeout=10
h --tags --progress git@diyvb:repos/dockerResourc
parse refs/remotes/origin/master^{commit} # timeo
parse refs/remotes/origin/origin/master^{commit}
t Revision 742b984c53e96e7d1465d9442af6c6606757e8
ig core.sparsecheckout # timeout=10
kout -f 742b984c53e96e7d1465d9442af6c6606757e845
ch -a -v --no-abbrev # timeout=10
ch -D master # timeout=10
kout -b master 742b984c53e96e7d1465d9442af6c66067
list 742b984c53e96e7d1465d9442af6c6606757e845 # t
sh
 Running shell script
ild -t my-image:snapshot .
ld context to Docker daemon 289.8 kB

OM java:8-jdk
5c114d6
INTAINER B. Laster (bclaster@nclasters.org)
 cache
94fbab0
V GRADLE_VERSION 2.14.1
 cache
3a28e12
N cd /opt   && wget https://services.gradle.org/di
& ln -s "/opt/gradle-${GRADLE_VERSION}/bin/gradle
 cache
f638f0d
V GRADLE_HOME /opt/gradle
```

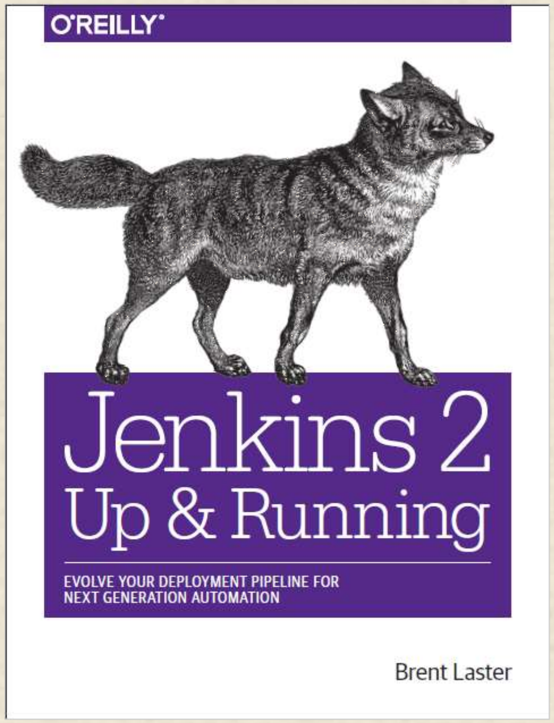
That's all - thanks!


Professional Git 1st Edition

by Brent Laster (Author)

★★★★★ 3 customer reviews

Look inside
Join the discussion @ p2p.wrox.com
Wrox Programmer to Programmer
Professional
Git
Brent Laster


O'REILLY
Jenkins 2
Up & Running
EVOLVE YOUR DEPLOYMENT PIPELINE FOR NEXT GENERATION AUTOMATION
Brent Laster