code
URJC

Tema 13

Integración Continua

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Universidad
Rey Juan Carlos

Integración Continua

code URJC

- El software se suele desarrollar en **equipos de desarrolladores**
- Los desarrolladores tienen que **compartir el código** que van desarrollando

Tipo de repositorio

GitLab

Herramientas / Servicios que ofrecen repositorios git

Repositorios Git

code URJC

- **GitHub** es un servicio para desarrolladores que proporciona repositorios git en sus servidores
  - Gratuitos para software libre
  - De pago para software privativo

https://github.com/

Repositorios Git

code URJC

- Para usar un repositorio Git es necesario usar un cliente
- Plugins de IDEs o Editores / Clientes interactivos

Repositorios Git

code URJC

- **Instalación de git para línea de comandos:**
  - **Windows:**
    - https://git-scm.com/download/win
    - Mantener valores por defecto
  - **Linux**
    - https://git-scm.com/download/linux
  - **MacOS**
    - https://git-scm.com/download/mac

Repositorios Git

code URJC

- **Comprobación de instalación correcta**
  - Windows:
    - Busca la aplicación Git bash y ábrela
    - Es una línea de comandos tipo unix
  - Linux y MacOS: abrir la línea de comandos
  - Comprobar que git está instalado y en el PATH:

```
git --version
```

Repositorios Git

code URJC

- Para poder **crear proyectos y añadir cambios** a un repositorio GitHub es necesario
  - Crearse una cuenta en GitHub
  - Configurar el cliente con las credenciales
  - Usar los clientes interactivos o por linea de comandos para subir los cambios
- Para poder descargar un proyecto de GitHub
  - Basta con ejecutar el comando:

```
git clone https://github.com/codeurjc/testing.git
```

Integración Continua

code URJC

- Cada vez que un desarrollador sube un cambio al repositorio, se debería comprobar que
  - El proyecto que hay en el repositorio **sigue compilando**
  - Todos los **tests del proyecto siguen pasando** (en verde)
- Para compilar y ejecutar los tests se usa un servidor con un software de **Integración Continua**
- Si hay algún problema con el código, el **problema se notifica** inmediatamente a los desarrolladores

Integración Continua

code URJC

Jenkins

GitLab

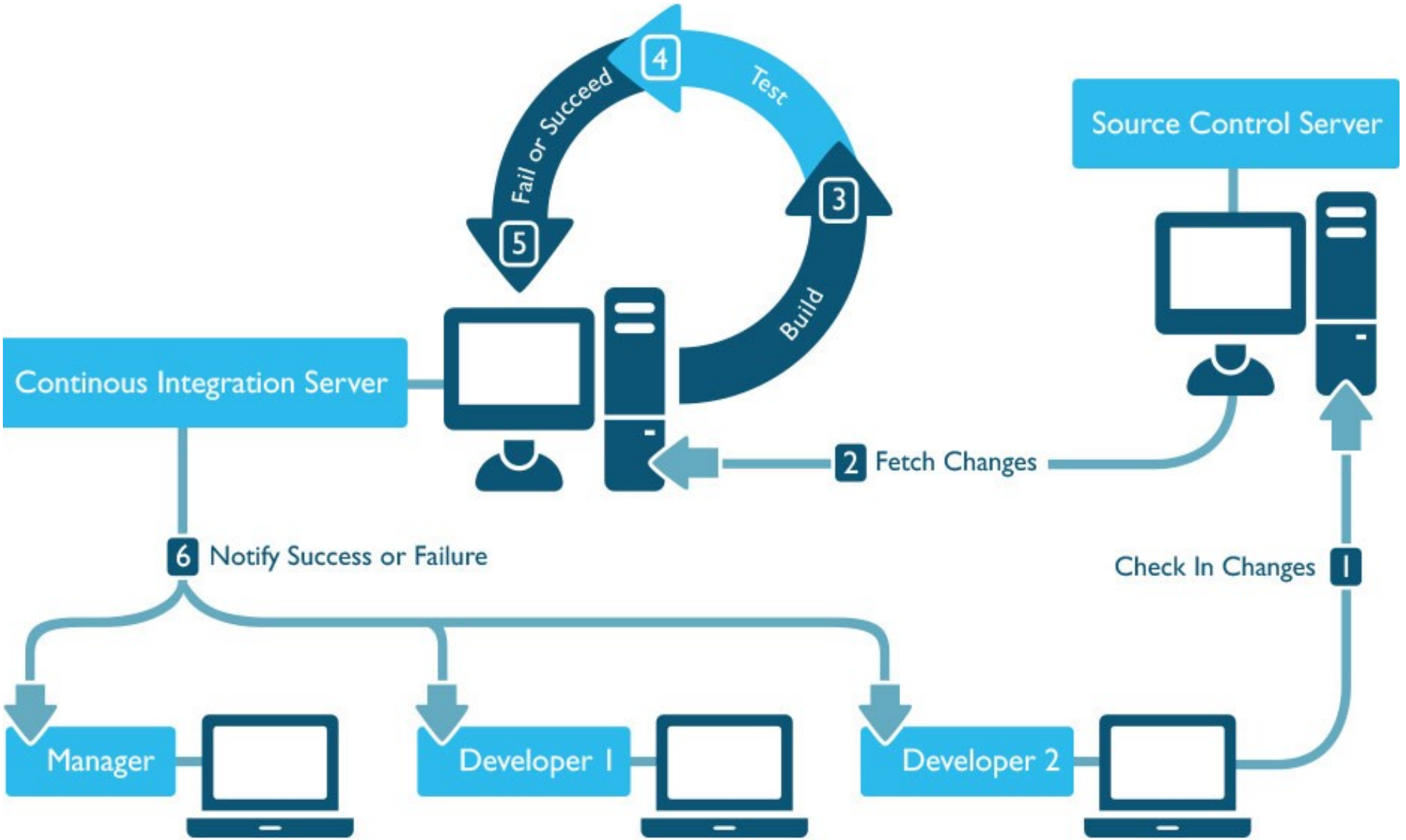
Travis CI

Bamboo

Servidores y servicios de Integración Continua

Integración Continua

code URJC



Integración Continua

code URJC

Jenkins

ENABLE AUTO REFRESH

New Job
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
My Views

add description

All | Software | +

S	W	Name ↓	Last Success	Last Failure	Last Duration
		build-chebi	5 days 22 hr (#6)	9 days 14 hr (#5)	4 min 49 sec
		build-cl	3 days 1 hr (#12)	19 days (#10)	1 min 55 sec
		build-fbbt	5 days 22 hr (#7)	15 days (#6)	2 min 2 sec
		build-fypo	3 days 9 hr (#7)	7 days 5 hr (#5)	2 min 22 sec
		build-qaz	1 day 0 hr (#3)	1 day 0 hr (#2)	26 min
		build-go	4 hr 55 min (#178)	N/A	6 min 48 sec
		build-go-taxon	1 mo 3 days (#106)	44 min (#243)	3 min 40 sec
		build-go-xp-chebi	4 hr 48 min (#115)	N/A	2 min 21 sec
		build-mp	N/A	14 hr (#68)	4 min 1 sec

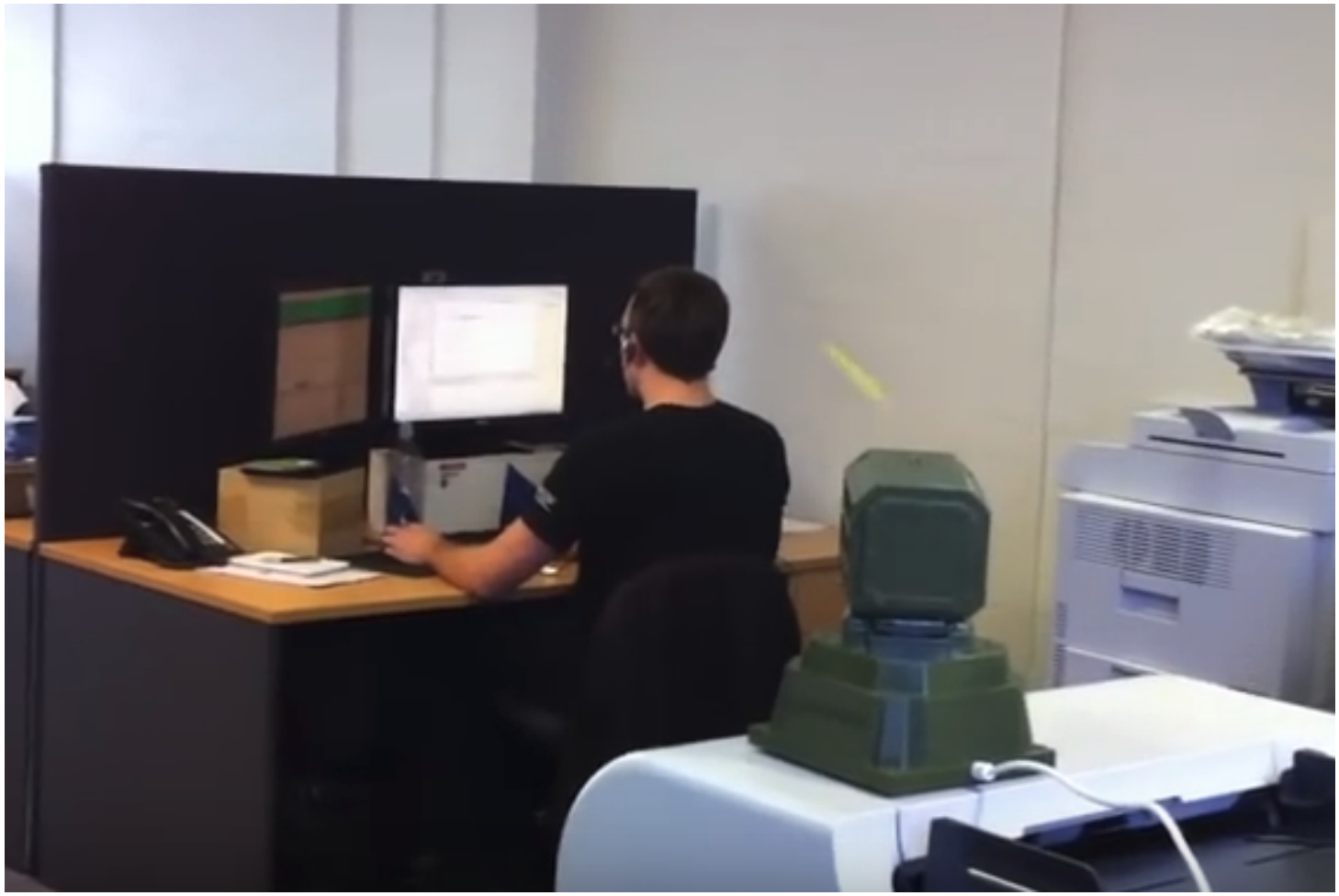
Build Queue

No builds in the queue.

Build Executor Status

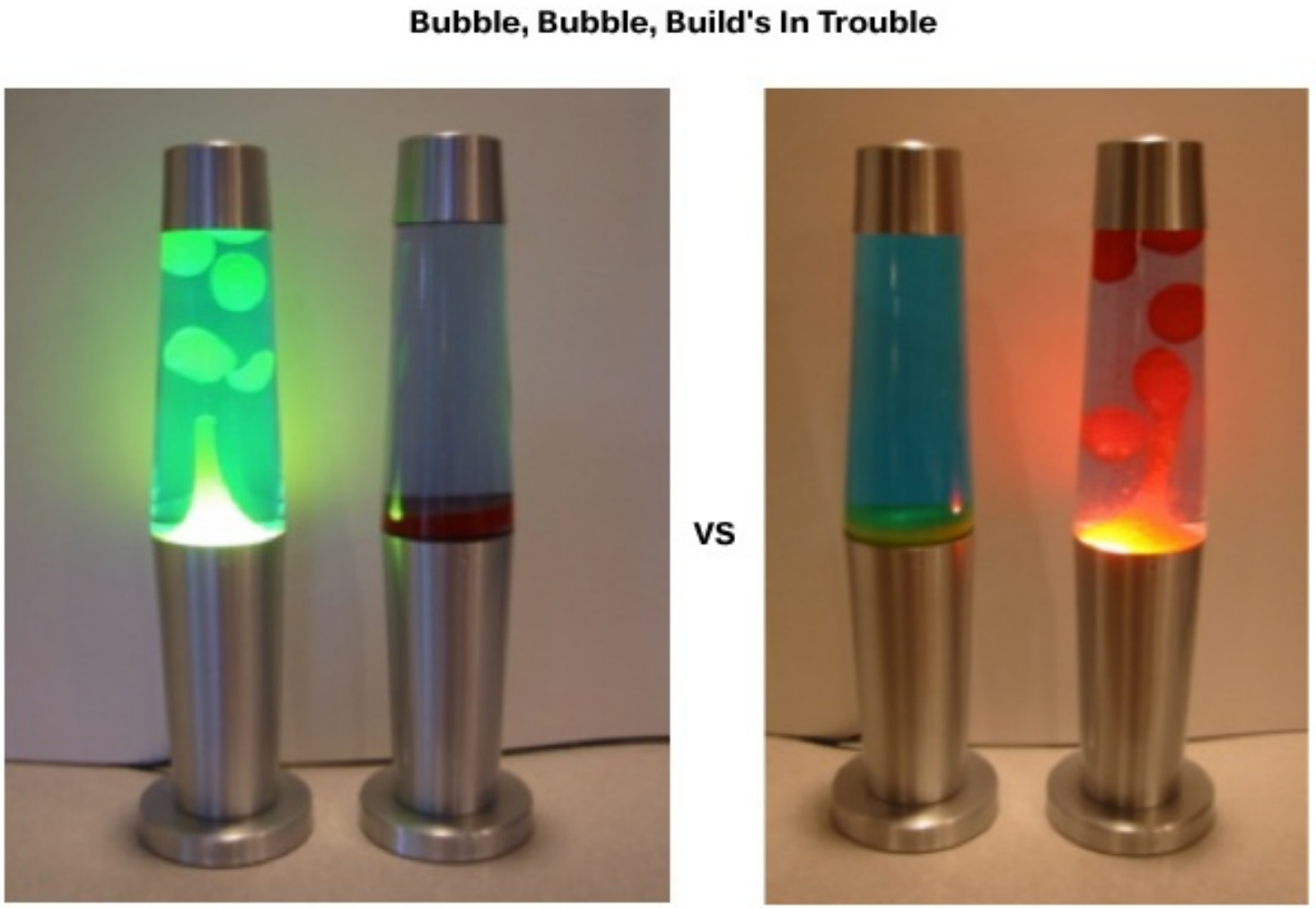
#	Status
1	Idle
2	Idle

Integración Continua



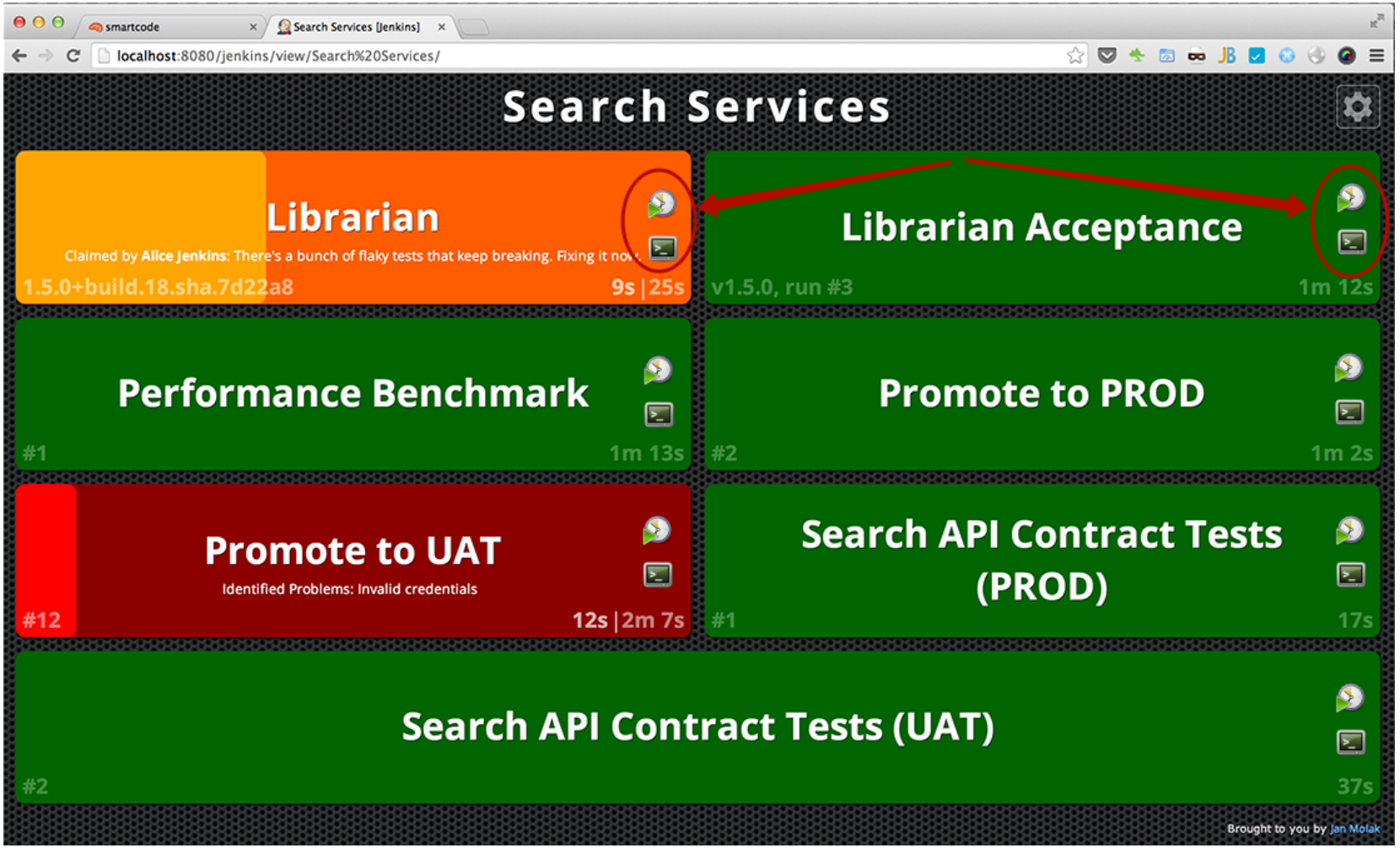
https://www.youtube.com/watch?v=1EGk2rvZe8A

Integración Continua



Integración Continua

code URJC



Integración Continua

code URJC

- El **proceso de integración** (subida al repositorio) tiene que realizarse de forma continua (**al menos una vez al día**)
- Eso permite tener una **realimentación rápida** de que el código que desarrolla cada programador se **integra bien con los demás**
- Cualquier problema detectado por CI debe **repararse de forma inmediata** antes de continuar con el desarrollo
- En cada momento tenemos una **visión general del estado del proyecto** software desarrollado

Jenkins

- Jenkins es un servidor diseñado para ejecutar tareas (Jobs)
- Como se usa en desarrollo software, estas tareas suelen **construir software**, ejecutar **tests** y algunas veces **desplegar** ese software

Jenkins

https://jenkins.io/

Jenkins

- **Instalación rápida**
  - Necesitamos Java 8 o superior
  - Vamos a https://jenkins.io/
  - Seleccionamos download
  - Descargamos la LTS Release (Generic Java package)

Jenkins

- **Instalación rápida**
  - Ejecutamos

```
$ java -jar jenkins.war --httpPort=8081
```

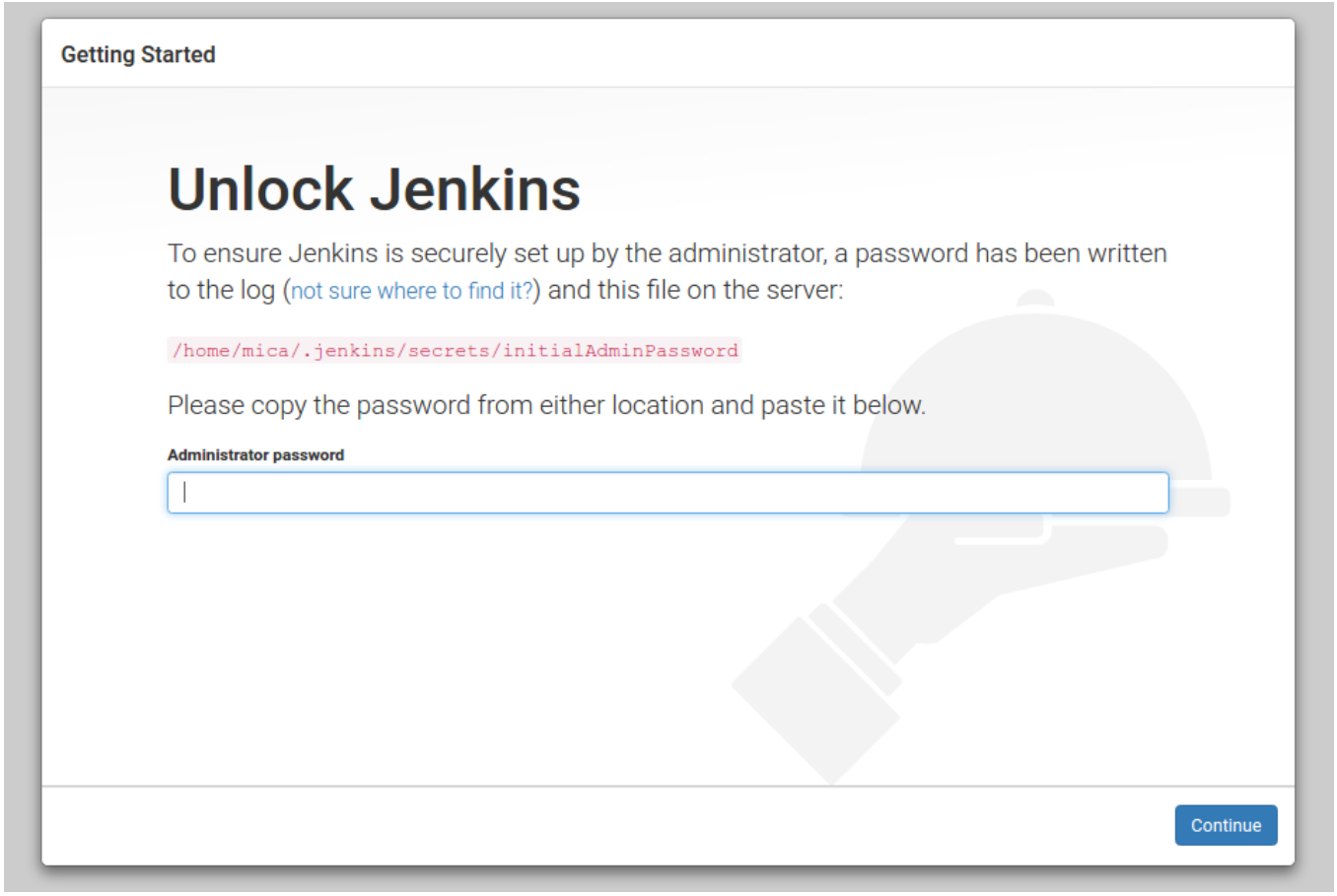
  - La password de admin aparece en el log o en el fichero

```
~/.jenkins/secrets/initialAdminPassword
```

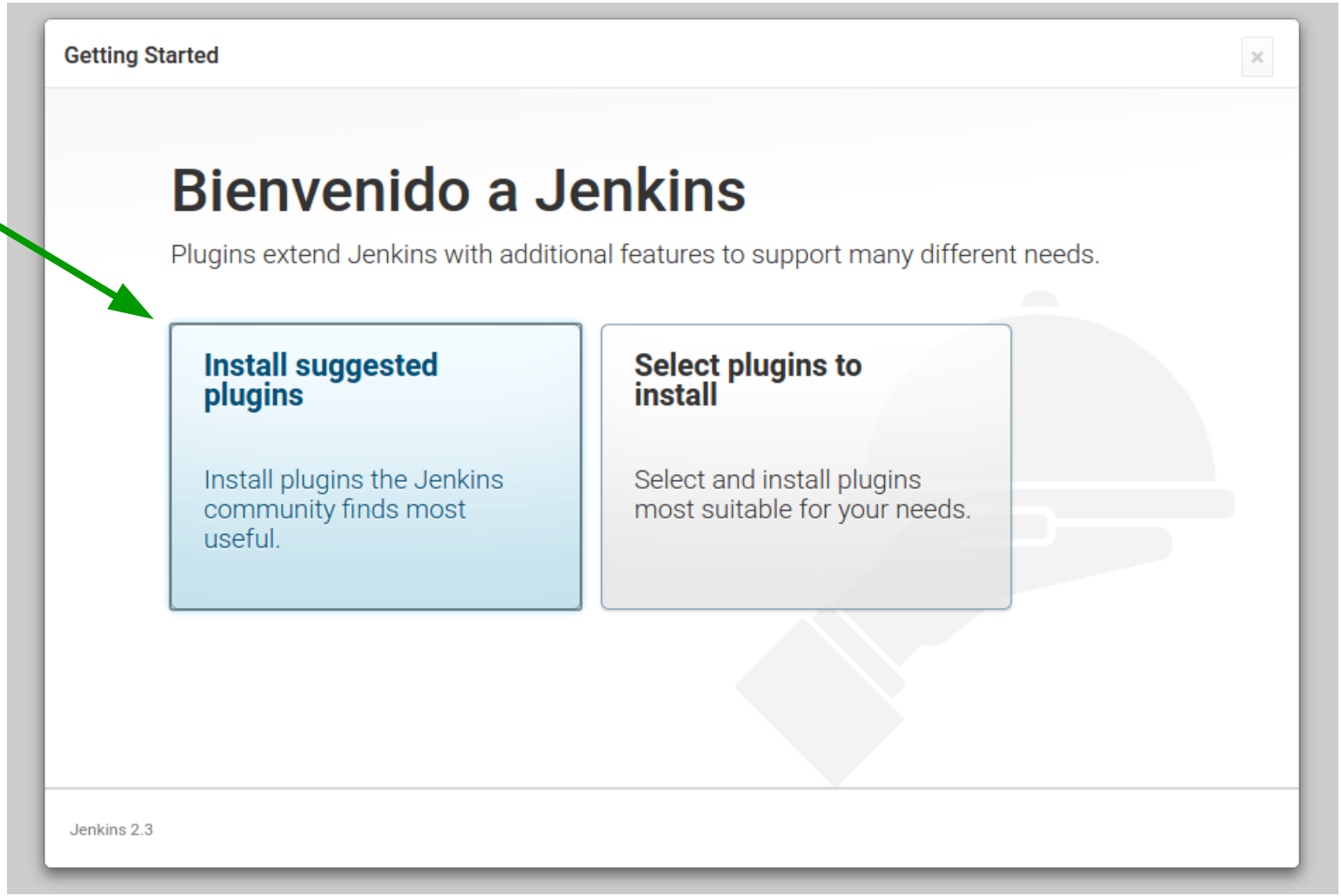
  - **Nota**: Borrar la carpeta ~/.jenkins si ya teníamos instalado Jenkins de antes y queremos empezar de cero

Jenkins

- **Instalación rápida**
  - Accedemos a http://localhost:8081/

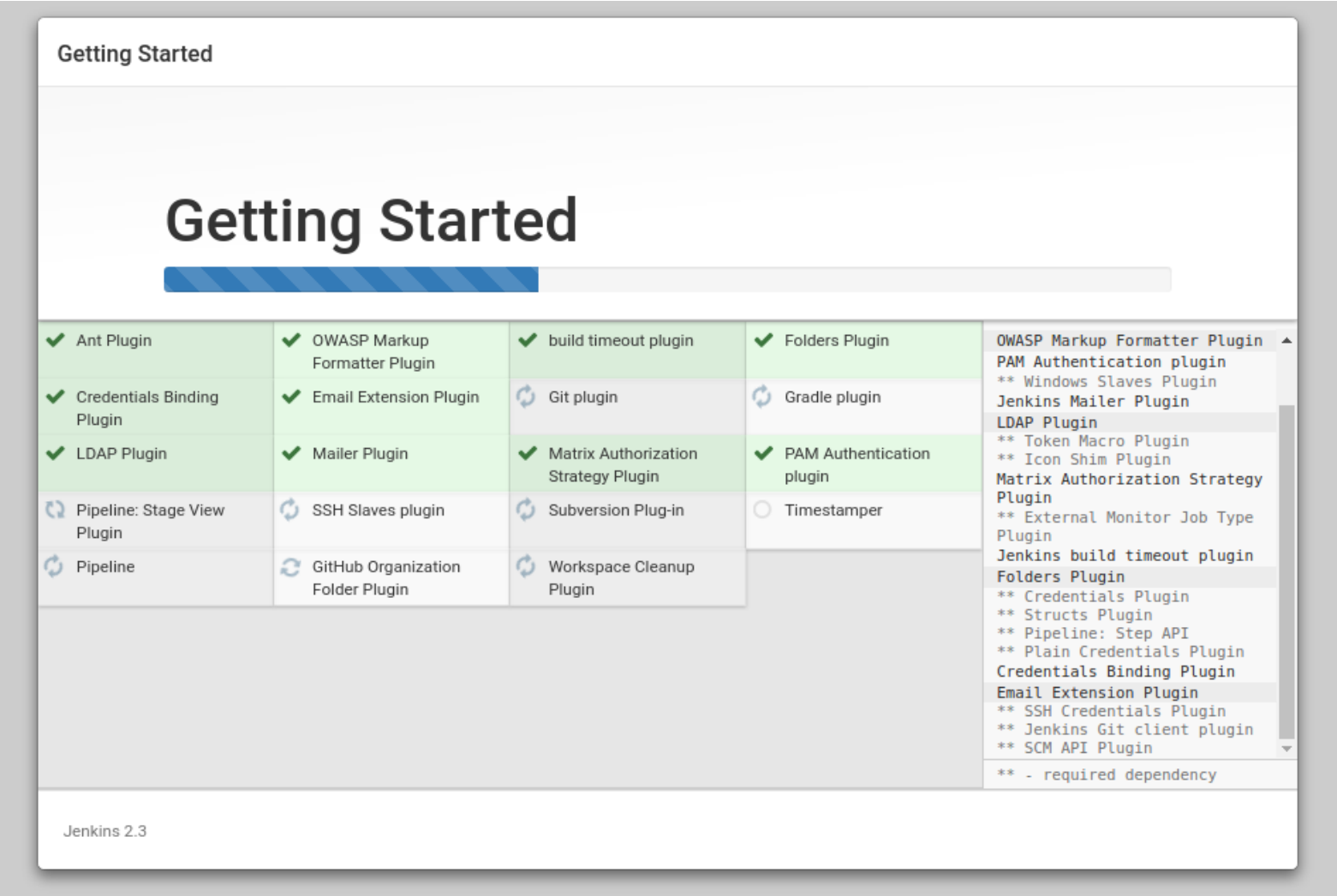


Jenkins

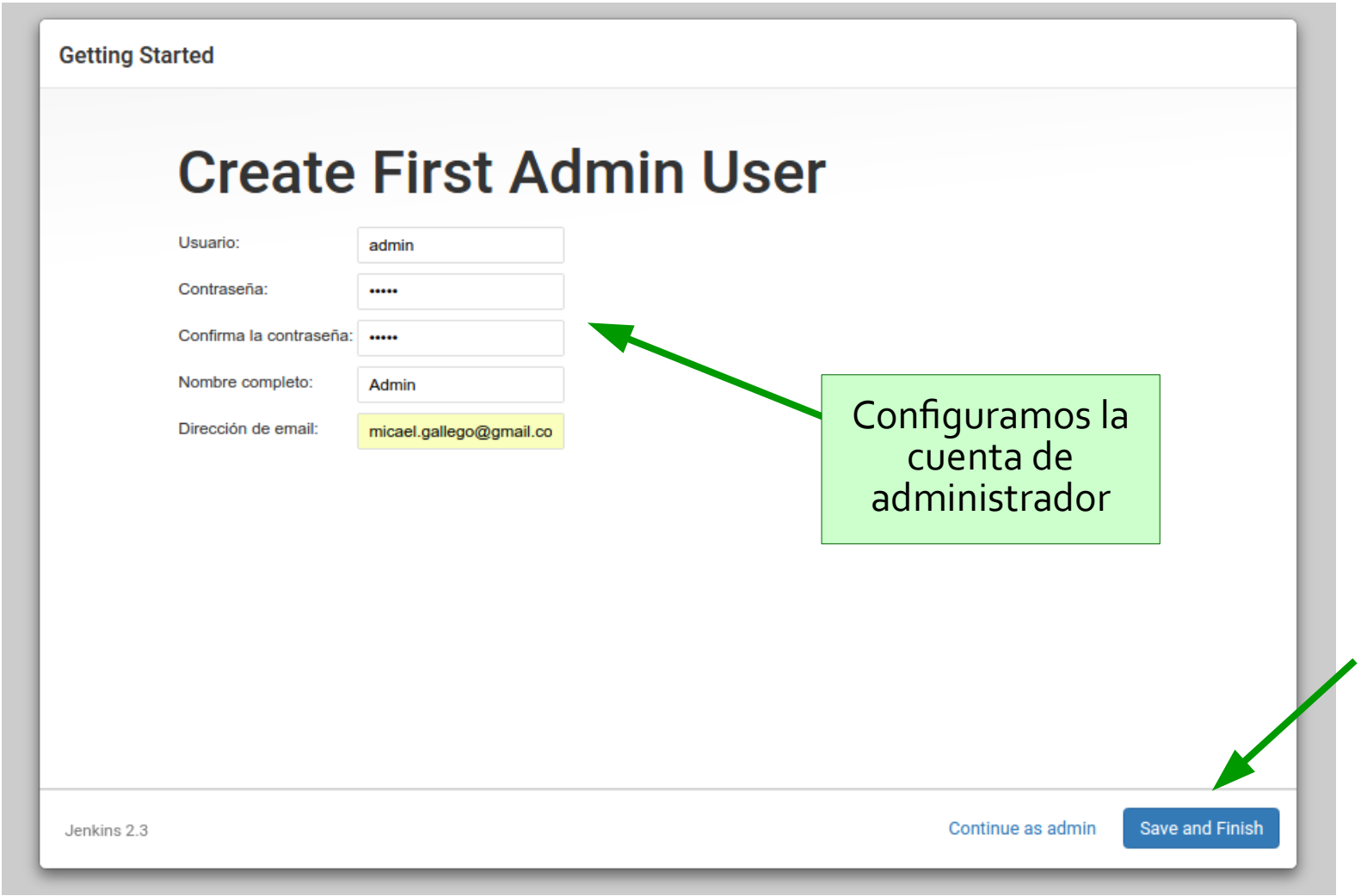

Getting Started
Bienvenido a Jenkins
Plugins extend Jenkins with additional features to support many different needs.
Install suggested plugins
Install plugins the Jenkins community finds most useful.
Select plugins to install
Select and install plugins most suitable for your needs.
Jenkins 2.3

Jenkins

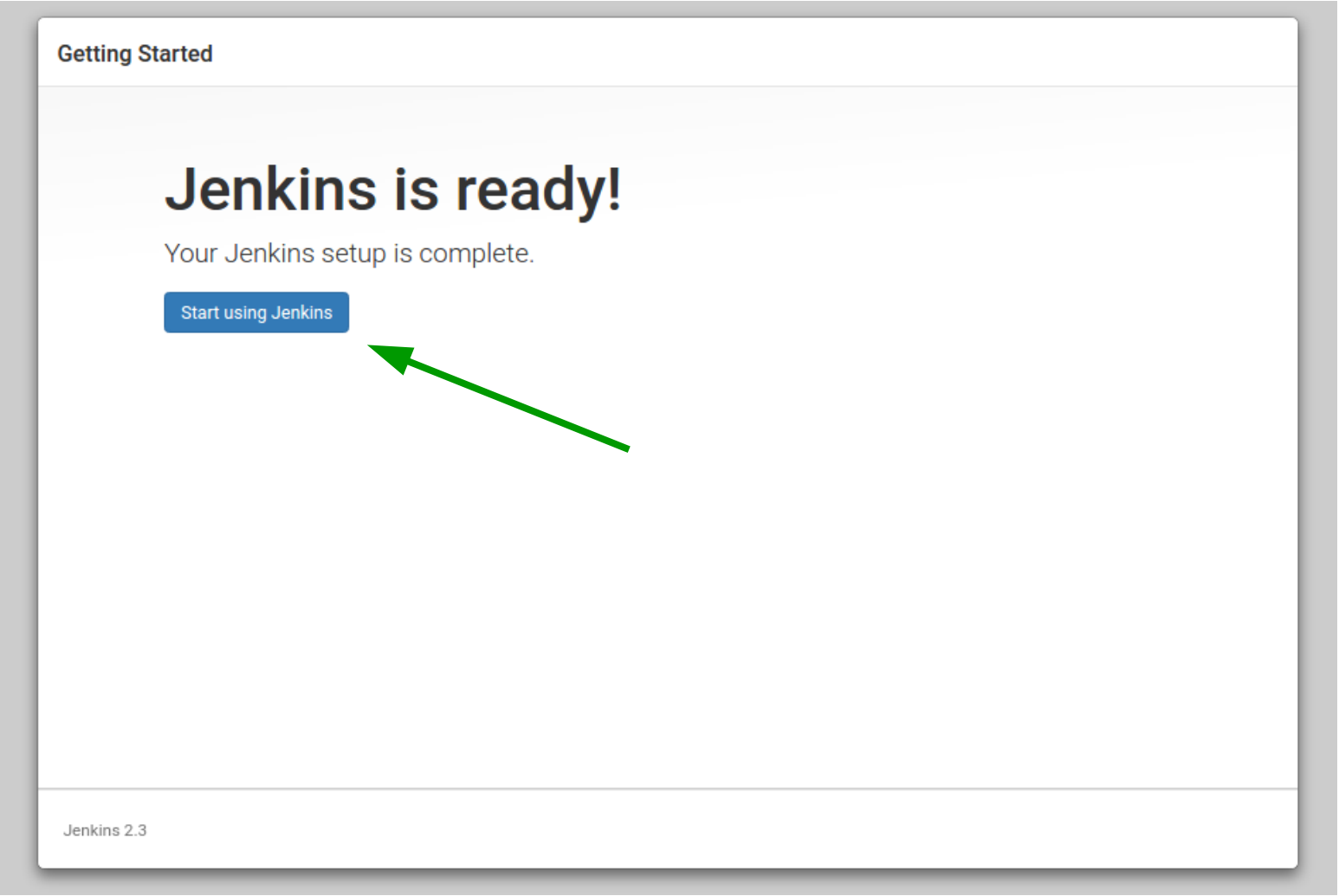
code URJC



Jenkins


Getting Started
Create First Admin User
Usuario:
admin
Contraseña:
Confirma la contraseña:
Nombre completo:
Admin
Dirección de email:
micael.gallego@gmail.co
Configuramos la cuenta de administrador
Jenkins 2.3
Continue as admin
Save and Finish

Jenkins


Getting Started
Jenkins is ready!
Your Jenkins setup is complete.
Start using Jenkins
Jenkins 2.3

Jenkins

code URJC

- **Creación de tareas (jobs)**
  - El objetivo de jenkins es ejecutar **tareas**
  - Una tarea es la **ejecución de un comando**
  - Esos comandos pueden **terminar correctamente** o con **error**
  - La **salida del comando** se puede **consultar** mientras se ejecuta y se **guarda** para una consulta posterior
  - Los comandos pueden generar **ficheros** que también se **guardan**

Jenkins

code URJC

- **Creación de tareas (jobs)**
  - Las tareas se pueden ejecutar:
    - Iniciadas por el desarrollador (**manualmente**)
    - Cada cierto tiempo (**todas las noches**)
    - Motivadas por un evento externo (**nuevo cambio en el repositorio de código**)
  - Nosotros las vamos a ejecutar manualmente porque no vamos a subir cambios a los repositorios

Jenkins

code URJC

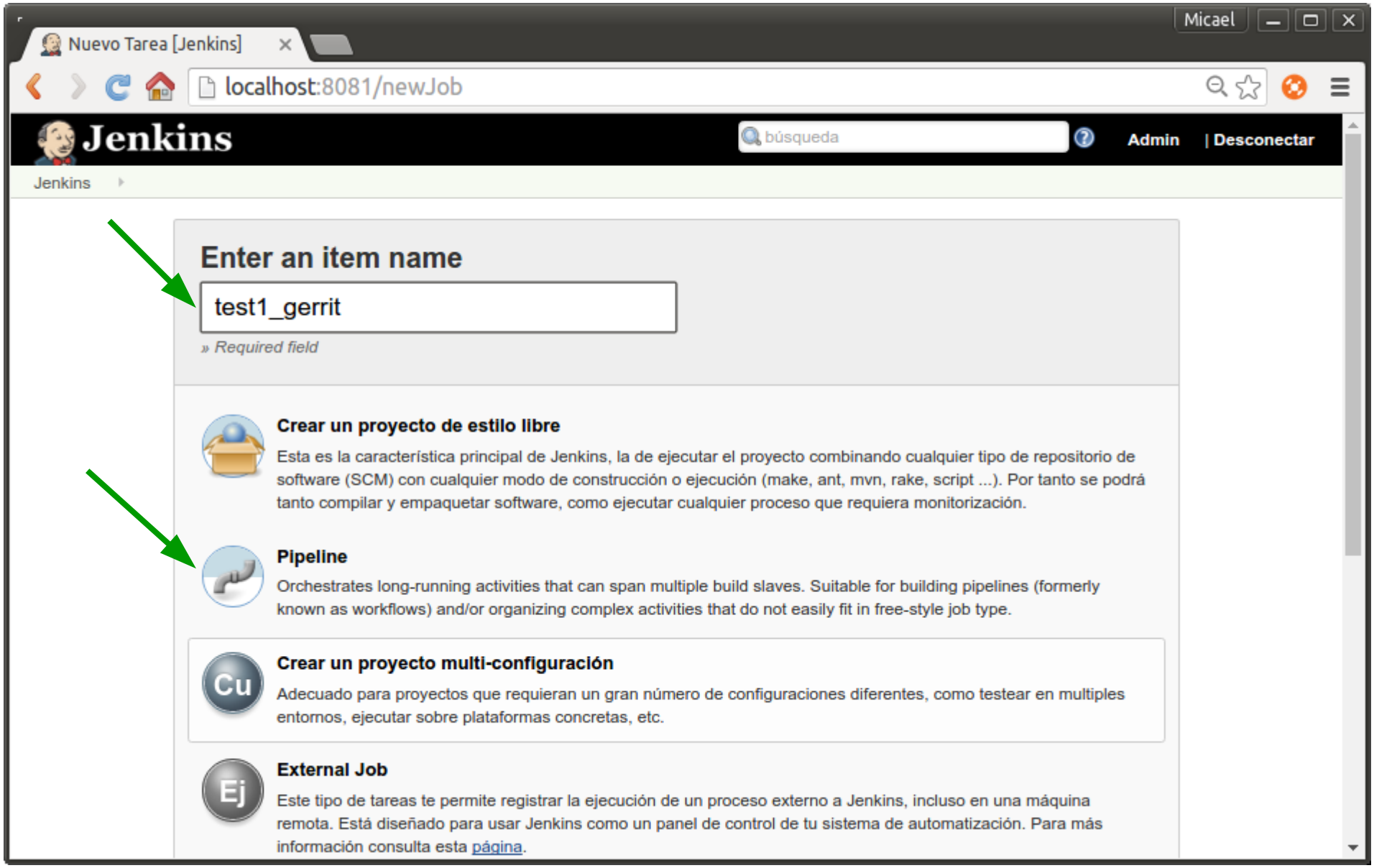
- **Creación de tareas (jobs)**
  - Una tarea típica en Jenkins consiste en:
    - 1) Descargar un **proyecto software** de un repositorio **git**
    - 2) **Compilar** el proyecto
    - 3) Ejecutar los **tests** sobre el proyecto
    - 4) Opcionalmente **generar un binario/paquete** y publicarlo en algún repositorio de binarios

Jenkins

Creamos una nueva tarea para descargar el proyecto y ejecutar los tests

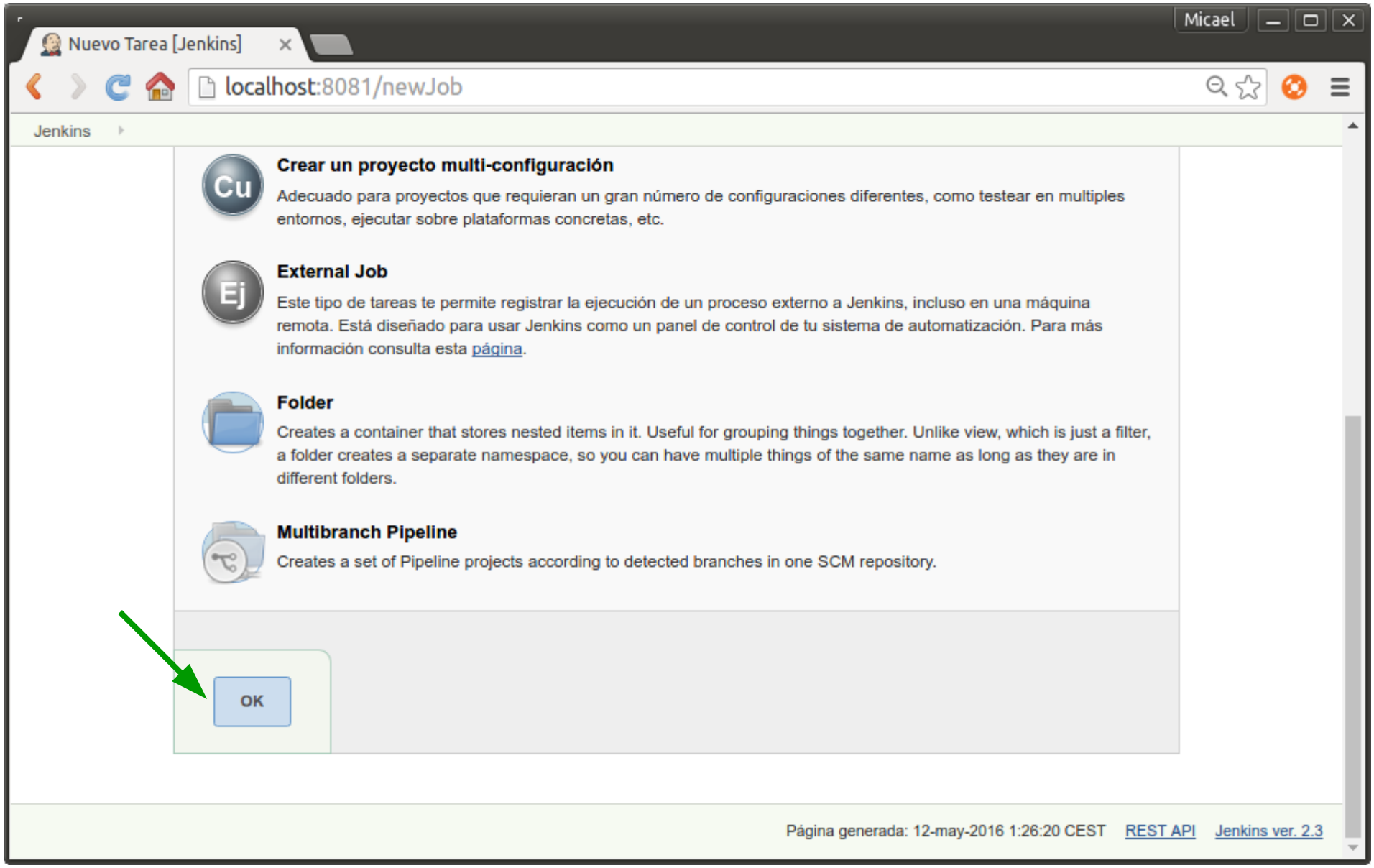
Jenkins

code URJC

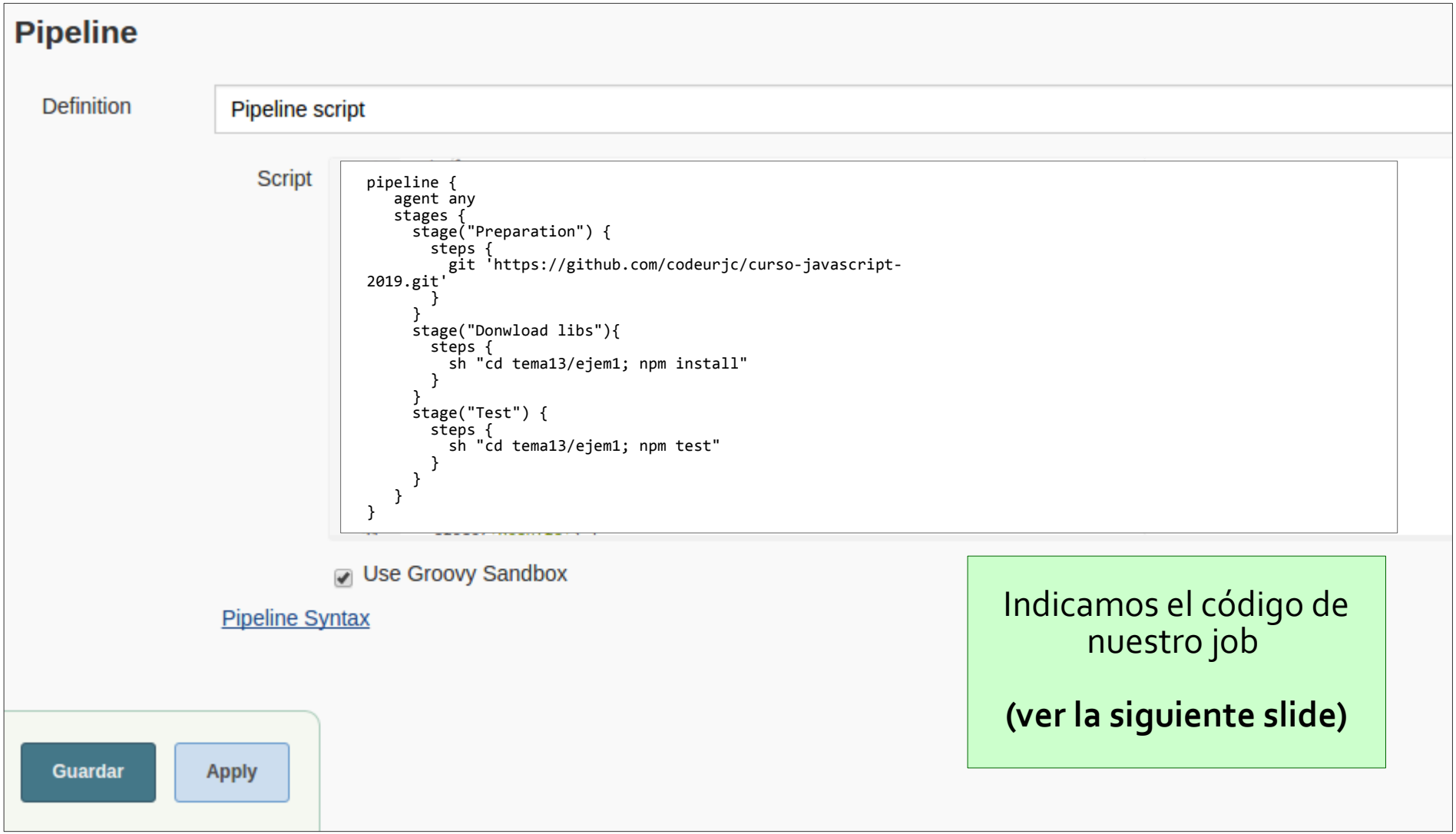


Jenkins

code URJC



Jenkins



Indicamos el código de nuestro job

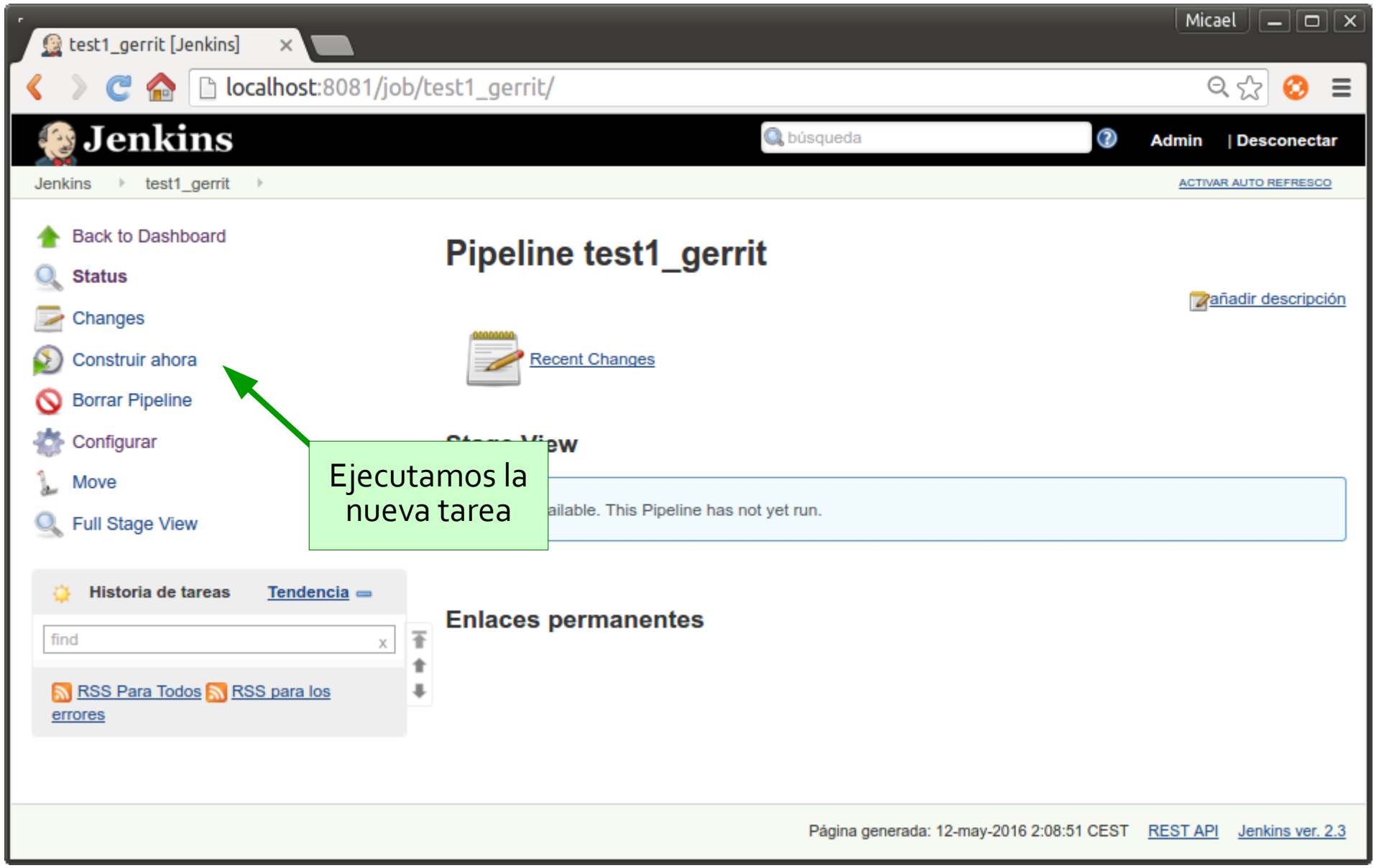
(ver la siguiente slide)

Jenkins

code URJC

```
pipeline {
    agent any
    stages {
      stage("Preparation") {
        steps {
          git 'https://github.com/codeurjc/curso-javascript-2019.git'
        }
      }
      stage("Donwload libs"){
        steps {
          sh "cd tema13/ejem1; npm install"
        }
      }
      stage("Test") {
        steps {
          sh "cd tema13/ejem1; npm test"
        }
      }
    }
}
```

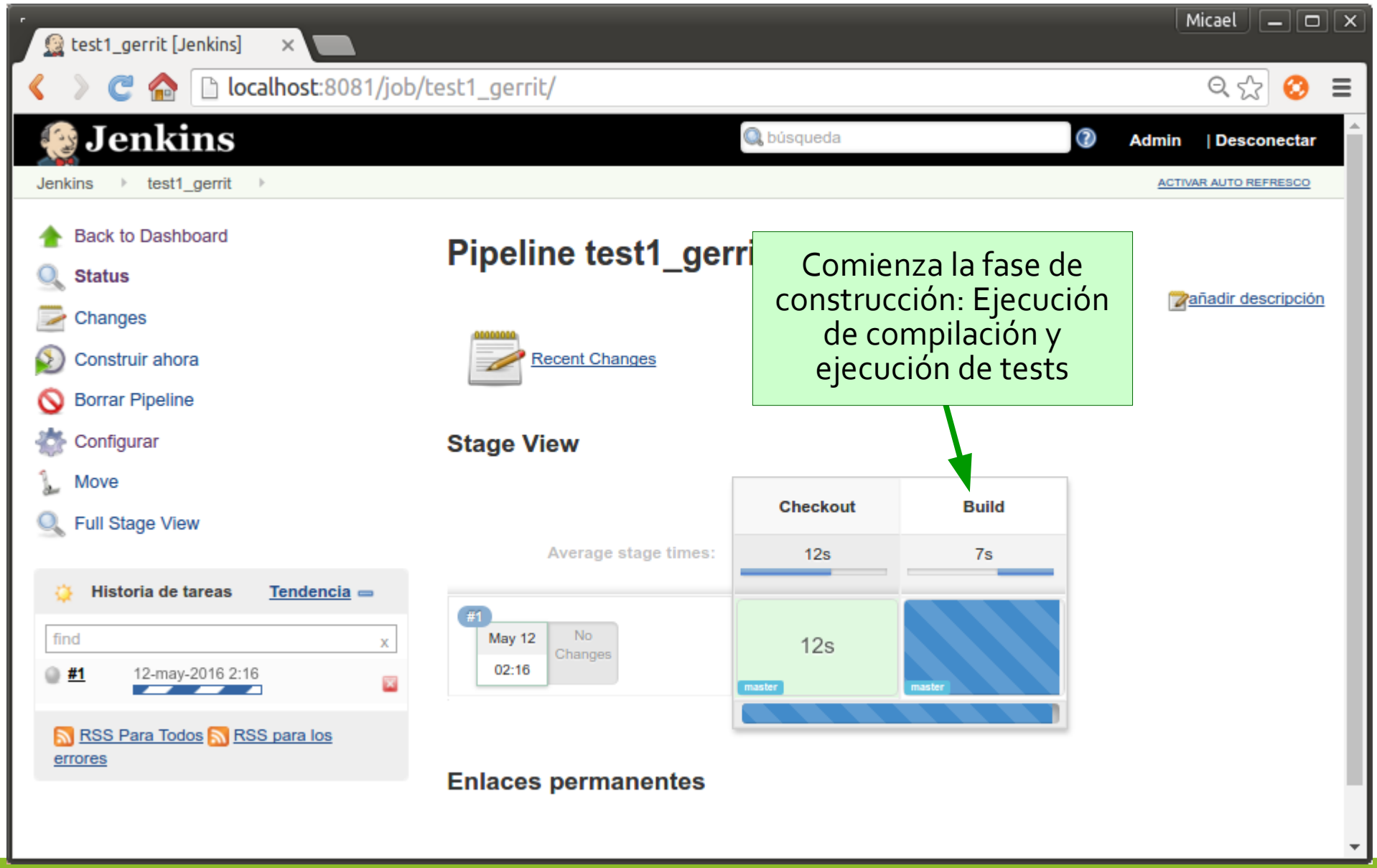
Jenkins


test1_gerrit [Jenkins]
Micael
localhost:8081/job/test1_gerrit/
Jenkins
búsqueda
Admin
| Desconectar
Jenkins
test1_gerrit
ACTIVAR AUTO REFRESCO
Back to Dashboard
Status
Changes
Construir ahora
Borrar Pipeline
Configurar
Move
Full Stage View
Pipeline test1_gerrit
añadir descripción
Recent Changes
ew
ailable. This Pipeline has not yet run.
Ejecutamos la nueva tarea
Historia de tareas
Tendencia
find
RSS Para Todos
RSS para los errores
Enlaces permanentes
Página generada: 12-may-2016 2:08:51 CEST
REST API
Jenkins ver. 2.3

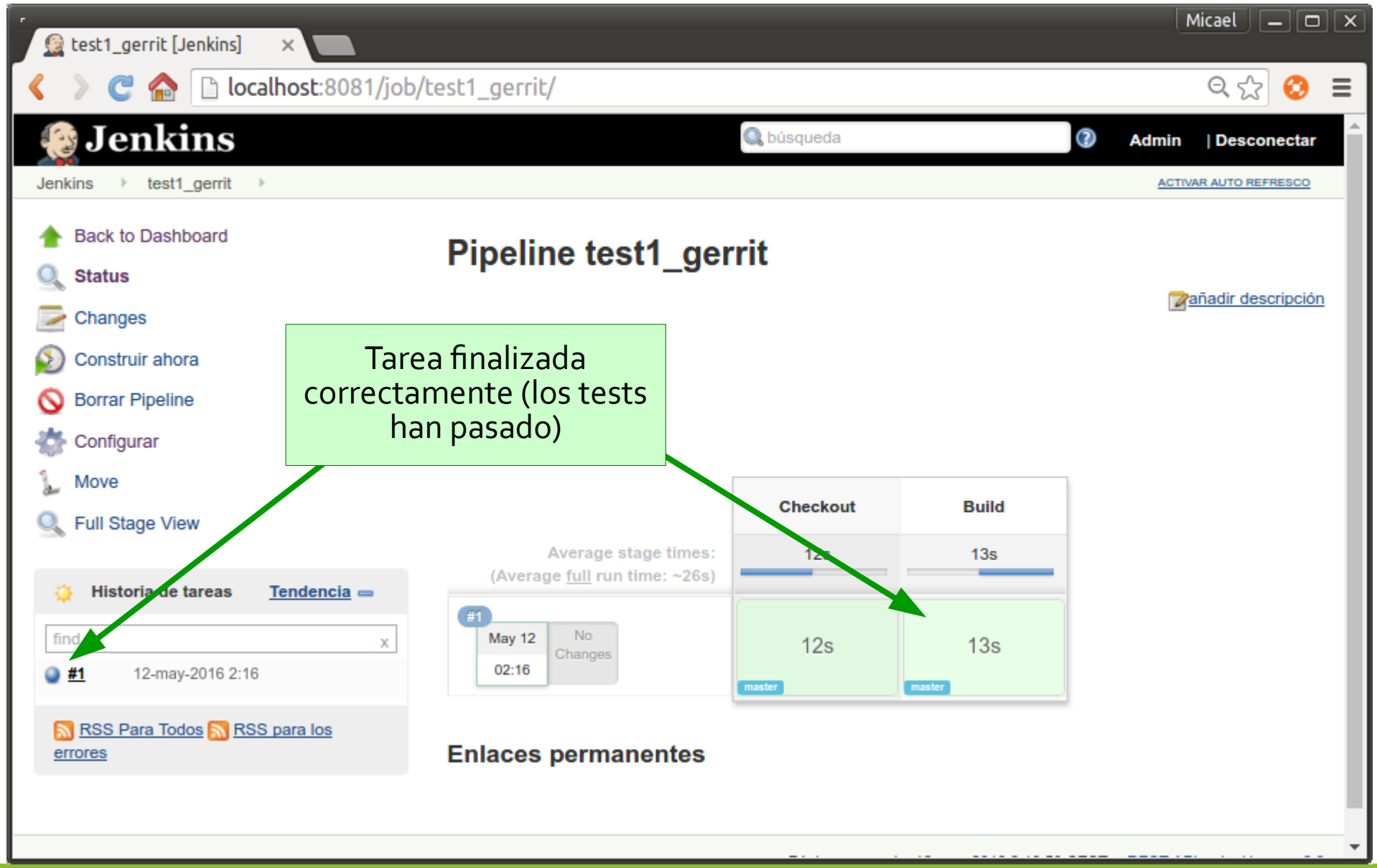
Jenkins


test1_gerrit [Jenkins]
Micael
localhost:8081/job/test1_gerrit/
Jenkins
búsqueda
Admin
| Desconectar
Jenkins
test1_gerrit
ACTIVAR AUTO REFRESCO
Back to Dashboard
Status
Changes
Construir ahora
Borrar Pipeline
Configurar
Move
Full Stage View
Historia de tareas
Tendencia
find
#1
12-may-2016 2:16
RSS Para Todos
RSS para los errores
Pipeline test1_gerrit
añadir descripción
Recent Changes
Comienza la fase de descarga del repositorio
Stage View
Checkout
Average stage times:
5s
#1
May 12
02:16
No Changes
master
Enlaces permanentes

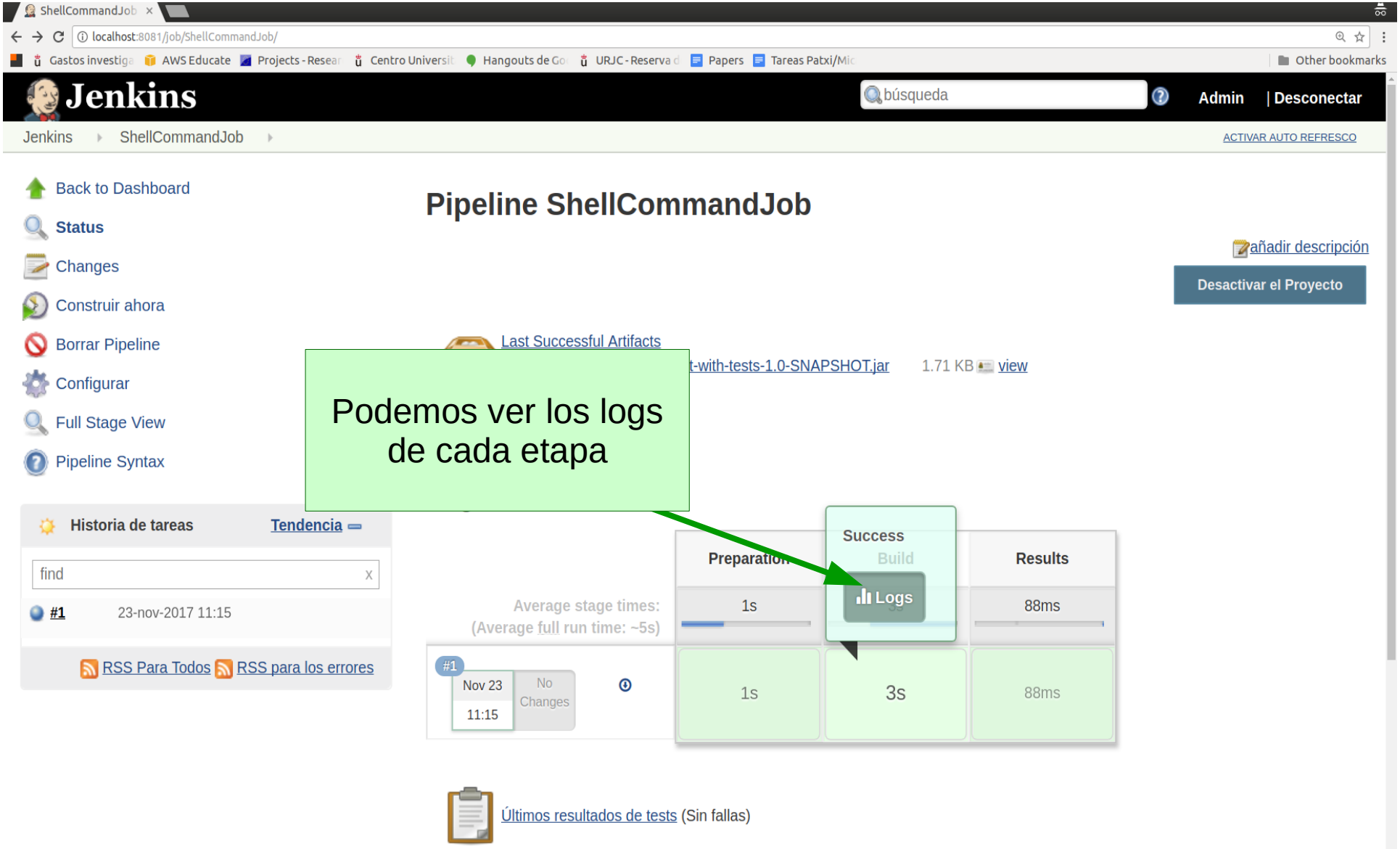
Jenkins


test1_gerrit [Jenkins]
Micael
localhost:8081/job/test1_gerrit/
Jenkins
búsqueda
Admin
| Desconectar
Jenkins
test1_gerrit
ACTIVAR AUTO REFRESCO
Back to Dashboard
Status
Changes
Construir ahora
Borrar Pipeline
Configurar
Move
Full Stage View
Historia de tareas
Tendencia
find
#1
12-may-2016 2:16
RSS Para Todos
RSS para los errores
Pipeline test1_gerri
añadir descripción
Recent Changes
Comienza la fase de construcción: Ejecución de compilación y ejecución de tests
Stage View
Checkout
Build
Average stage times:
12s
7s
#1
May 12
02:16
No Changes
12s
master
master
Enlaces permanentes

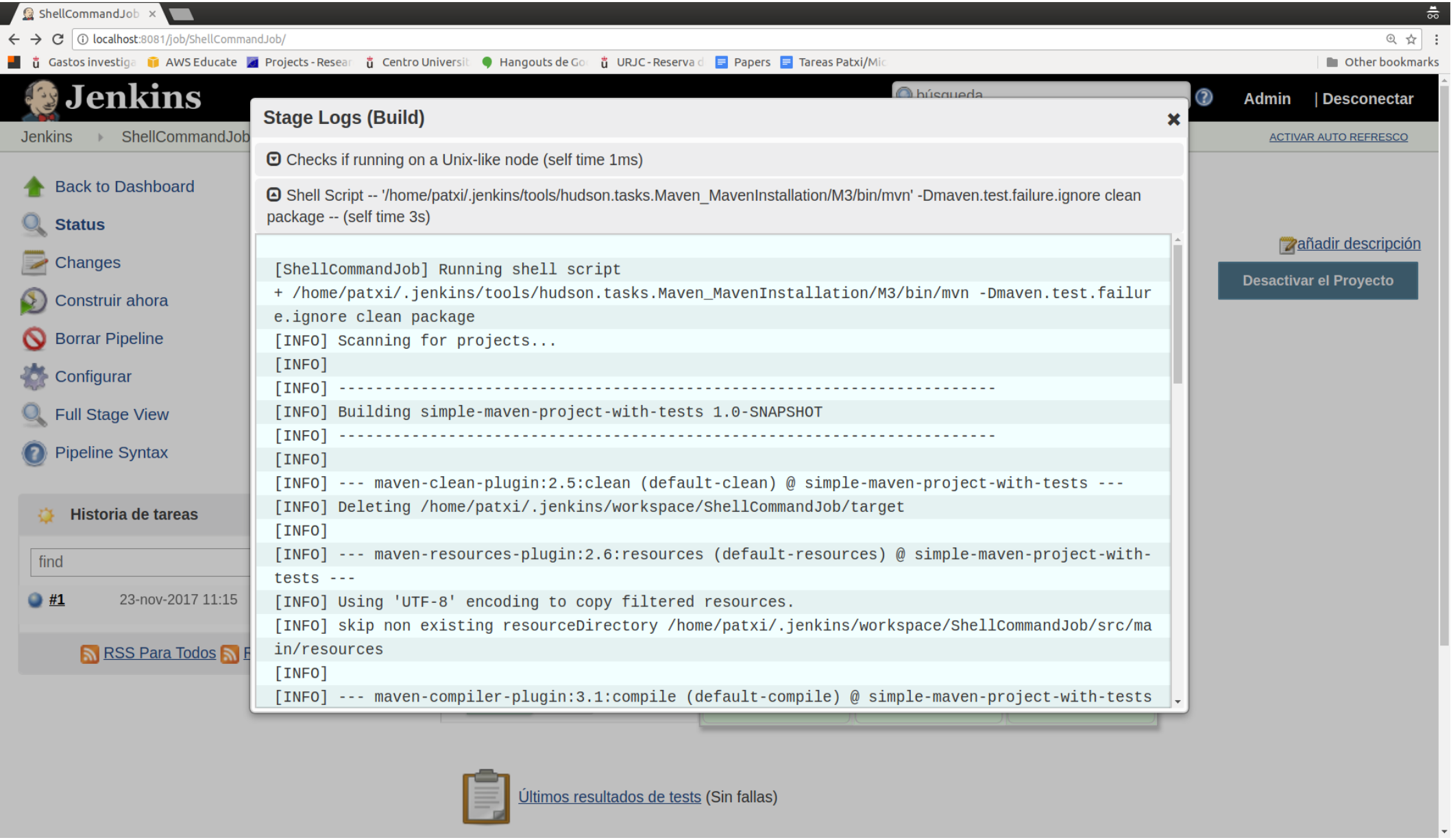
Jenkins


test1_gerrit [Jenkins]
Micael
localhost:8081/job/test1_gerrit/
Jenkins
búsqueda
Admin
| Desconectar
Jenkins
test1_gerrit
ACTIVAR AUTO REFRESCO
Back to Dashboard
Status
Changes
Construir ahora
Borrar Pipeline
Configurar
Move
Full Stage View
Pipeline test1_gerrit
añadir descripción
Tarea finalizada correctamente (los tests han pasado)
Historia de tareas
Tendencia
#1
12-may-2016 2:16
RSS Para Todos
RSS para los errores
Checkout
Build
Average stage times:
(Average full run time: ~26s)
13s
#1
May 12
02:16
No Changes
12s
13s
master
master
Enlaces permanentes

Jenkins


ShellCommandJob
localhost:8081/job/ShellCommandJob/
Jenkins
búsqueda
Admin
| Desconectar
Jenkins
ShellCommandJob
ACTIVAR AUTO REFRESCO
Back to Dashboard
Status
Changes
Construir ahora
Borrar Pipeline
Configurar
Full Stage View
Pipeline Syntax
Pipeline ShellCommandJob
añadir descripción
Desactivar el Proyecto
Last Successful Artifacts
t-with-tests-1.0-SNAPSHOT.jar 1.71 KB view
Podemos ver los logs de cada etapa
Historia de tareas
Tendencia
find
#1 23-nov-2017 11:15
RSS Para Todos
RSS para los errores
Success
Build
Logs
Preparation
Results
Average stage times:
(Average full run time: ~5s)
1s
88ms
#1
Nov 23
11:15
No Changes
1s
3s
88ms
Últimos resultados de tests (Sin fallas)

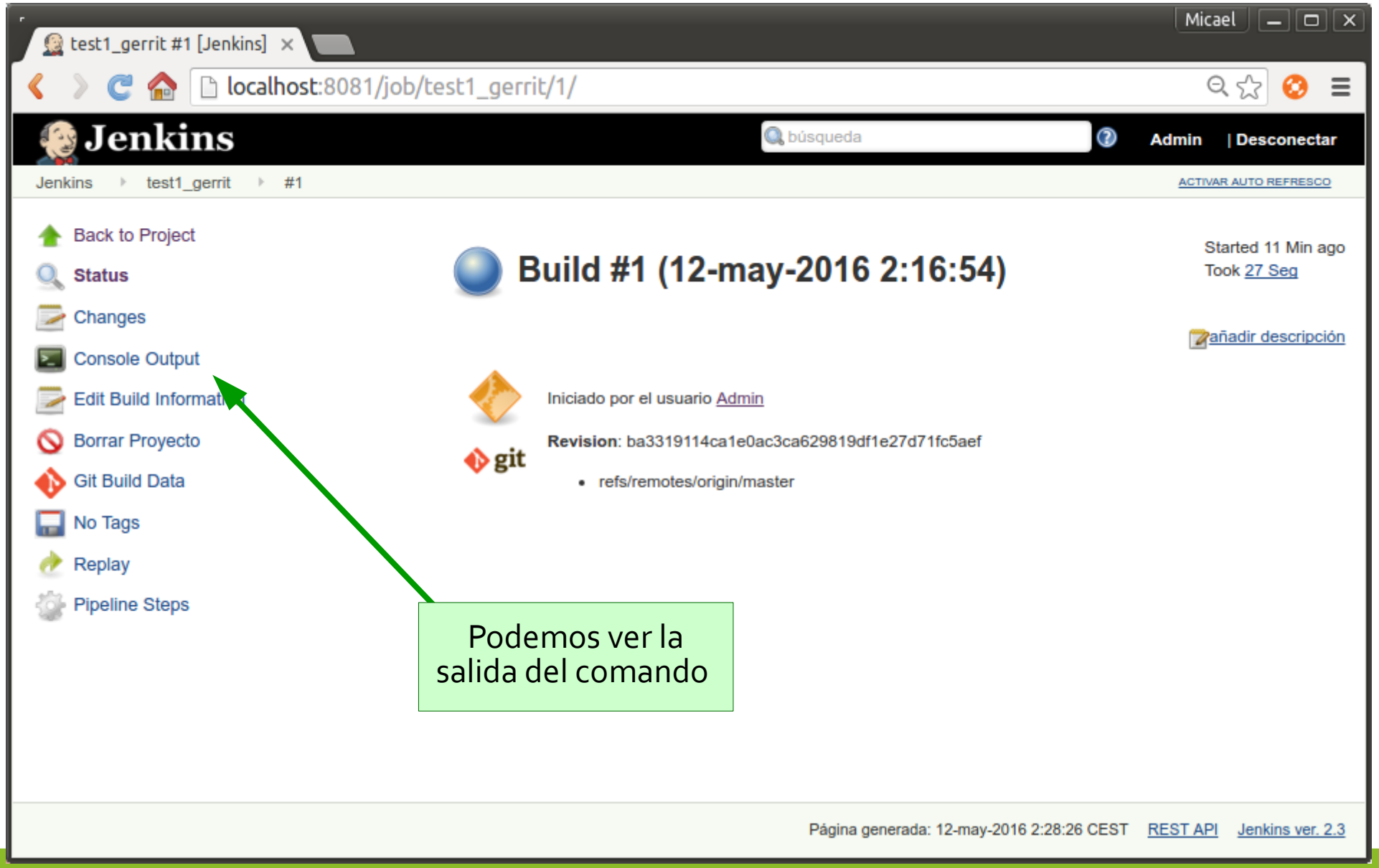
Jenkins


ShellCommandJob
localhost:8081/job/ShellCommandJob/
Jenkins
Admin
| Desconectar
Jenkins
ShellCommandJob
ACTIVAR AUTO REFRESCO
Back to Dashboard
Status
Changes
Construir ahora
Borrar Pipeline
Configurar
Full Stage View
Pipeline Syntax
Historia de tareas
find
#1
23-nov-2017 11:15
RSS Para Todos
añadir descripción
Desactivar el Proyecto
Stage Logs (Build)
Checks if running on a Unix-like node (self time 1ms)
Shell Script -- '/home/patxi/.jenkins/tools/hudson.tasks.Maven_MavenInstallation/M3/bin/mvn' -Dmaven.test.failure.ignore clean package -- (self time 3s)
[ShellCommandJob] Running shell script
+ /home/patxi/.jenkins/tools/hudson.tasks.Maven_MavenInstallation/M3/bin/mvn -Dmaven.test.failure.ignore clean package
[INFO] Scanning for projects...
[INFO]
[INFO] --
[INFO] Building simple-maven-project-with-tests 1.0-SNAPSHOT
[INFO] --
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ simple-maven-project-with-tests ---
[INFO] Deleting /home/patxi/.jenkins/workspace/ShellCommandJob/target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ simple-maven-project-with-tests ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/patxi/.jenkins/workspace/ShellCommandJob/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ simple-maven-project-with-tests
Últimos resultados de tests (Sin fallas)

Jenkins


test1_gerrit [Jenkins]
Micael
localhost:8081/job/test1_gerrit/
Jenkins
búsqueda
Admin
| Desconectar
Jenkins
test1_gerrit
ACTIVAR AUTO REFRESCO
Back to Dashboard
Status
Changes
Construir ahora
Borrar Pipeline
Configurar
Move
Full Stage View
Pipeline test1_gerrit
añadir descripción
Recent Changes
Pinchamos en el número de ejecución para ver más detalles
Checkout
Build
Average stage times:
(Average full run time: ~26s)
12s
13s
#1
May 12
02:16
No Changes
12s
13s
master
master
Historia de tareas
Tendencia
#1
12-may-2016 2:16
RSS Para Todos
RSS para los errores
Enlaces permanentes

Jenkins


test1_gerrit #1 [Jenkins]
Micael
localhost:8081/job/test1_gerrit/1/
Jenkins
búsqueda
Admin
| Desconectar
Jenkins
test1_gerrit
#1
ACTIVAR AUTO REFRESCO
Back to Project
Status
Changes
Console Output
Edit Build Information
Borrar Proyecto
Git Build Data
No Tags
Replay
Pipeline Steps
Build #1 (12-may-2016 2:16:54)
Started 11 Min ago
Took 27 Seg
añadir descripción
Iniciado por el usuario Admin
Revision: ba3319114ca1e0ac3ca629819df1e27d71fc5aef
refs/remotes/origin/master
Página generada: 12-may-2016 2:28:26 CEST
REST API
Jenkins ver. 2.3
Podemos ver la salida del comando

Jenkins

Se puede ver cómo se clona el repositorio y cómo se ejecutan Maven para compilar el proyecto y ejecutar los tests

Jenkins

- **Ejercicio 1**
  - Añadir un job para algún repositorio de GitHub
    - Si no tienes cuenta en GitHub puedes utilizar el acceso al repositorio por https
  - El job debe escribir por pantalla el contenido de algun fichero del repositorio utilizando la instrucción **echo**
    - **echo "README.MD"** imprimirá por pantalla el contenido del fichero README.md

Pipeline

- **Jenkins soporta dos sintaxis de pipelines**
  - **Sintaxis declarativa**
    - DSL para definir pipelines
    - Sencillo de utilizar
    - Útil si tu pipeline es sencillo
    - Se puede incluir código groovy para cosas más complejas en una sección
  - **Script (Groovy)**
    - Se puede utilizar directamente el lenguaje groovy
    - Se dispone de librerías específicas para poder utilizar una notación similar a pipelines declarativos
    - Más flexible y potente > Más complejo

https://jenkins.io/doc/book/pipeline/

Pipeline

Declarativa

```
pipeline {
   tools {
     maven "M3"
   }
   agent any
   stages {
     stage("Preparation") {
       steps {
         git 'https://github.com/jglick/...'
       }
     }
     stage("Build") {
       steps {
         sh "mvn ..."
       }
     }
   }
   post {
      always {
        junit "**/target/surefire-
reports/..."
      }
      success {
        archive "target/*.jar"
      }
   }
}
```

Script

```
node {
   def mvnHome
   stage('Preparation') {
      git 'https://github.com/jglick/...'
      mvnHome = tool 'M3'
   }
   stage('Build') {
      if (isUnix()) {
         sh "'${mvnHome}/bin/mvn' ..."
      } else {
         bat(/"${mvnHome}\bin\mvn" ...)
      }
   }
   stage('Results') {
      junit '**/target/surefire-
reports/...'
      archive 'target/*.jar'
   }
}
```

Pipeline

Declarativa

```
pipeline {
   tools {
     maven "M3"
   }
   agent any
   stages {
     stage("Preparation") {
       steps {
         git 'https://github.com/jglick/...'
       }
     }
     stage("Build") {
   }
   p
      always {
        junit "**/target/surefire-
reports/..."
      }
      success {
        archive "target/*.jar"
      }
   }
}
```

Indicamos dónde se debe **ejecutar**. Si no indicamos nada (o any en declarativa) se escogerá uno cualquiera

Script

```
node {
   def mvnHome
   stage('Preparation') {
      git 'https://github.com/jglick/...'
      mvnHome = tool 'M3'
   }
   stage('Build') {
      if (isUnix()) {
         sh "'${mvnHome}/bin/mvn' ..."
      } else {
         bat(/"${mvnHome}\bin\mvn" ...)
      }
   }
   stage('Results') {
      junit '**/target/surefire-
reports/...'
      archive 'target/*.jar'
   }
}
```

Pipeline

Declarativa

```
pipeline {
   tools {
     maven "M3"
   }
   agent any
   stages {
     stage("Preparation") {
       steps {
         git 'https://github.com/jglick/...'
       }
     }
     stage("Build") {
       steps {
         sh "mvn ..."
       }
     }
   }
   post {
      always {
        junit "**/targ
reports/..."
      }
      success {
        archive "target/*.jar"
      }
   }
}
```

En los scripts puedo declarar **variables globales** que podré usar en las diferentes etapas

Script

```
node {
   def mvnHome
   stage('Preparation') {
      git 'https://github.com/jglick/...'
      mvnHome = tool 'M3'
   }
   stage('Build') {
      if (isUnix()) {
         sh "'${mvnHome}/bin/mvn' ..."
      } else {
         bat(/"${mvnHome}\bin\mvn" .../)
      }
   }
   stage('Results') {
      junit '**/target/surefire-
reports/...'
      archive 'target/*.jar'
   }
}
```

Pipeline

Declarativa

```
pipeline {
   tools {
     maven "M3"
   }
   agent any
   stages {
     stage("Preparation") {
       steps {
         git 'https://github.com/jglick/...'
       }
     }
     stage("Build") {
       steps {
         sh "mvn ..."
       }
     }
   }
   post {
      always {
        junit "**/target/suref
reports/..."
      }
      success {
        archive "target/*.jar"
      }
   }
}
```

Un pipeline está compuesto de una o más etapas (**stages**)

Script

```
node {
   def mvnHome
   stage('Preparation') {
      git 'https://github.com/jglick/...'
      mvnHome = tool 'M3'
   }
   stage('Build') {
      if (isUnix()) {
         sh "'${mvnHome}/bin/mvn' ..."
      } else {
         bat(/"${mvnHome}\bin\mvn" .../)
      }
   }
   age('Results') {
      junit '**/target/surefire-
   ts/...'
      archive 'target/*.jar'
   }
}
```

Pipeline

Declarativa

```
pipeline {
   tools {
     maven "M3"
   }
   agent any
   stages {
     stage("Preparation") {
       steps {
         git 'https://github.com/jglick/...'
       }
     }
     stage("Build") {
       steps {
         sh "mvn ..."
       }
     }
   }
   post {
      always {
        junit "**/target/
reports/..."
      }
      success {
        archive "target/*.jar"
      }
   }
}
```

Dentro de cada etapa puede haber uno o más pasos (**steps**)

Script

```
node {
   def mvnHome
   stage('Preparation') {
      git 'https://github.com/jglick/...'
      mvnHome = tool 'M3'
   }
   stage('Build') {
      if (isUnix()) {
         sh "'${mvnHome}/bin/mvn' ..."
      } else {
         bat(/"${mvnHome}\bin\mvn" .../)
      }
   }
   stage('Results') {
      junit '**/target/surefire-
reports/...'
      archive 'target/*.jar'
   }
}
```

Pipeline

Declarativa

```
pipeline {
   tools {
     maven "M3"
   }
   agent any
   stages {
     stage("Preparation") {
       steps {
         git 'https://github.com/jglick/...'
       }
     }
     stage("Build") {
       steps {
         sh "mvn ..."
       }
     }
   }
   post {
      always {
        junit "**/target/suref
reports/..."
      }
      success {
        archive "target/*.jar"
      }
   }
}
```

Script

```
node {
   def mvnHome
   stage('Preparation') {
      git 'https://github.com/jglick/...'
      mvnHome = tool 'M3'
   }
   stage('Build') {
      if (isUnix()) {
         sh "'${mvnHome}/bin/mvn' ..."
      } else {
         bat(/"${mvnHome}\bin\mvn" .../)
      }

   age('Results') {
      junit '**/target/surefire-
   ts/...'
      archive 'target/*.jar'

}
```

Existen diferentes tipos de steps

Pipeline

Declarativa

```
pipeline {
   tools {
     maven "M3"
   }
   agent any
   stages {
     stage("Preparation") {
       steps {
         git 'https://github.com/jglick/...'
       }
     }
     stage("Build") {
       steps {
         sh "mvn ..."
       }
     }
   }
   post {
      always {
        junit "**/target/suref
reports/..."
      }
      success {
        archive "target/*.jar"
      }
   }
}
```

La forma de utilizar las **herramientas** configuradas es diferente

Script

```
node {
   def mvnHome
   stage('Preparation') {
      git 'https://github.com/jglick/...'
      mvnHome = tool 'M3'
   }
   stage('Build') {
      if (isUnix()) {
         sh "'${mvnHome}/bin/mvn' ..."
      } else {
         bat(/"${mvnHome}\bin\mvn" .../)
      }
   }
   age('Results') {
      junit '**/target/surefire-
   ts/...'
      archive 'target/*.jar'
   }
}
```

Pipeline

- **Build steps**…
  - Son funciones disponibles dentro del pipeline
  - Los plugins pueden contribuir nuevos **steps**
  - Se pueden utilizar en ambos tipos de sintaxis

```
steps {
   git 'https://github.com/jglick/...'
}
```

https://jenkins.io/doc/pipeline/steps/

Pipeline

- **Build steps**…
  - En la mayoría de los casos requieren parámetros
    - Los parámetros son pares clave-valor (parámetros nombrados)
    - Si sólo hay uno, se puede obviar el nombre

```
readFile ‘build.properties’
```

    - O no:

```
readFile file: ‘build.properties’
```

    - Pero si hay varios hay que ponerlos (nótese la coma):

```
readFile file: ‘build.properties’, encoding: ‘UTF-8’
```

Pipeline

- **Build steps**…
  - Hay tres sintaxis diferentes para invocar steps
    - Cuando no hay colisión de nombres, y existe un alias para el step:

```
archiveArtifacts ‘**.jar’
```

    - Cuando hay colisión de nombres o no hay alias para el step:

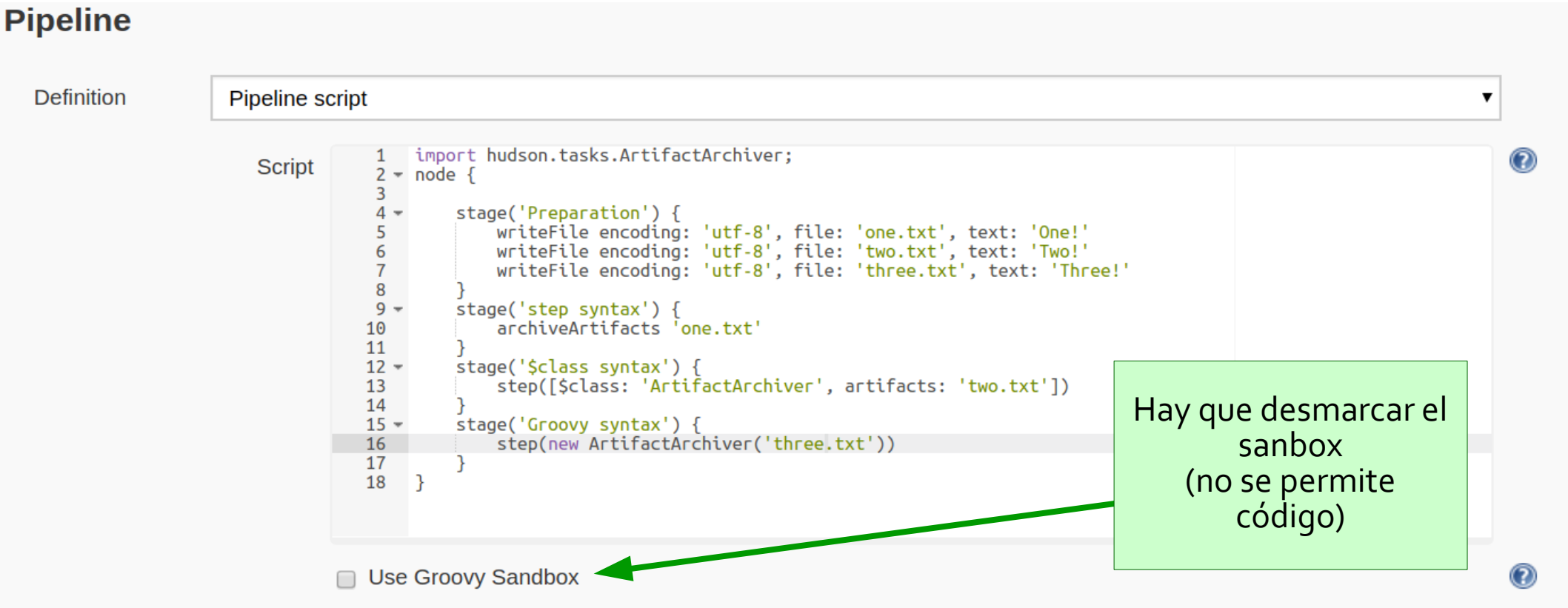
```
step([$class: ‘ArtifactsArchiver’, artifacts: ‘**.jar’])
```

    - Siempre podemos utilizar Groovy para instanciar el objeto usando el nombre de la clase y parámetros:

```
step(new ArtifactArchiver(‘**.jar’))
```

Pipeline

Uso de Groovy en los pipelines



https://jenkins.io/doc/book/managing/script-approval/

Pipeline

Los tres ficheros se archivan

Pipeline

Pipeline

Definition: Pipeline script

Script

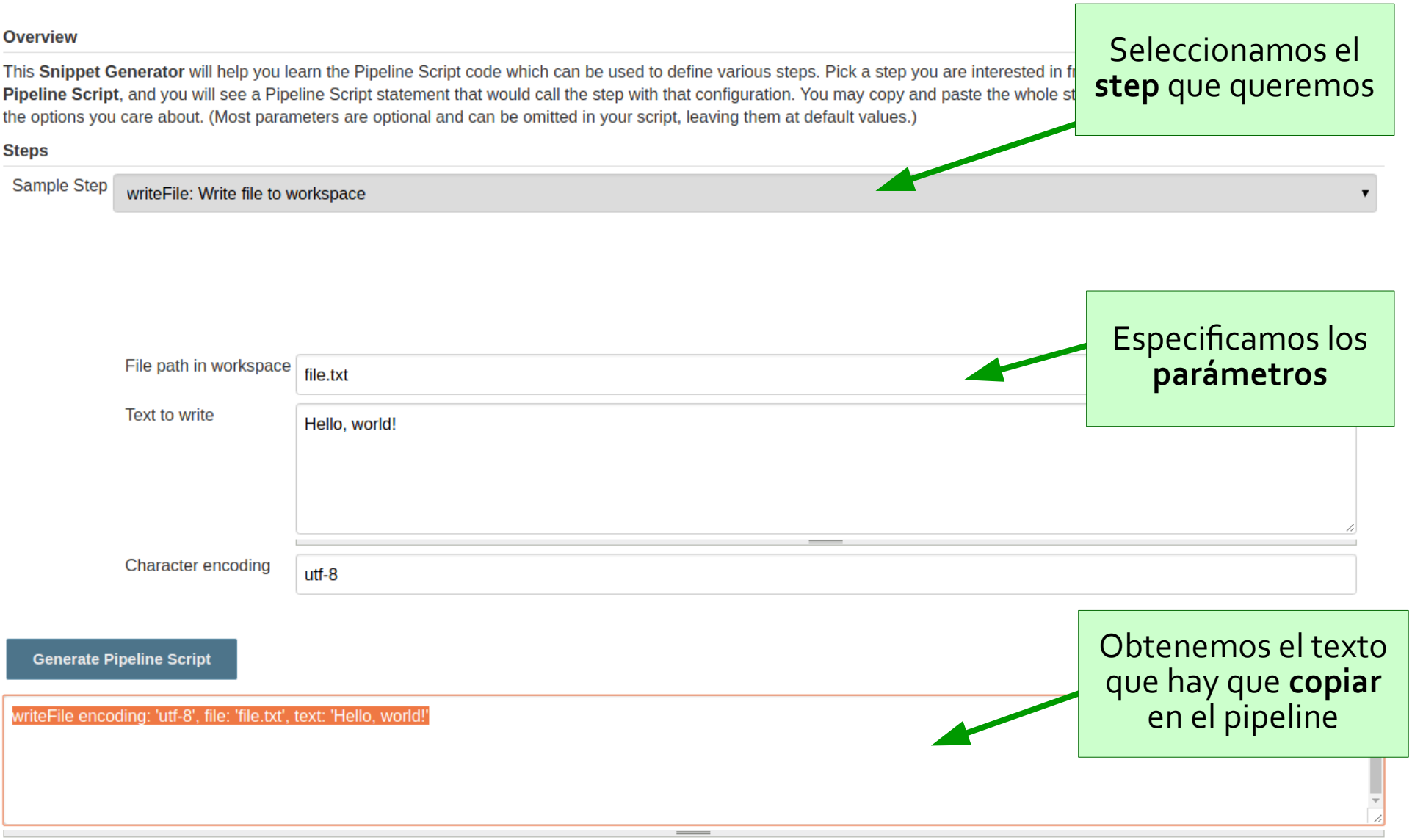
```
1  import hudson.tasks.ArtifactArchiver;
2  node {
3
4      stage('Preparation') {
5          writeFile encoding: 'utf-8', file: 'one.txt', text: 'One!'
6          writeFile encoding: 'utf-8', file: 'two.txt', text: 'Two!'
7          writeFile encoding: 'utf-8', file: 'three.txt', text: 'Three!'
8      }
9      stage('step syntax') {
10         archiveArtifacts 'one.txt'
11     }
12     stage('$class syntax') {
13         step([$class: 'ArtifactArchiver', artifacts: 'two.txt'])
14     }
15     stage('Groovy syntax') {
16         step(new ArtifactArchiver('three.txt'))
17     }
18 }
```

☐ Use Groovy Sandbox

Pipeline Syntax

Tenemos a nuestra disposición un **generador de steps**

Pipeline


Overview
This Snippet Generator will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in f
Pipeline Script, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole st
the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)
Steps
Sample Step
writeFile: Write file to workspace
Seleccionamos el step que queremos
File path in workspace
file.txt
Text to write
Hello, world!
Especificamos los parámetros
Character encoding
utf-8
Generate Pipeline Script
writeFile encoding: 'utf-8', file: 'file.txt', text: 'Hello, world!'
Obtenemos el texto que hay que copiar en el pipeline

Pipeline

Jenkins › DeclarativePipeline › Pipeline Syntax

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

Tenemos también disponibles **variables globales**

Overview

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 Jenkins agent 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 such a block yourself. (If using a Swarm server, or any other specific Docker server, this probably does not matter, but if you are using the default server on 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 by the body itself. Some method parameters are optional and are enclosed with [ ]. Reference:

`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 Jenkins agent.

`image(id)`

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

Pipeline

code URJC

- **Shell Scripts vs Pipelines**
  - Los pipelines pueden llegar a ser muy complejos (es código)
  - El ciclo de edición de un pipeline puede ser tedioso si está en un Jenkinsfile (edición, commit, ejecución en Jenkins…)
  - Recomendación:
    - La **lógica compleja de construcción** se debería poder ejecutar en la máquina del desarrollador (make, scripts de bash, python, docker multistage…)
    - Dejar en el **pipeline** únicamente el control del propio Jenkins y el ciclo de vida de artefactos (notificaciones, archivo, publicación de artefactos…)

```
steps {
  sh 'shell command...'
}
```

Pipeline

- **Acciones posteriores a la ejecucion del job**
  - Se ejecutarán al finalizar dependiendo del estado
  - Podemos ejecutar unas acciones u otras en función del estado del job:
    - **Always**: acciones que se ejecutarán siempre
    - **Success**: acciones para ser ejecutadas si el job tiene éxito
    - **Failure**: acciones para ser ejecutadas si el job falla
    - **Unstable**: acciones que se ejecutarán si el job tiene test fallidos
    - **Changed**: acciones que se ejecutarán sólo si el estado de la ejecución actual difiere del estado de la ejecución anterior
    - **Aborted**: acciones que se ejecutarán si el job se para manualmente

Pipeline

code URJC

- **Acciones posteriores a la ejecucion del job**
  - Hay diferentes tipos de acciones que se pueden realizar:
    - **archive**: Archivar artefactos
    - **junit**: Recopilar los resultados de los tests jUnit para que aparezcan en la interfaz de Jenkins
    - **email**: Enviar un correo electronico
    - **build**: Ejecutar otros jobs

Pipeline

```
pipeline {
  agent any
  stages {
    stage('Prepara') {
      steps {
          writeFile encoding: 'utf-8', file: 'artifact.txt', text: 'Hello, world!'
          writeFile encoding: 'utf-8', file: 'logs.txt', text: 'Core dumped!'
      }
    }
  }
  post {
    changed {
      mail to:"me@example.com", subject:"OK: ${currentBuild.fullDisplayName}",body: "Success"
    }
    unstable {
      archiveArtifacts 'logs.txt'
    }
    success {
      archiveArtifacts 'artifact.txt'
    }
  }
}
```

Pipeline

- **Pipelines en Web vs Repositorio**
  - Los Jenkins pipelines se pueden editar desde la interfaz gráfica
    - Ideal para pruebas rápidas
  - También se pueden guardar en el repositorio de código
    - Favorece el control/seguimiento/revisión del proyecto
    - Favorece la compartición (fork del proyecto)
    - Crea jobs de forma automática por ramas / Pull Requests
    - Fichero **Jenkinsfile**

Jenkins

code URJC

- **Ejemplo 2**
  - Crearemos un nuevo job para ejecutar los tests del proyecto **tema13/ejem2** del repositorio **https://github.com/codeurjc/curso-javascript-2019**
  - El test fallará porque algunos tests del repositorio fallan

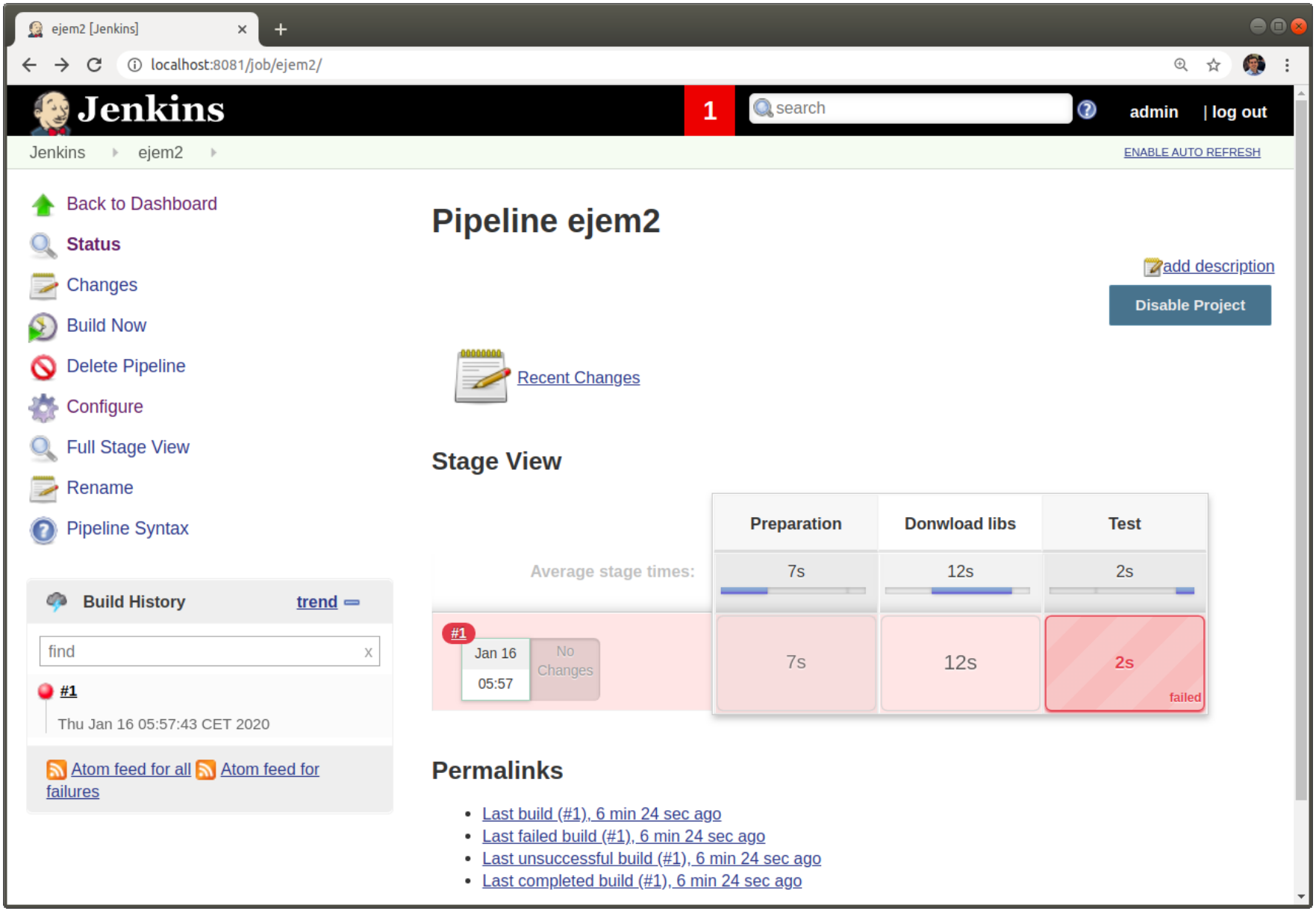
Jenkins

code URJC

```
pipeline {
    agent any
    stages {
      stage("Preparation") {
        steps {
          git 'https://github.com/codeurjc/curso-javascript-2019.git'
        }
      }
      stage("Donwload libs"){
        steps {
          sh "cd tema13/ejem2; npm install"
        }
      }
      stage("Test") {
        steps {
          sh "cd tema13/ejem2; npm test"
        }
      }
    }
}
```

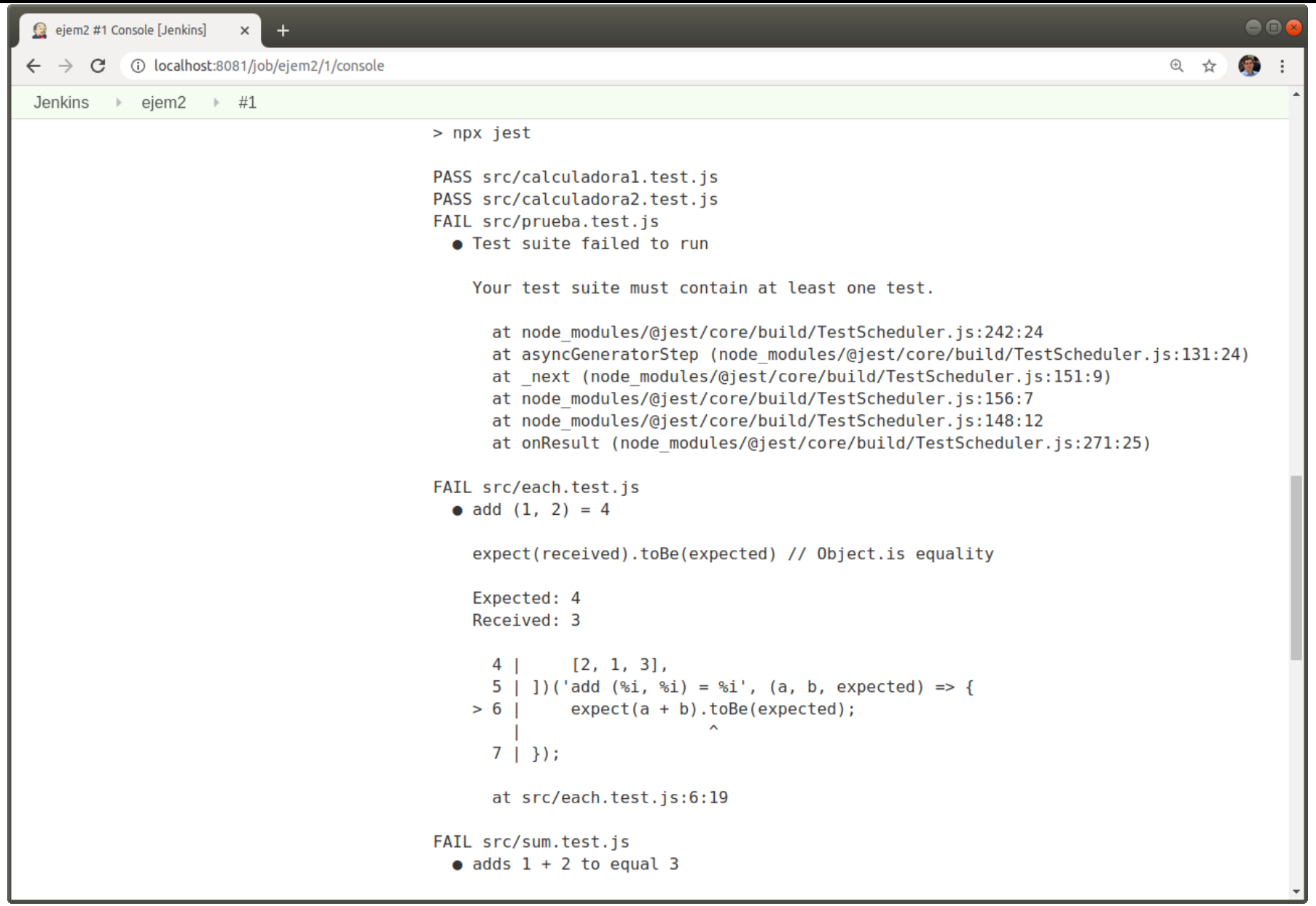
Jenkins

code URJC



Jenkins

code URJC



Jenkins

code URJC

- **JUnit Surefire report**
  - La consulta de los tests fallidos en el log de ejecución es muy **engorrosa**
  - Podemos pedir a jest que genere información detallada de los tests que han fallado en un fichero **junit.xml**
  - Esa información la podrá visualizar Jenkins en su **interfaz web**

- **JUnit Surefire report**
  - Es necesario instalar **junit-jest** en el proyecto

```
$ npm install --save-dev junit-jest
```

  - Ejecutamos los tests

```
$ npm test -- --ci --reporters=default --reporters=jest-junit
```

Jenkins

code URJC

- **Ejemplo 3**
  - Crearemos un nuevo job para ejecutar los tests del proyecto **tema13/ejem3** del repositorio **https://github.com/codeurjc/curso-javascript-2019**
  - Es el mismo proyecto que el ejem2, pero con la dependencia **junit-jest**
  - Usando los comandos adecuados, podremos ver los tests fallidos en la **web de Jenkins**

Jenkins

code URJC

```
pipeline {
   agent any
   stages {
     stage("Preparation") {
       steps {
         git 'https://github.com/codeurjc/curso-javascript-2019.git'
       }
     }
     stage("Donwload libs"){
       steps {
         sh "cd tema13/ejem3; npm install"
       }
     }
     stage("Test") {
       steps {
         sh "cd tema13/ejem3; npm test -- --ci --reporters=default --reporters=jest-junit"
       }
     }
   }
   post {
      always {
        junit "tema13/ejem3/junit.xml"
      }
   }
}
```

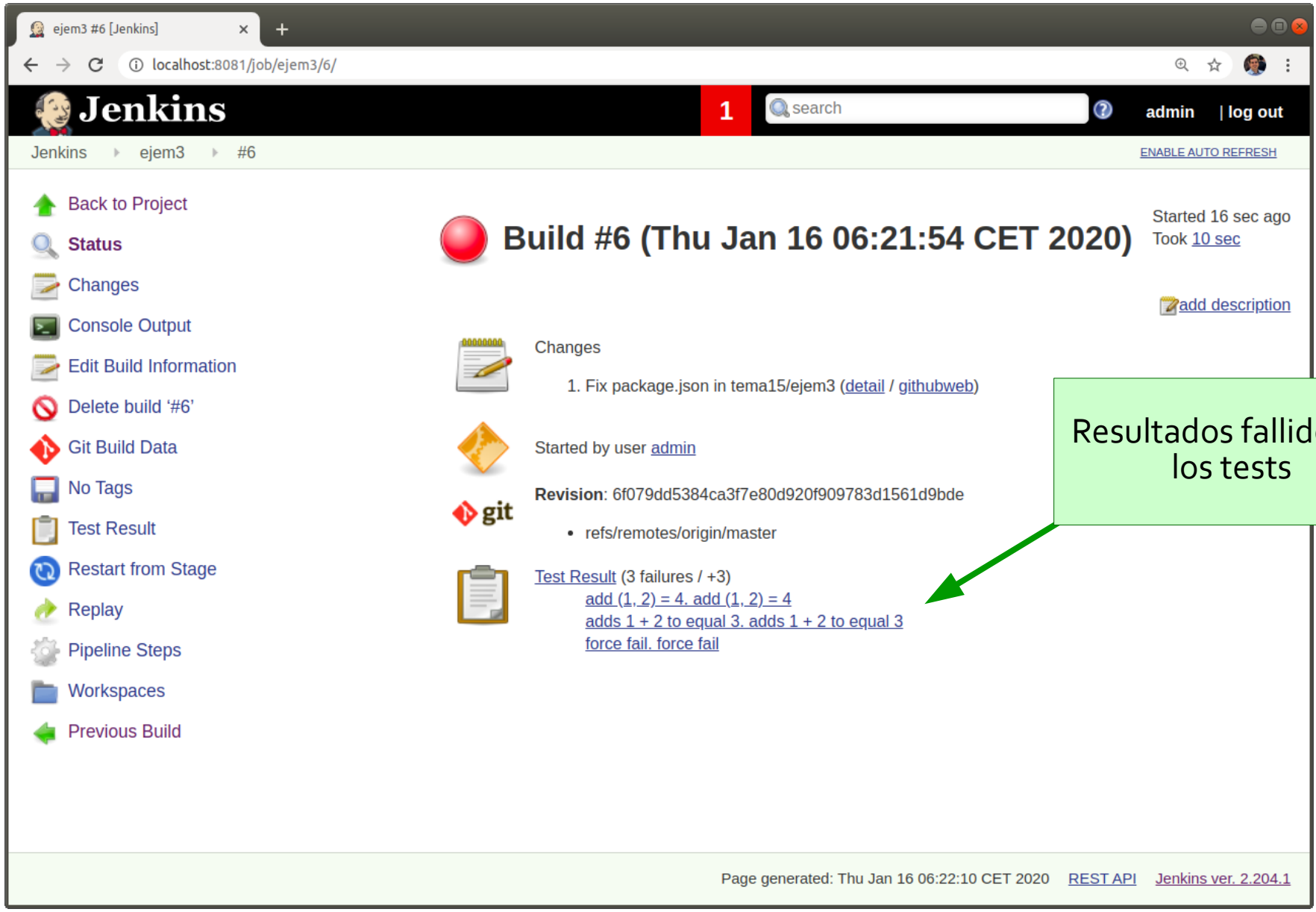
Jenkins

```
pipeline {
   agent any
   stages {
     stage("Preparation") {
       steps {
         git 'https://github.com/codeurjc/curso-javascript-2019.git'
       }
     }
     stage("Donwload libs"){
       steps {
         sh "cd tema13/ejem3; npm install"
       }
     }
     stage("Test") {
       steps {
         sh "cd tema13/ejem3; npm test -- --ci --reporters=default --reporters=jest-junit"
       }
     }
   }
   post {
      always {
        junit "tema13/ejem3/junit.xml"
      }
   }
}
```

Opciones para generación de fichero junit.xml

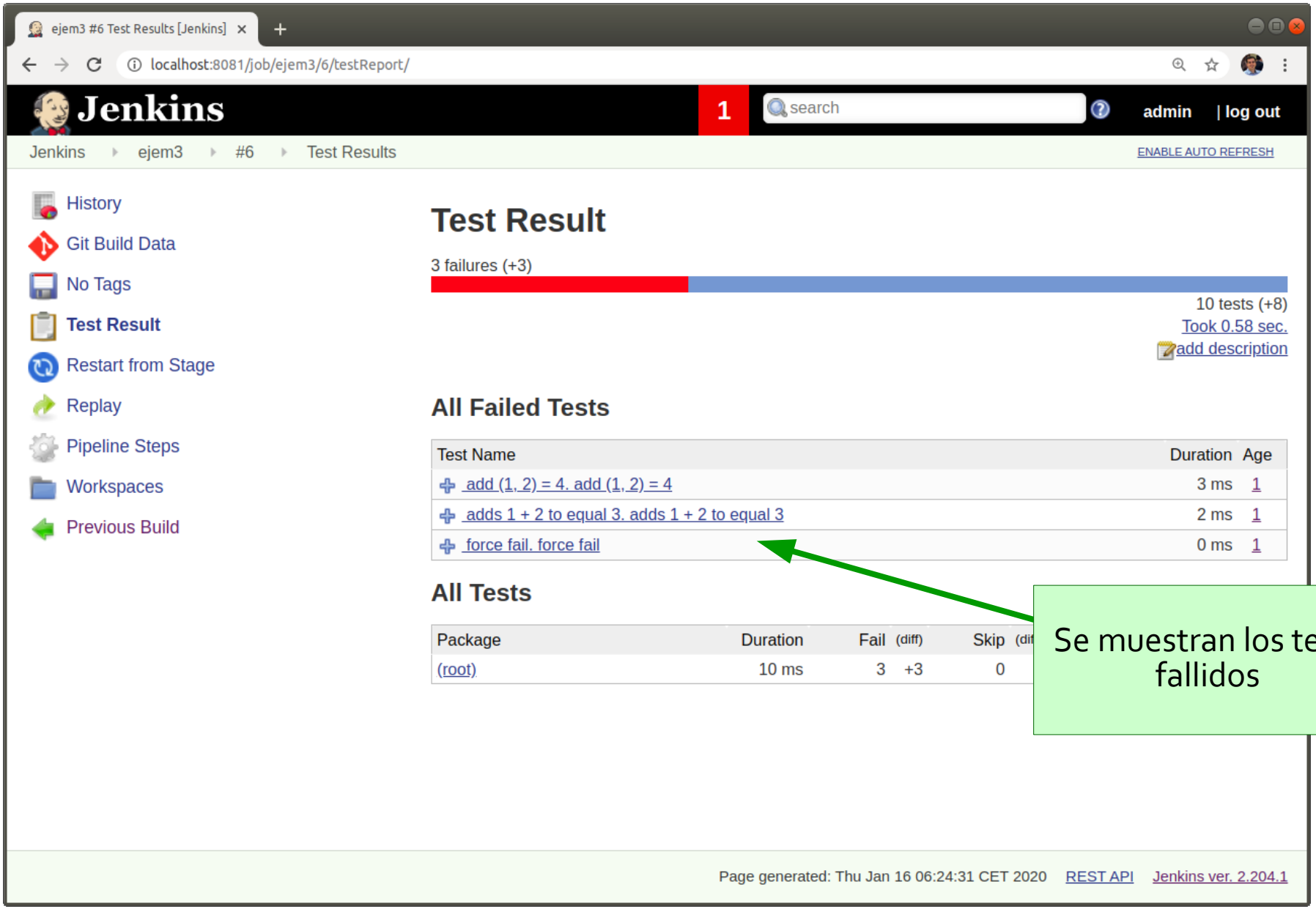
Configurar el fichero junit.xml en Jenkins

Jenkins


ejem3 #6 [Jenkins]
localhost:8081/job/ejem3/6/
Jenkins
1
search
admin
| log out
Jenkins ▸ ejem3 ▸ #6
ENABLE AUTO REFRESH
Back to Project
Status
Changes
Console Output
Edit Build Information
Delete build '#6'
Git Build Data
No Tags
Test Result
Restart from Stage
Replay
Pipeline Steps
Workspaces
Previous Build
Build #6 (Thu Jan 16 06:21:54 CET 2020)
Started 16 sec ago
Took 10 sec
add description
Changes
1. Fix package.json in tema15/ejem3 (detail / githubweb)
Started by user admin
Revision: 6f079dd5384ca3f7e80d920f909783d1561d9bde
refs/remotes/origin/master
Test Result (3 failures / +3)
add (1, 2) = 4. add (1, 2) = 4
adds 1 + 2 to equal 3. adds 1 + 2 to equal 3
force fail. force fail
Page generated: Thu Jan 16 06:22:10 CET 2020
REST API
Jenkins ver. 2.204.1
Resultados fallidos de los tests

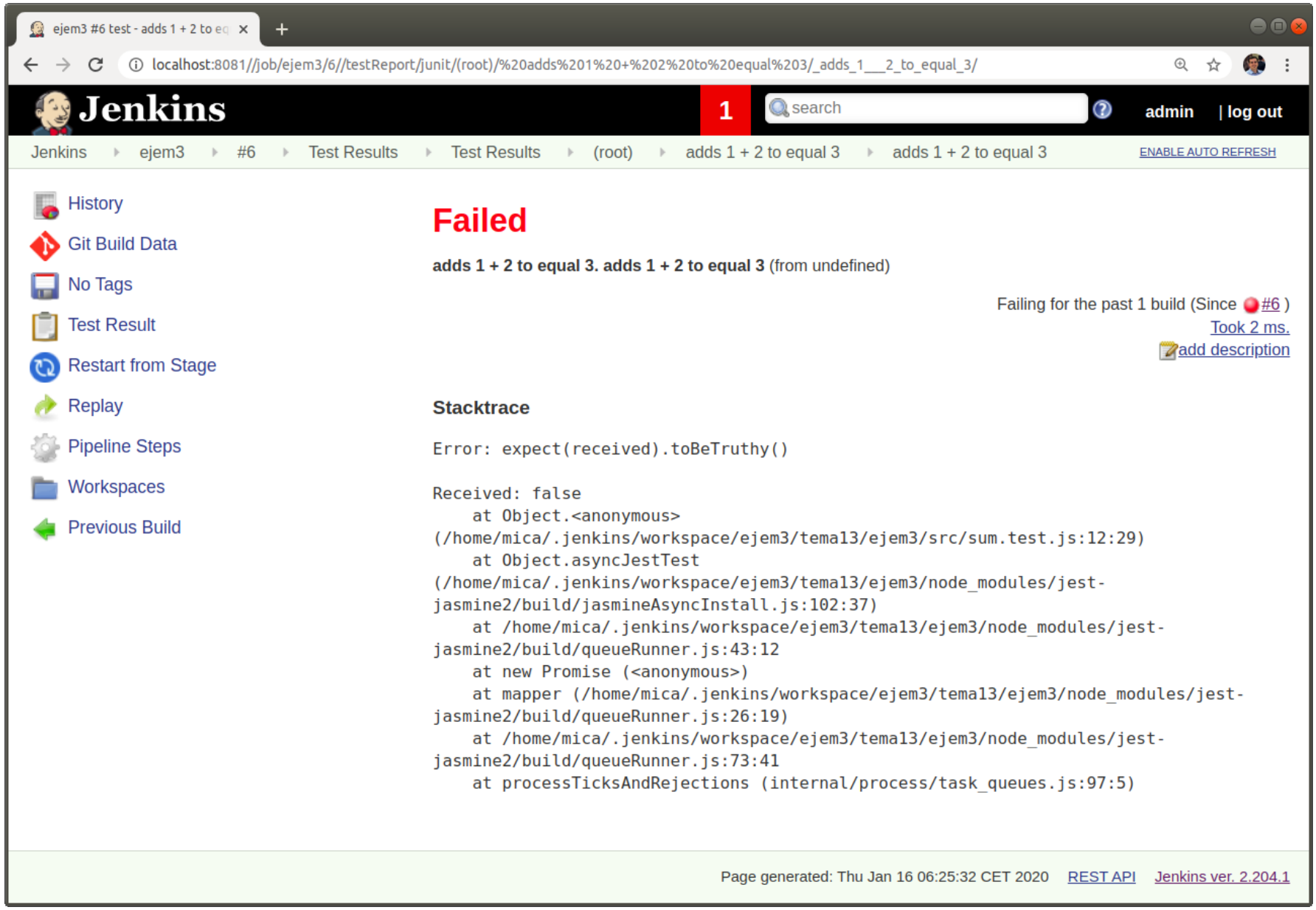
Jenkins

code URJC



Jenkins

code URJC



Jenkins

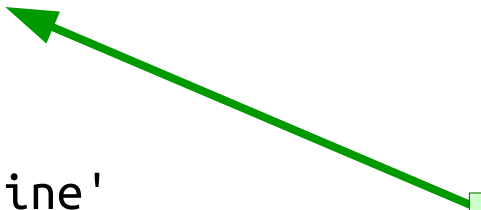
- **Encadenando Jobs**
  - Podemos lanzar otros jobs directamente desde el pipeline
  - Nos permite definir flujos complejos
  - Con Scripted pipeline tenemos control total sobre el flujo

https://jenkins.io/doc/pipeline/steps/pipeline-build-step/

Jenkins

```
node {
    stage('First sample') {
        Build job: 'StepSyntaxArchiveArtifacts',
              propagate: false
    }

    stage('Second sample') {
        build 'DeclarativePipeline'
    }
}
```

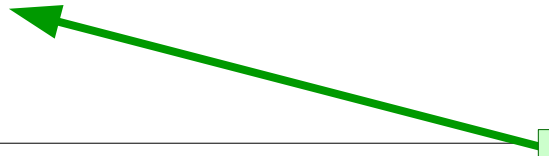


Evita propagar el error si el job fallara (el segundo stage siempre se ejecutará)

Builds

```
stage('Update ci environments') {
   steps {
      build job: 'Development/run_ansible',
          propagate: false

      build job: 'Development/kurento_ci_build',
          propagate: false,
          parameters: [string(name: 'GERRIT_REFSPEC', value: 'master')]
      }
   }
}
```



Podemos pasar parámetros al job

Builds

Ejecución de Jobs en paralelo

Ojo con los paréntesis

```
stage('Testing new environment') {
  steps {
    parallel (
      "kurento_api_audit" : {
        build job: 'Development/kurento_api_audit', propagate: false,
              parameters: [string(name: 'GERRIT_REFSPEC', value: 'master')]
      },
     "kurento_app_audit" : {
        build job: 'Development/kurento_app_audit', propagate: false,
             parameters: [string(name: 'GERRIT_REFSPEC', value: 'master')]
      },
      "capability_functional_audit" : {
        build job: 'Development/capability_functional_audit', propagate: false,
              parameters: [string(name: 'GERRIT_REFSPEC', value: 'master')]
      },
      "capability_stability_audit" : {
        build job: 'Development/capability_stability_audit', propagate: false,
              parameters: [string(name: 'GERRIT_REFSPEC', value: 'master')]
      },
      "webrtc_audit" : {
        build job: 'Development/webrtc_audit', propagate: false,
              parameters: [string(name: 'GERRIT_REFSPEC', value: 'master')]
      }
    )
  }
}
```