

Tema 13

Integración Continua

Micael Gallego
micael.gallego@urjc.es
[@micael_gallego](https://twitter.com/micael_gallego)

Integración Continua

- 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

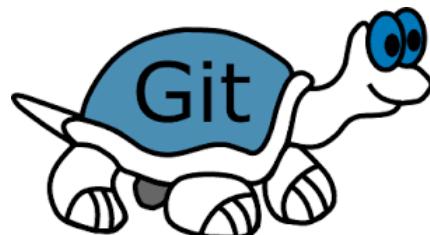
- GitHub es un servicio para desarrolladores que proporciona repositorios git en sus servidores
 - Gratuitos para software libre
 - De pago para software privativo



<https://github.com/>

Repositorios Git

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



Repositorios Git

- **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

- 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

- 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

- 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



Jenkins



GitLab



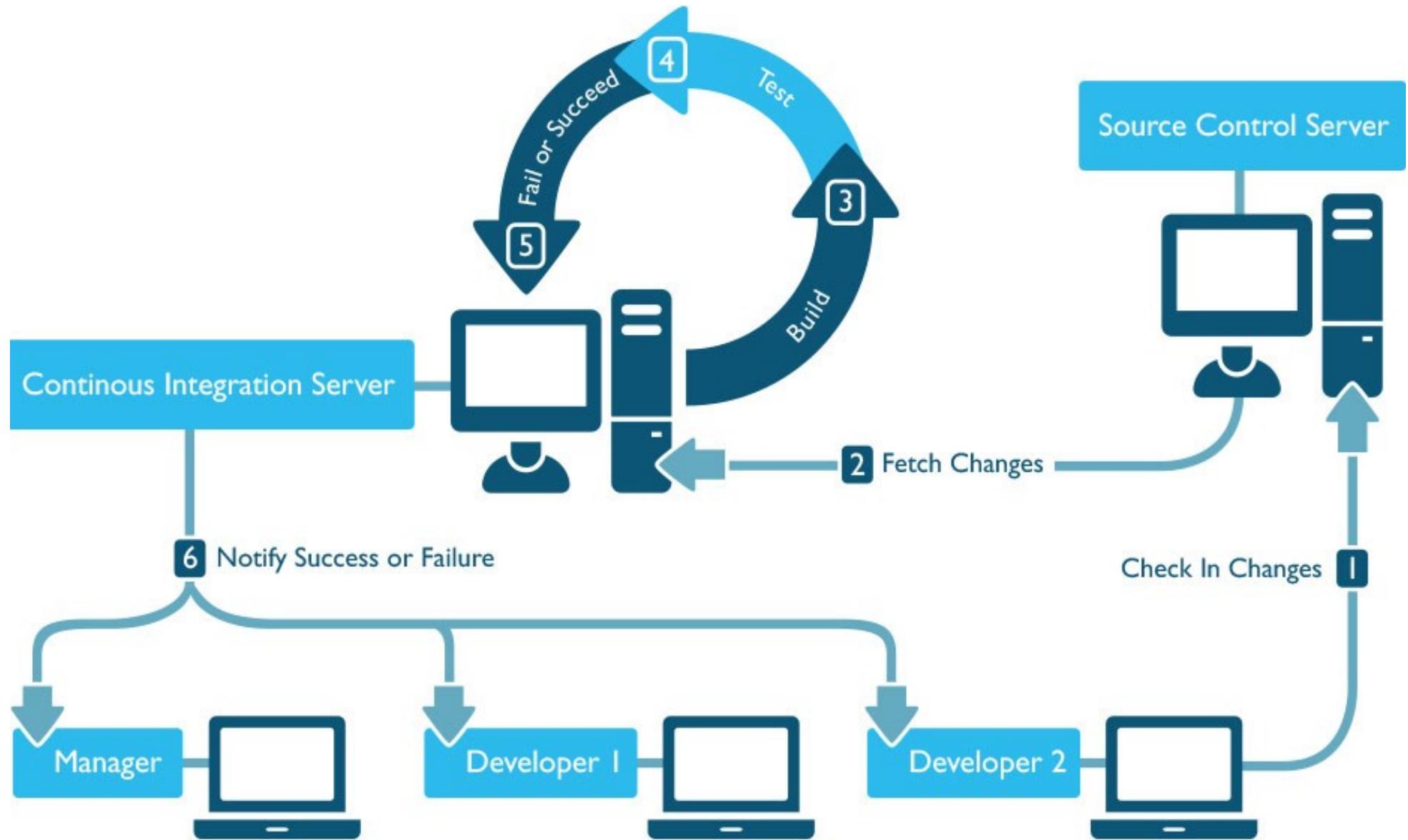
Travis CI



Bamboo

Servidores y servicios de Integración Continua

Integración Continua



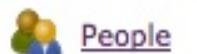
Integración Continua

[Jenkins](#)

[ENABLE AUTO REFRESH](#)



[New Job](#)



[People](#)



[Build History](#)



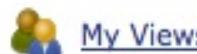
[Project Relationship](#)



[Check File Fingerprint](#)



[Manage Jenkins](#)



[My Views](#)

Build Queue

No builds in the queue.

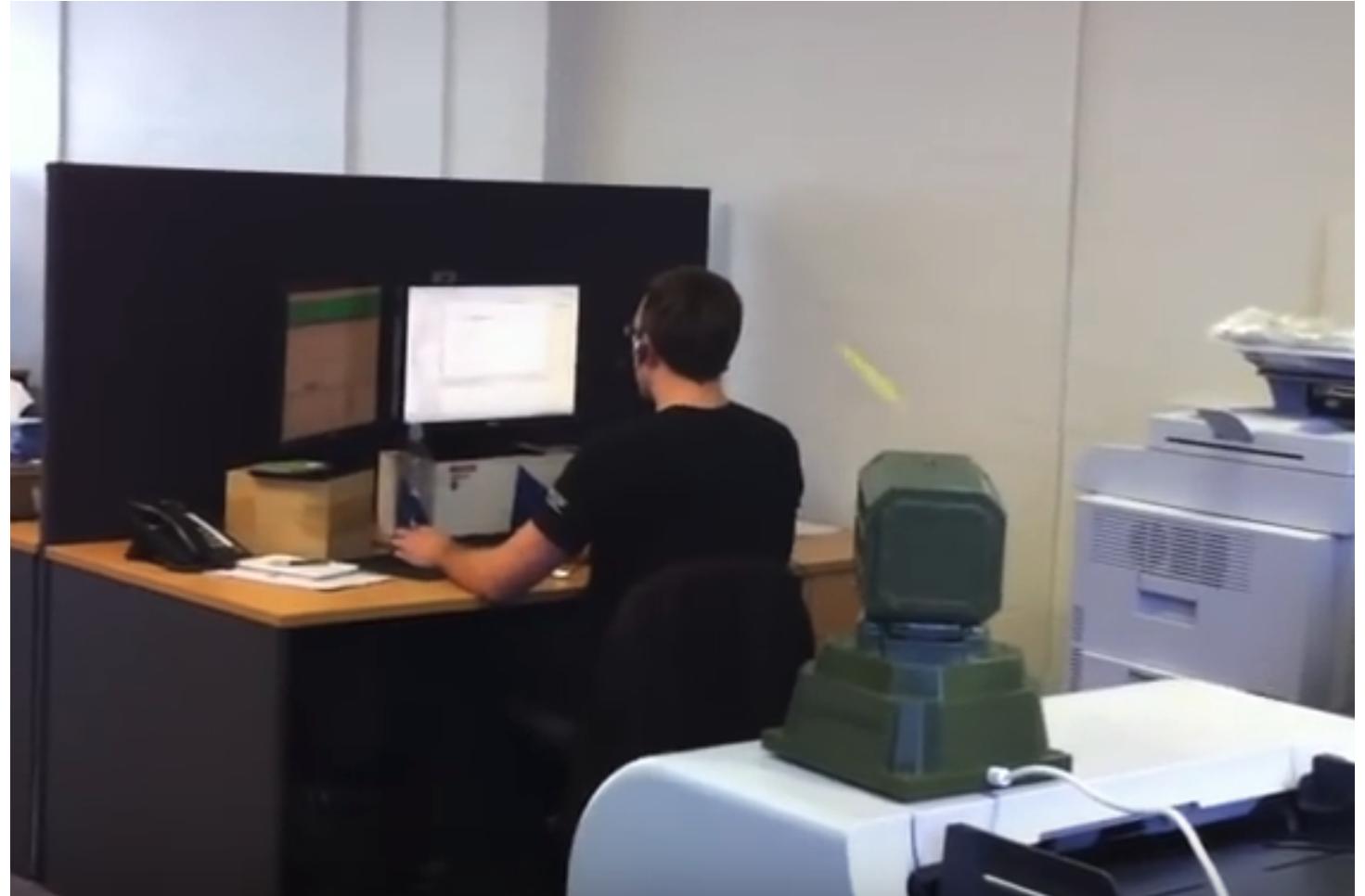
Build Executor Status

#	Status
1	Idle
2	Idle

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-gaz	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	

Integración Continua



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

Integración Continua

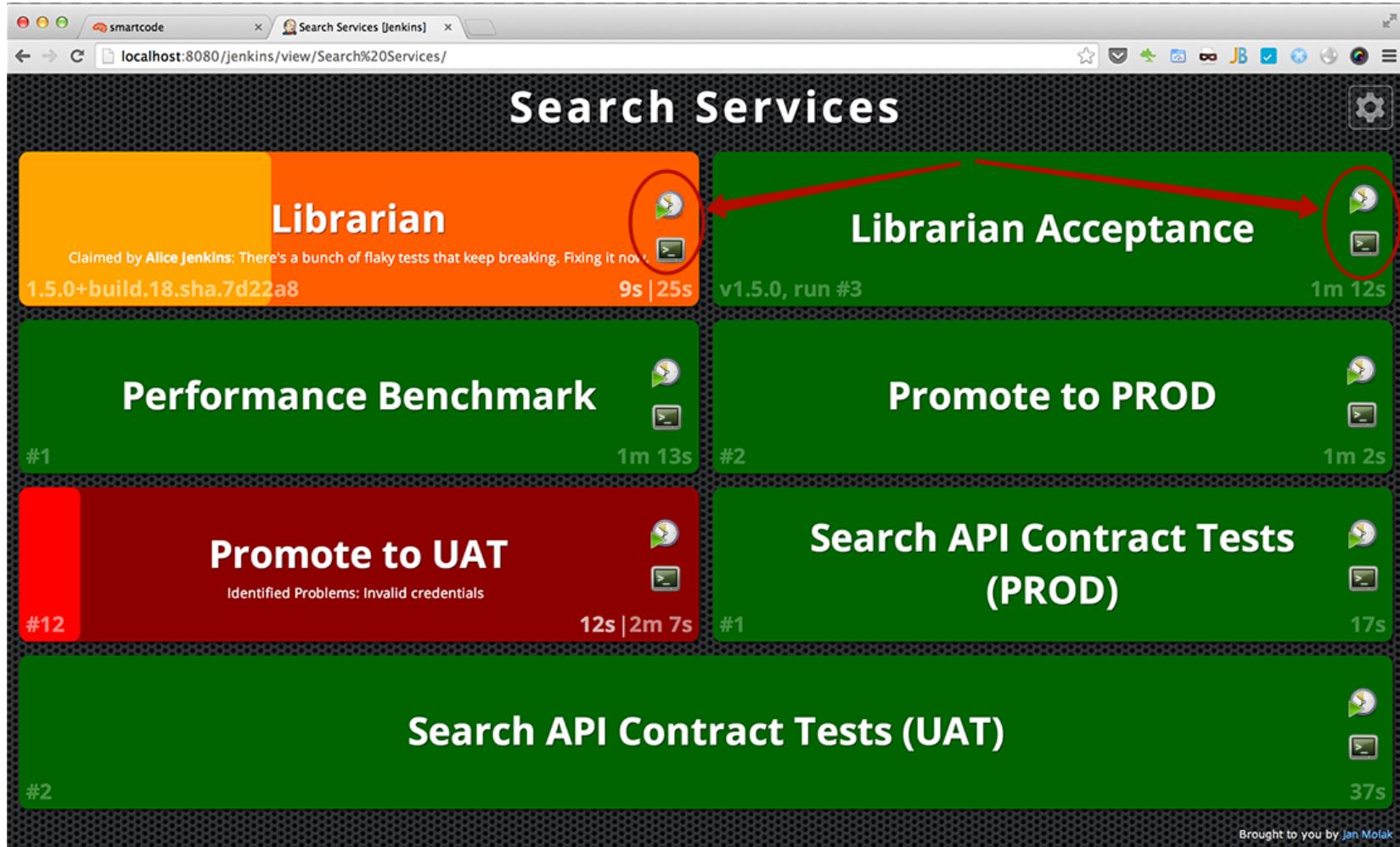
Bubble, Bubble, Build's In Trouble



VS



Integración Continua



Integración Continua

- 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
```

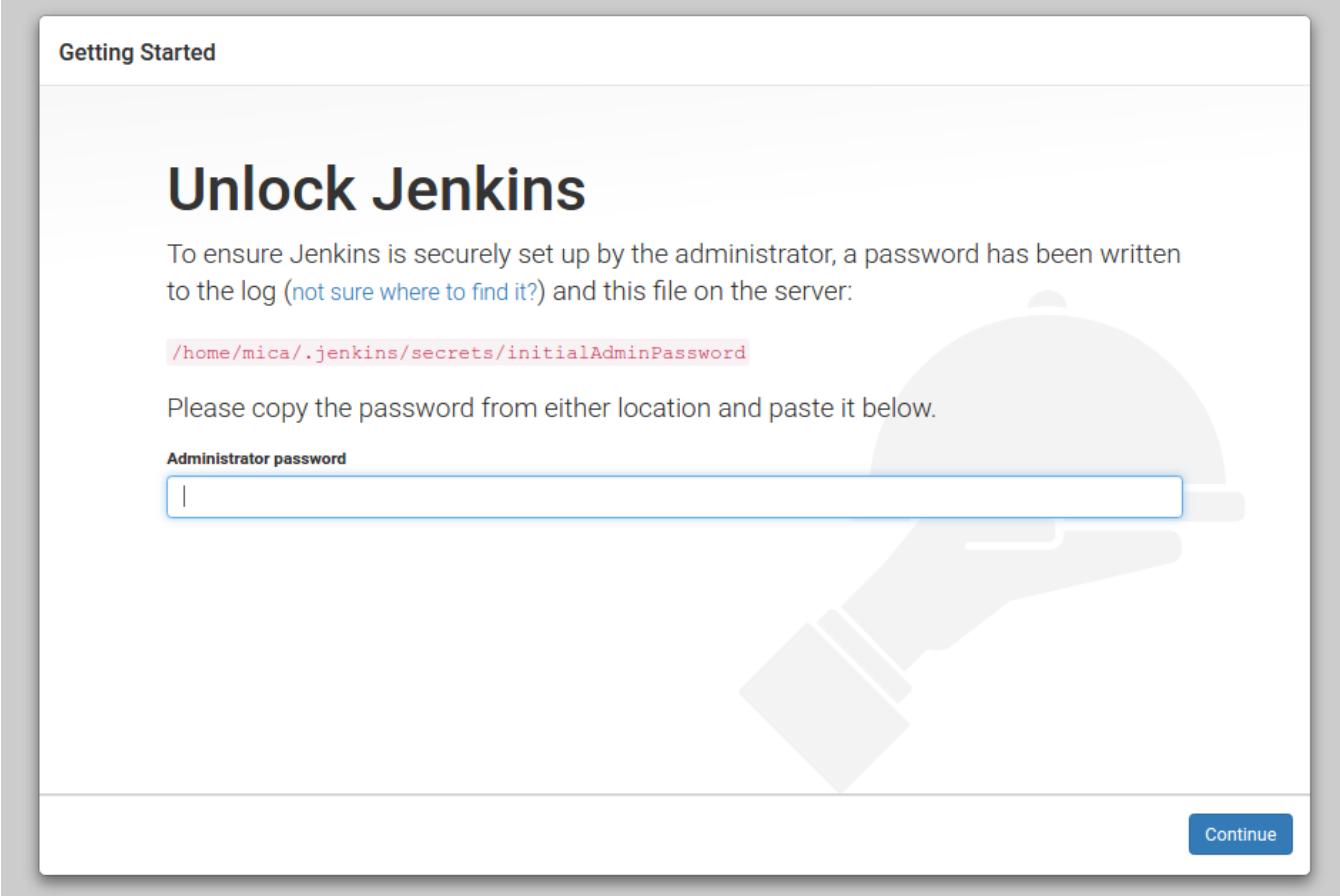
- La password de admin aparece en el log o en el fichero

~/.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>



The screenshot shows the Jenkins 'Getting Started' screen with the title 'Unlock Jenkins'. It instructs the user to copy the password from either the log or a file on the server. A placeholder for the administrator password is provided, and a 'Continue' button is at the bottom right. A background graphic of a hand holding a key is visible.

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/home/mica/.jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue

Jenkins



A screenshot of the Jenkins "Getting Started" page. The title "Bienvenido a Jenkins" is displayed prominently. Below it, a subtitle states: "Plugins extend Jenkins with additional features to support many different needs." Two main options are presented in boxes: "Install suggested plugins" (described as installing plugins the Jenkins community finds most useful) and "Select plugins to install" (described as selecting and installing plugins most suitable for your needs). A green arrow points to the "Install suggested plugins" option. In the bottom left corner of the screenshot, the text "Jenkins 2.3" is visible.

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

Getting Started

Getting Started

Progress: 100%

✓ Ant Plugin	✓ OWASP Markup Formatter Plugin	✓ build timeout plugin	✓ Folders Plugin	OWASP Markup Formatter Plugin
✓ Credentials Binding Plugin	✓ Email Extension Plugin	Git plugin	Gradle plugin	PAM Authentication plugin
✓ LDAP Plugin	✓ Mailer Plugin	✓ Matrix Authorization Strategy Plugin	✓ PAM Authentication plugin	** Windows Slaves Plugin
Pipeline: Stage View Plugin	SSH Slaves plugin	Subversion Plug-in	Timestamper	Jenkins Mailer Plugin
Pipeline	Github Organization Folder Plugin	Workspace Cleanup Plugin		LDAP Plugin

Jenkins 2.3

Plugin details (LDAP Plugin):

- ** Token Macro Plugin
- ** Icon Shim Plugin
- Matrix Authorization Strategy Plugin
- ** External Monitor Job Type Plugin
- Jenkins build timeout plugin
- Folders Plugin
- ** Credentials Plugin
- ** Structs Plugin
- ** Pipeline: Step API
- ** Plain Credentials Plugin
- Credentials Binding Plugin
- Email Extension Plugin
- ** SSH Credentials Plugin
- ** Jenkins Git client plugin
- ** SCM API Plugin

** - required dependency

Jenkins

Getting Started

Create First Admin User

Usuario:

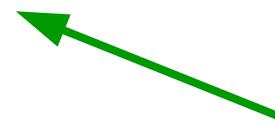
Contraseña:

Confirma la contraseña:

Nombre completo:

Dirección de email:

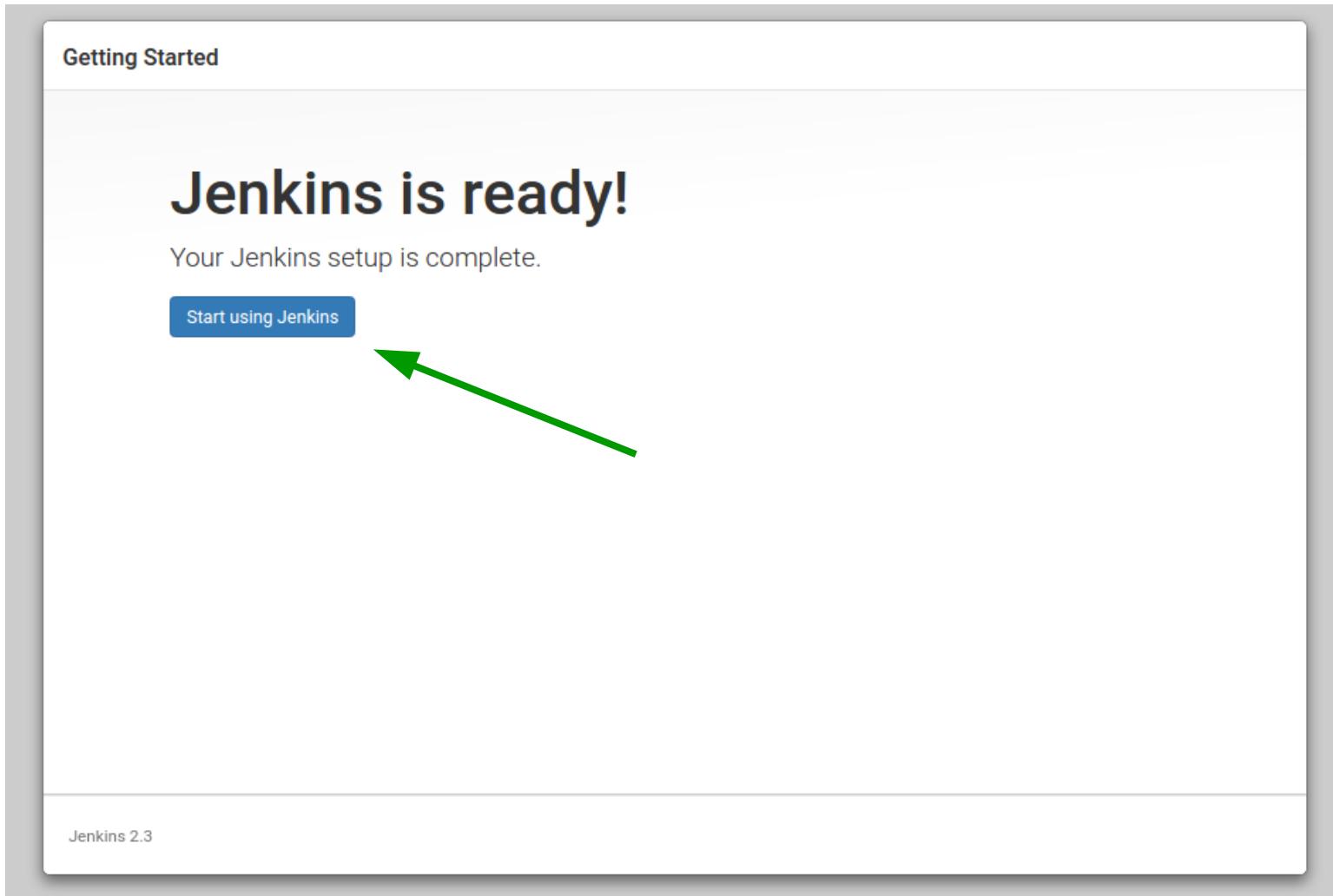
Configuramos la cuenta de administrador





Jenkins 2.3 Continue as admin Save and Finish

Jenkins



Jenkins

- **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

- **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

- **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



The screenshot shows the Jenkins dashboard at localhost:8081. The main heading is "¡Bienvenido a Jenkins!" with the sub-instruction "Por favor, [crea una nueva tarea](#) para empezar." A green callout box with a green arrow points to the "crea una nueva tarea" link, containing the text: "Creamos una nueva tarea para descargar el proyecto y ejecutar los tests".

Micael

Panel de control [Jenkins] ×

localhost:8081

Jenkins

búsqueda

Admin | Desconectar

Nueva Tarea

Personas

Historial de trabajos

Administrar Jenkins

Mis vistas

Credentials

Trabajos en la cola

No hay trabajos en la cola

Estado del ejecutor de construcciones

1 Inactivo

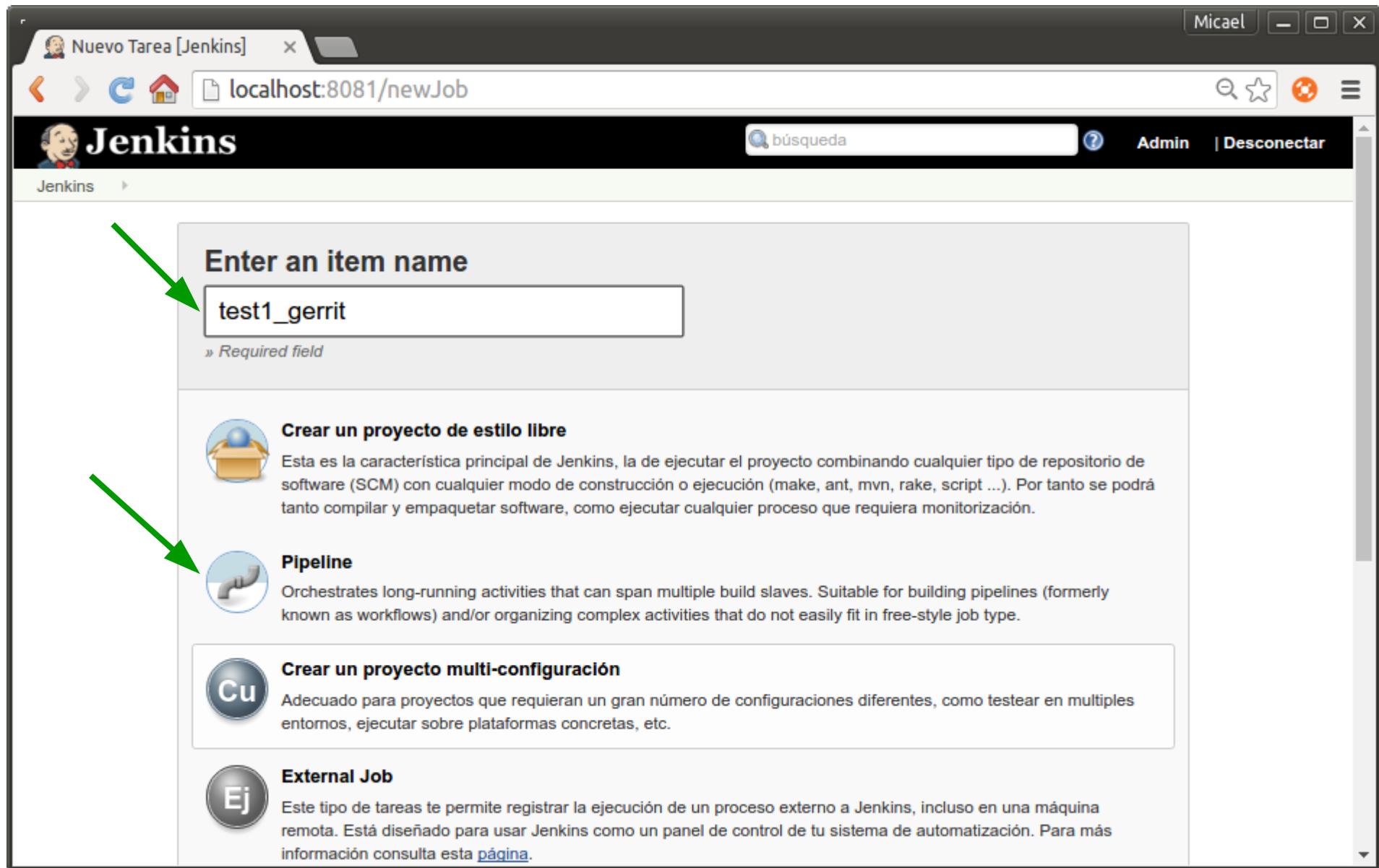
2 Inactivo

Añadir descripción

ACTIVAR AUTO REFRESCO

Página generada: 12-may-2016 1:24:14 CEST REST API Jenkins ver. 2.3

Jenkins



Nuevo Tarea [Jenkins] x

localhost:8081/newJob

Jenkins

Micael

búsqueda Admin | Desconectar

Jenkins

Enter an item name

test1_gerrit

» Required field

Crear un proyecto de estilo libre

Esta es la característica principal de Jenkins, la de ejecutar el proyecto combinando cualquier tipo de repositorio de software (SCM) con cualquier modo de construcción o ejecución (make, ant, mvn, rake, script ...). Por tanto se podrá tanto compilar y empaquetar software, como ejecutar cualquier proceso que requiera monitorización.

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.

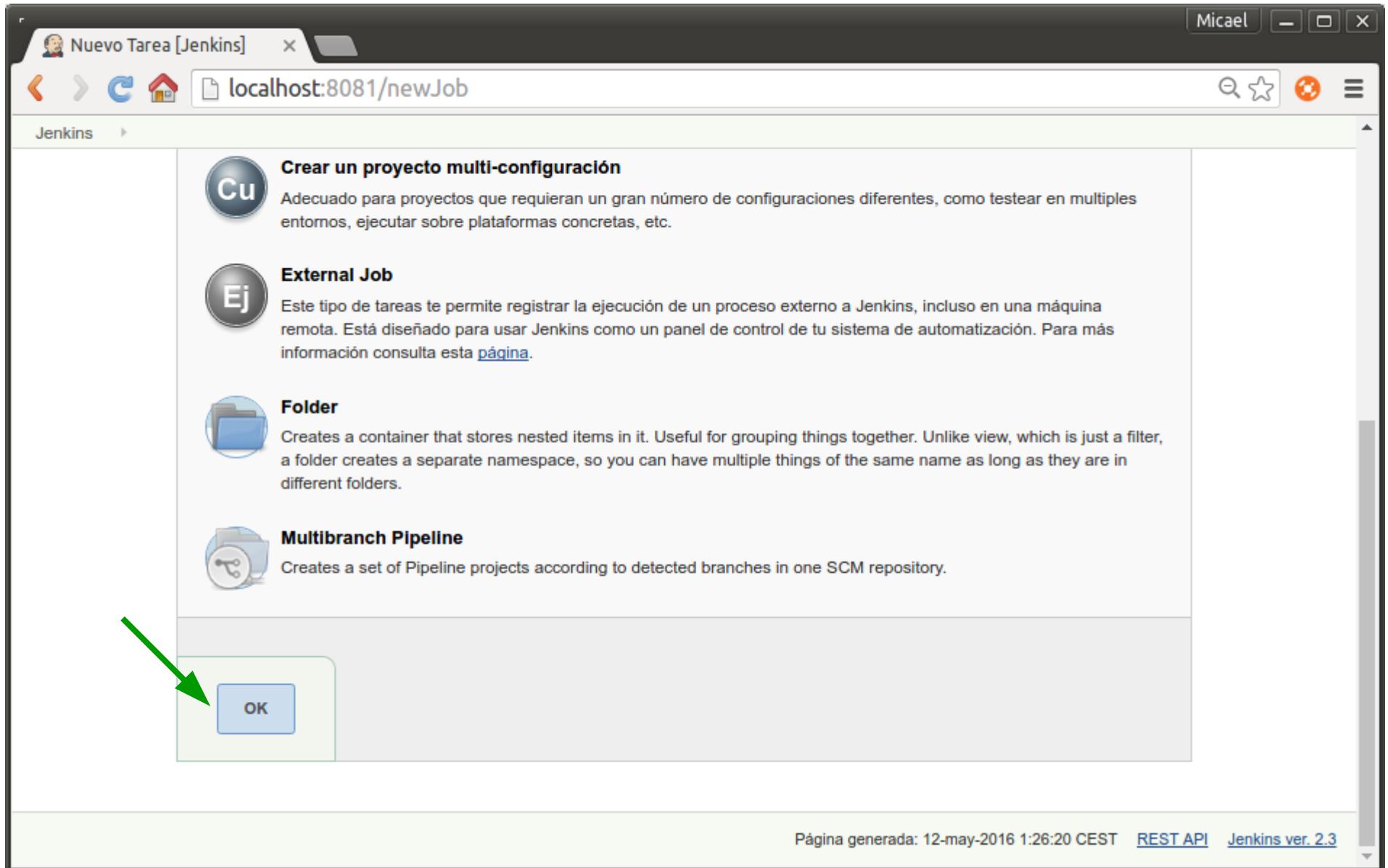
Crear un proyecto multi- configuración

Adecuado para proyectos que requieran un gran número de configuraciones diferentes, como testear en múltiples entornos, ejecutar sobre plataformas concretas, etc.

External Job

Este tipo de tareas te permite registrar la ejecución de un proceso externo a Jenkins, incluso en una máquina remota. Está diseñado para usar Jenkins como un panel de control de tu sistema de automatización. Para más información consulta esta [página](#).

Jenkins



Nuevo Tarea [Jenkins] x

Micael

localhost:8081/newJob

Jenkins

Crear un proyecto multi-configuración
 Cu
Adecuado para proyectos que requieran un gran número de configuraciones diferentes, como testear en múltiples entornos, ejecutar sobre plataformas concretas, etc.

External Job
 Ej
Este tipo de tareas te permite registrar la ejecución de un proceso externo a Jenkins, incluso en una máquina remota. Está diseñado para usar Jenkins como un panel de control de tu sistema de automatización. Para más información consulta esta [página](#).

Folder
 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.

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

OK

Página generada: 12-may-2016 1:26:20 CEST REST API Jenkins ver. 2.3

Jenkins

Pipeline

Definition

Pipeline script

Script

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

Use Groovy Sandbox

[Pipeline Syntax](#)

Guardar

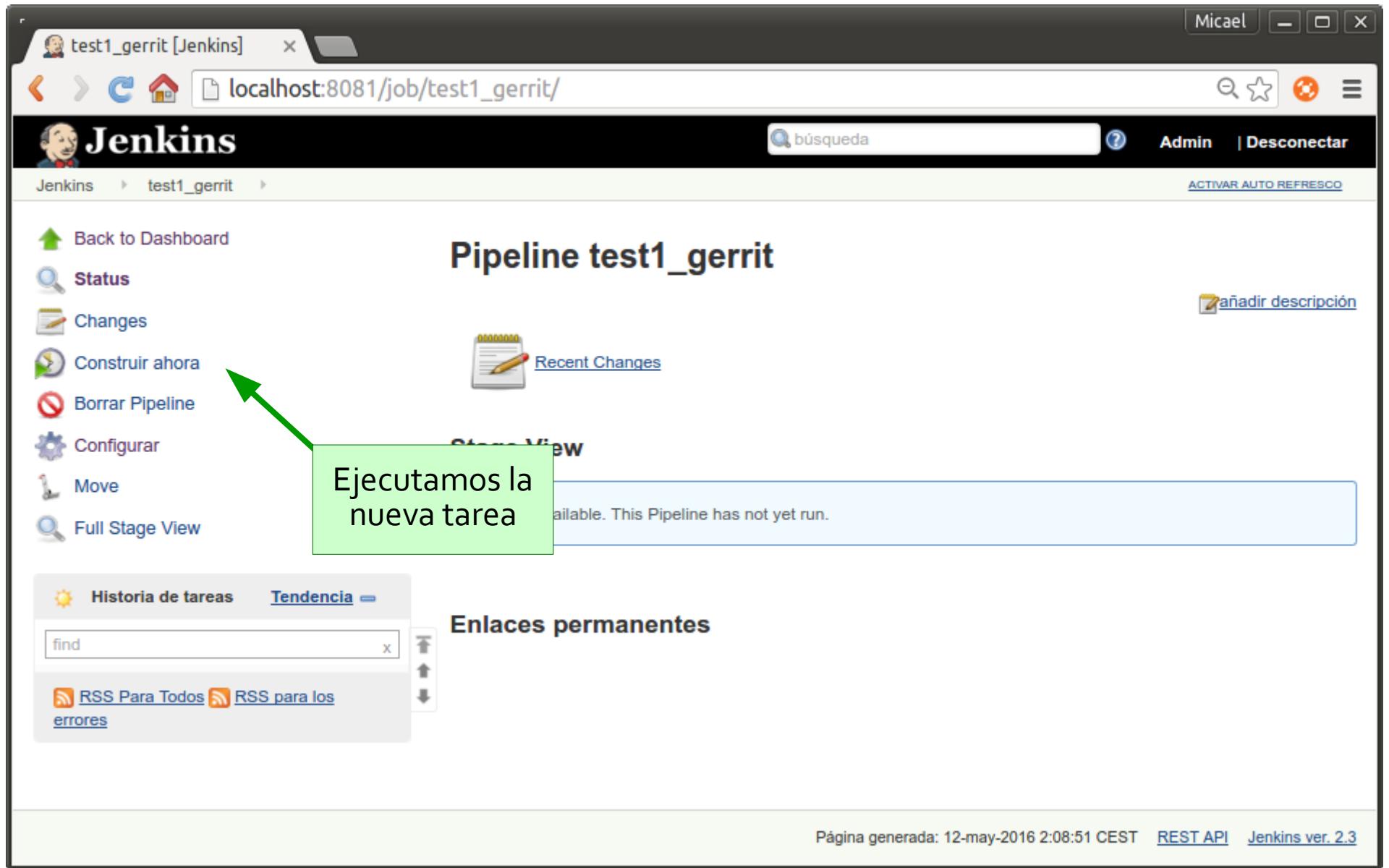
Apply

Indicamos el código de
nuestro job
(ver la siguiente slide)

Jenkins

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

Jenkins



A screenshot of a Jenkins pipeline job named "test1_gerrit". The browser title bar shows "test1_gerrit [Jenkins]". The URL in the address bar is "localhost:8081/job/test1_gerrit/". The Jenkins header includes the Jenkins logo, a search bar, and user links for "Micael", "Admin", and "Desconectar". The left sidebar has a "Back to Dashboard" link and several other options: "Status", "Changes", "Construir ahora", "Borrar Pipeline", "Configurar", "Move", and "Full Stage View". A green callout box with the text "Ejecutamos la nueva tarea" has a green arrow pointing to the "Configurar" link in the sidebar. The main content area is titled "Pipeline test1_gerrit" and contains a "Recent Changes" section with a "Recent View" button. Below it is a message stating "available. This Pipeline has not yet run." At the bottom, there's a "Historia de tareas" section with a "find" input field and RSS links for "RSS Para Todos" and "RSS para los errores". The footer includes page generation information ("Página generada: 12-may-2016 2:08:51 CEST") and links to "REST API" and "Jenkins ver. 2.3".

Ejecutamos la
nueva tarea

Página generada: 12-may-2016 2:08:51 CEST REST API Jenkins ver. 2.3

Jenkins



The screenshot shows the Jenkins interface for the pipeline job `test1_gerrit`. The left sidebar contains links like Back to Dashboard, Status, Changes, Construir ahora, Borrar Pipeline, Configurar, Move, and Full Stage View. The main area displays the `Stage View` for the `Pipeline test1_gerrit`. A green callout box with the text "Comienza la fase de descarga del repositorio" has a green arrow pointing to the `Checkout` stage, which is currently running and has an average time of 5 seconds. The stage view also shows a recent change from May 12 at 02:16 with "No Changes". At the bottom, there's a section for permanent links and RSS feeds for errors.

test1_gerrit [Jenkins] × Micael

localhost:8081/job/test1_gerrit/

Jenkins

Búsqueda Admin | Desconectar 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

Recent Changes

Stage View

Average stage times:

#1 May 12 02:16 No Changes

Checkout

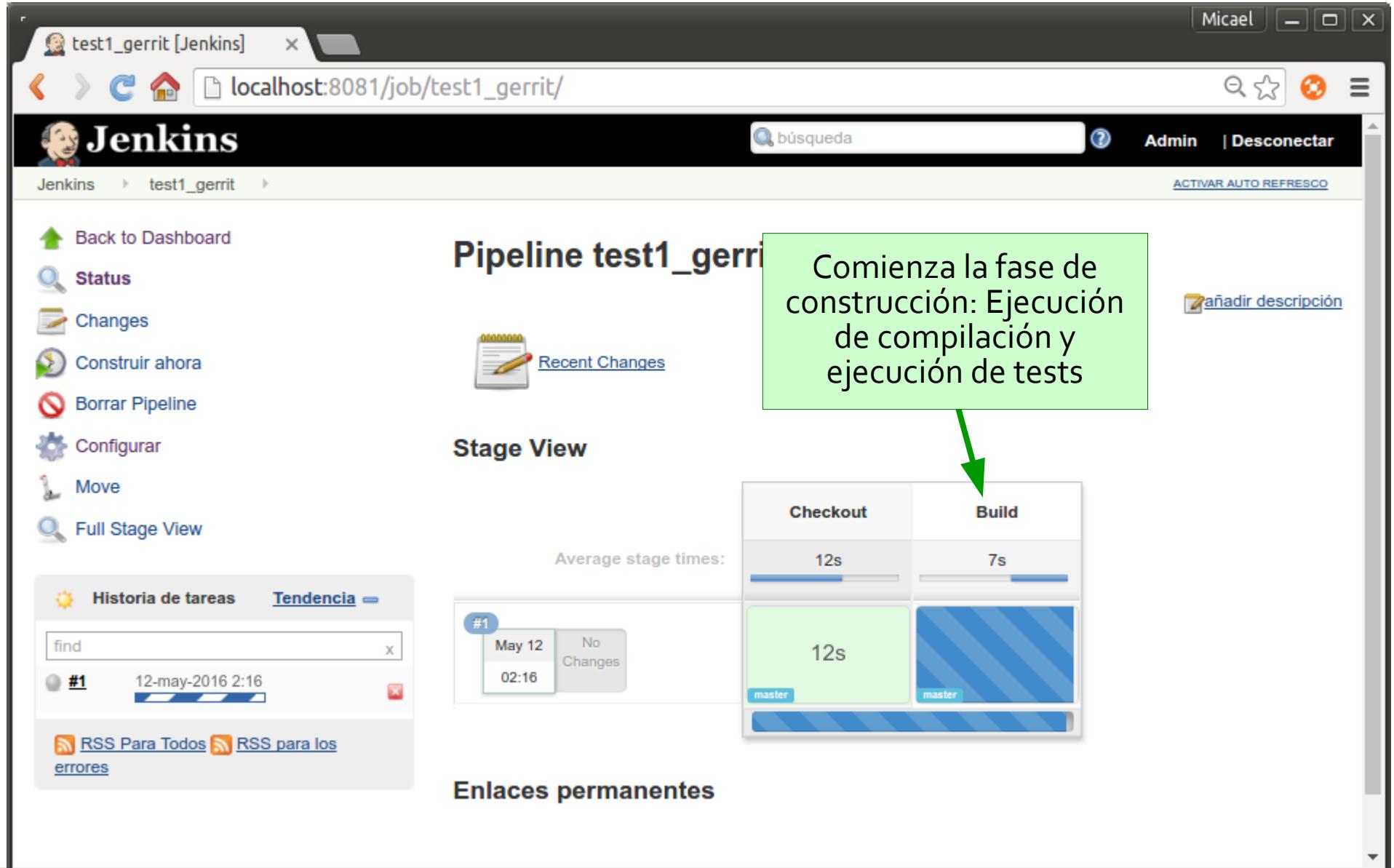
5s

master

Comienza la fase de descarga del repositorio

Enlaces permanentes

Jenkins



The screenshot shows the Jenkins Pipeline interface for the job "test1_gerrit". The pipeline has two stages: "Checkout" and "Build". The "Build" stage took 7 seconds and is shown with a blue diagonal striped background. A green callout box with a black arrow points to the "Build" stage, containing the text: "Comienza la fase de construcción: Ejecución de compilación y ejecución de tests".

Pipeline test1_gerrit

Stage View

Checkout	Build
12s	7s

Average stage times:

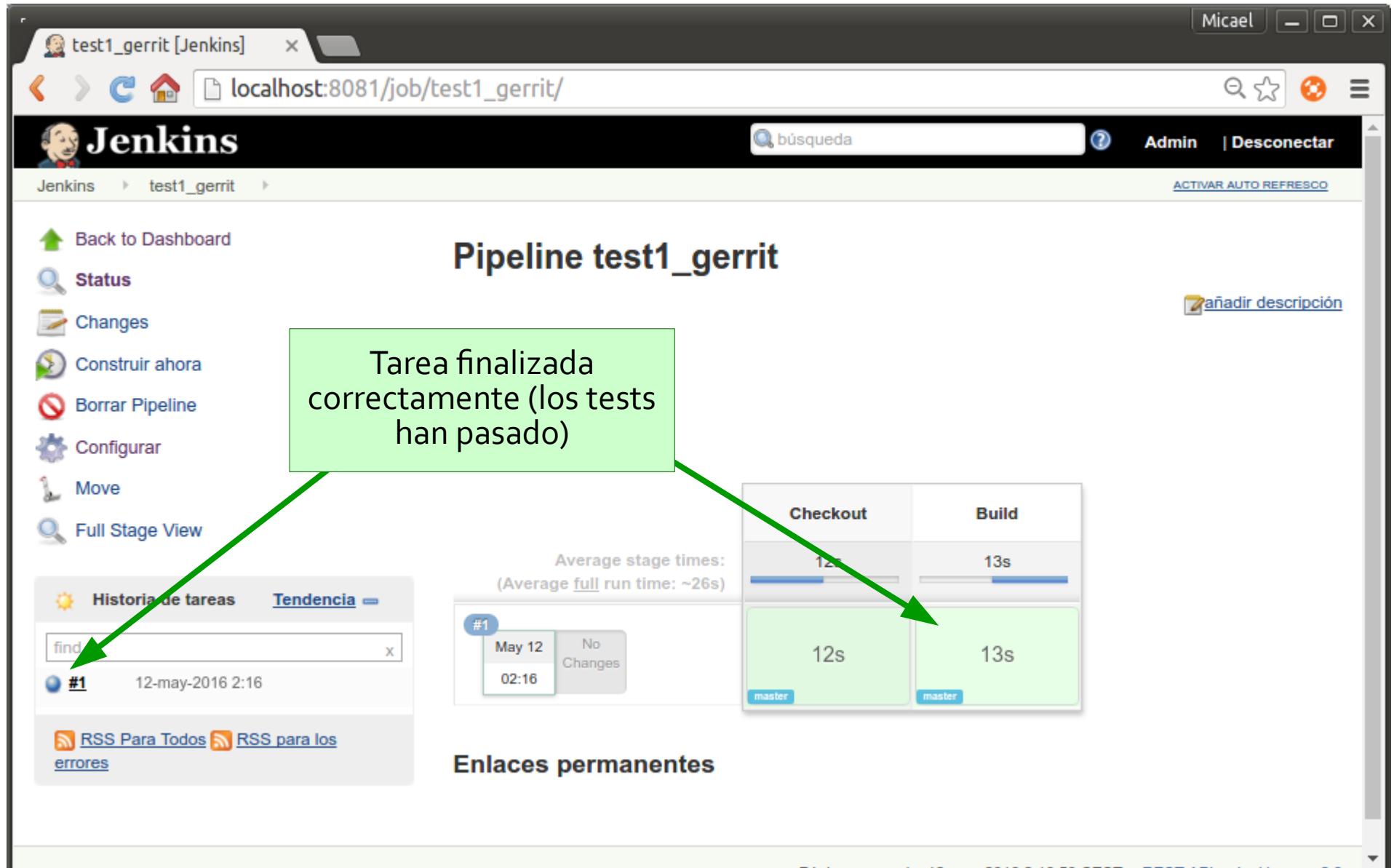
#1 May 12 02:16 No Changes

Enlaces permanentes

Historia de tareas Tendencia

find #1 12-may-2016 2:16 RSS Para Todos RSS para los errores

Jenkins



A screenshot of a Jenkins pipeline job named "test1_gerrit". The pipeline has two stages: "Checkout" and "Build". The "Checkout" stage took 12s and the "Build" stage took 13s, both running on the "master" node. A green callout box highlights the message "Tarea finalizada correctamente (los tests han pasado)". A green arrow points from the "Historia de tareas" section to the "Checkout" stage, and another green arrow points from the "Checkout" stage to the "Build" stage.

Pipeline test1_gerrit

Tarea finalizada correctamente (los tests han pasado)

Average stage times:
(Average full run time: ~26s)

Checkout	Build
12s	13s

Checkout Build

Average stage times:
(Average full run time: ~26s)

#1	May 12	No Changes
	02:16	

Historia de tareas Tendencia

find #1 12-may-2016 2:16

RSS Para Todos RSS para los errores

Enlaces permanentes

Jenkins

S ShellCommandJob x localhost:8081/job/ShellCommandJob/ Jenkins Admin | Desconectar

localhost:8081/job/ShellCommandJob/ Other bookmarks

Gastos investiga AWS Educate Projects - Resear Centro Universit Hangouts de Go URJC - Reserva d Papers Tareas Patxi/Mic

Jenkins Jenkins ShellCommandJob ACTIVAR AUTO REFRESCO

Back to Dashboard Status Changes Construir ahora Borrar Pipeline Configurar Full Stage View Pipeline Syntax

añadir descripción Desactivar el Proyecto

Pipeline ShellCommandJob

Last Successful Artifacts

Podemos ver los logs de cada etapa

Average stage times: (Average full run time: ~5s)

#	Date	Time	Changes	Preparation	Build	Results
#1	Nov 23	11:15	No Changes	1s	Success Logs 3s	88ms

Últimos resultados de tests (Sin fallas)

36

Jenkins

ShellCommandJob x

localhost:8081/job/ShellCommandJob/

Gastos investiga AWS Educate Projects - Resear Centro Universit Hangouts de Go URJC - Reserva Papers Tareas Patxi/Mic Other bookmarks

Admin | Desconectar ACTIVAR AUTO REFRESCO

Añadir descripción Desactivar el Proyecto

Jenkins

ShellCommandJob

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 E

Stage Logs (Build)

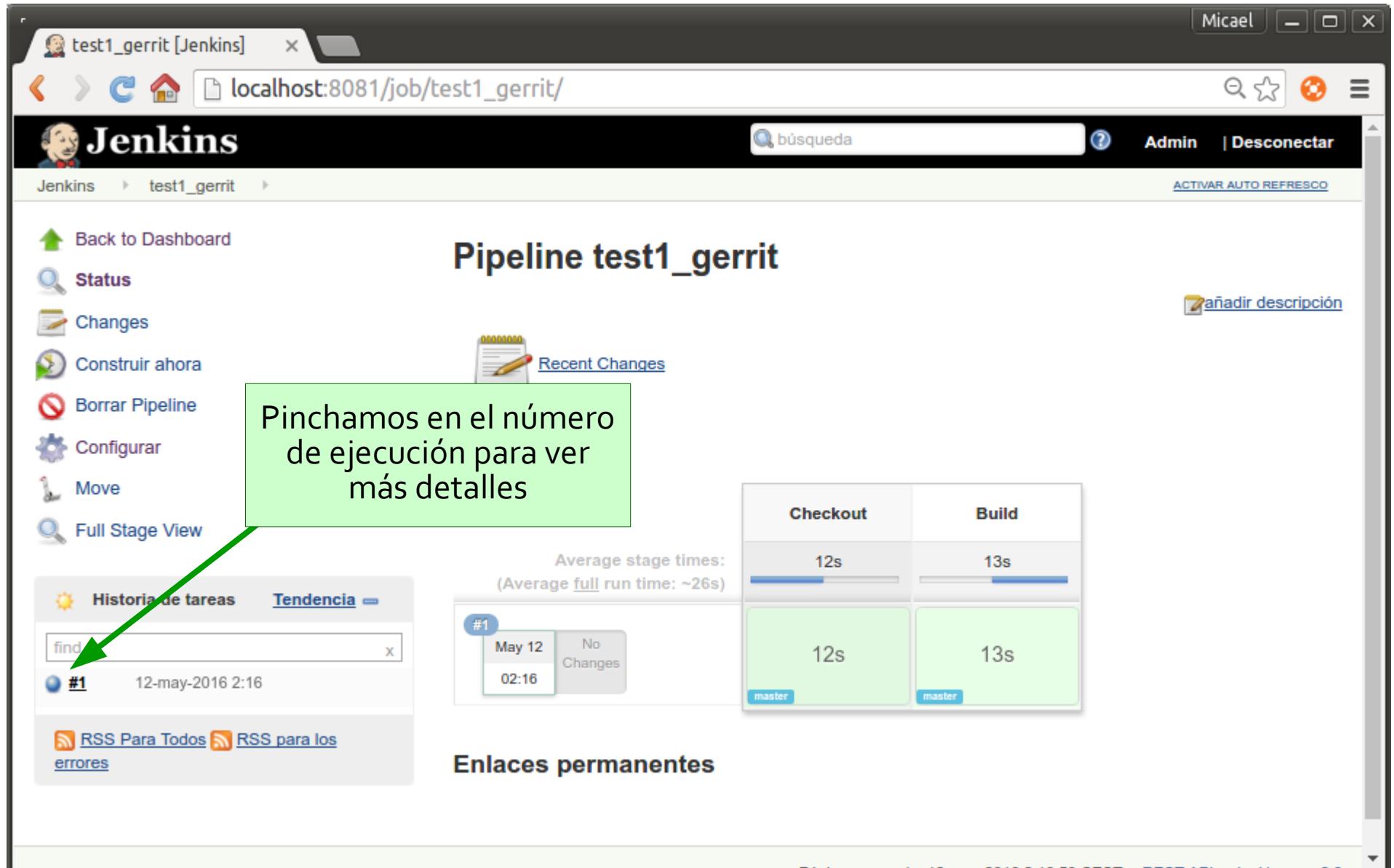
Checks if running on a Unix-like node (self time 1ms)

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



A screenshot of a Jenkins pipeline interface titled "Pipeline test1_gerrit". The left sidebar shows navigation options like "Back to Dashboard", "Status", "Changes", "Construir ahora", "Borrar Pipeline", "Configurar", "Move", and "Full Stage View". A green callout box points to the "#1" link in the "Historia de tareas" section, with the text: "Pinchamos en el número de ejecución para ver más detalles". The main area displays pipeline stages: "Checkout" (12s) and "Build" (13s), both completed successfully. Below the stages, a summary states: "Average stage times: (Average full run time: ~26s)". A "Recent Changes" section shows a single entry from May 12 at 02:16 with "No Changes". At the bottom, there's a "RSS Para Todos" feed and a "Enlaces permanentes" section.

Micael

localhost:8081/job/test1_gerrit/

Jenkins

búsqueda

Admin | Desconectar

ACTIVAR AUTO REFRESCO

Back to Dashboard

Status

Changes

Construir ahora

Borrar Pipeline

Configurar

Move

Full Stage View

Historia de tareas Tendencia

#1 12-may-2016 2:16

RSS Para Todos RSS para los errores

Pinchamos en el número de ejecución para ver más detalles

Average stage times:
(Average full run time: ~26s)

Recent Changes

#1 May 12 02:16 No Changes

Checkout	Build
12s	13s
master	master

Checkout Build

12s 13s

master master

Enlaces permanentes

Jenkins



A screenshot of a web browser displaying a Jenkins job page. The title bar shows the URL as `localhost:8081/job/test1_gerrit/1`. The main content area shows the build details for "Build #1 (12-may-2016 2:16:54)". It includes information about the user who started the build ("Iniciado por el usuario Admin"), the git revision ("Revision: ba3319114ca1e0ac3ca629819df1e27d71fc5aef"), and the commit message ("refs/remotes/origin/master"). On the left sidebar, there is a list of actions: "Back to Project", "Status", "Changes", "Console Output", "Edit Build Information" (which has a green arrow pointing to a callout box), "Borrar Proyecto", "Git Build Data", "No Tags", "Replay", and "Pipeline Steps". A green callout box with the text "Podemos ver la salida del comando" is positioned over the "Edit Build Information" link.

Micael

localhost:8081/job/test1_gerrit/1

Jenkins

búsqueda

Admin | Desconectar

ACTIVAR AUTO REFRESCO

Started 11 Min ago
Took 27 Seg

añadir descripción

Iniciado por el usuario Admin

Revision: ba3319114ca1e0ac3ca629819df1e27d71fc5aef

- refs/remotes/origin/master

Back to Project

Status

Changes

Console Output

Edit Build Information

Borrar Proyecto

Git Build Data

No Tags

Replay

Pipeline Steps

Podemos ver la salida del comando

Página generada: 12-may-2016 2:28:26 CEST REST API Jenkins ver. 2.3

Jenkins



The screenshot shows a Jenkins job named "test1_gerrit" with build number #1. The console output is displayed, showing the build process. A green callout box on the left contains the following text:

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

A green arrow points from this callout box to the "Console Output" section of the Jenkins interface.

Salida de consola

```
Lanzada por el usuario Admin
[Pipeline] node
Running on principal in /home/mica/.jenkins/workspace/test1_gerrit
[Pipeline] {
[Pipeline] stage (Checkout)
Entering stage Checkout
Proceeding
[Pipeline] git
Cloning the remote Git repository
Cloning repository http://localhost:8080/p/test1
> git init /home/mica/.jenkins/workspace/test1_gerrit # timeout=10
Fetching upstream changes from http://localhost:8080/p/test1
> git --version # timeout=10
> git -c core.askpass=true fetch --tags --progress http://localhost:8080/p/test1
+refs/heads/*:refs/remotes/origin/*
> git config remote.origin.url http://localhost:8080/p/test1 # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/*
timeout=10
> git config remote.origin.url http://localhost:8080/p/test1 # timeout=10
Fetching upstream changes from http://localhost:8080/p/test1
> git -c core.askpass=true fetch --tags --progress http://localhost:8080/p/test1
+refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
```

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

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") {  
            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'  
                }  
            }  
        }  
    }  
}
```

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 "**/target/surefire-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/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'  
    }  
}
```

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

Pipeline

Declarativa

```
pipeline {  
    tools {  
        maven "M3"  
    }  
    agent any  
    stages {  
        stage("Preparation") {  
            steps {  
                git 'https://github.com/jglick/...'  
            }  
        }  
        stage("Build") {  
            steps {  
                sh "mvn ..." // Step inside a stage  
            }  
        }  
    }  
    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" ...) // Step inside a stage  
        }  
    }  
    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/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'  
    }  
}
```

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/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'  
    }  
}
```

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

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

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

Hay que desmarcar el
sandbox
(no se permite
código)

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

Pipeline



Build #3 (23-nov-2017 17:01:45)

Los tres ficheros se archivan



[Build Artifacts](#)

	one.txt	4 B		view
	three.txt	6 B		view
	two.txt	4 B		view



Iniciado por el usuario [Admin](#)



Revision: fd2af2fab4580dc9893f6e71f967b9be32e98e49

- refs/remotes/origin/master

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 from the **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 statement or just 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!

Character encoding

utf-8

Especificamos los
parámetros

Generate Pipeline Script

`writeFile encoding: 'utf-8', file: 'file.txt', text: 'Hello, world!'`

Obtenemos el texto
que hay que copiar
en el pipeline

Pipeline

 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

- ## 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 ejecución 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

- **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.displayName}",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

- ## 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

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

Screenshot of the Jenkins Pipeline ejem2 job interface:

The top navigation bar shows the job name "ejem2 [Jenkins]" and the URL "localhost:8081/job/ejem2/". The pipeline status is "1" (green). The search bar contains "search". The top right has "admin" and "log out" buttons, and an "ENABLE AUTO REFRESH" link.

The left sidebar contains the following links:

- Back to Dashboard
- Status
- Changes
- Build Now
- Delete Pipeline
- Configure
- Full Stage View
- Rename
- Pipeline Syntax

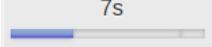
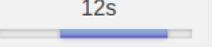
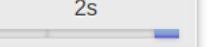
The main content area displays the pipeline details:

Pipeline ejem2

 Recent Changes

Stage View

Average stage times:

Preparation	Download libs	Test
7s	12s	2s
		

Recent build information:

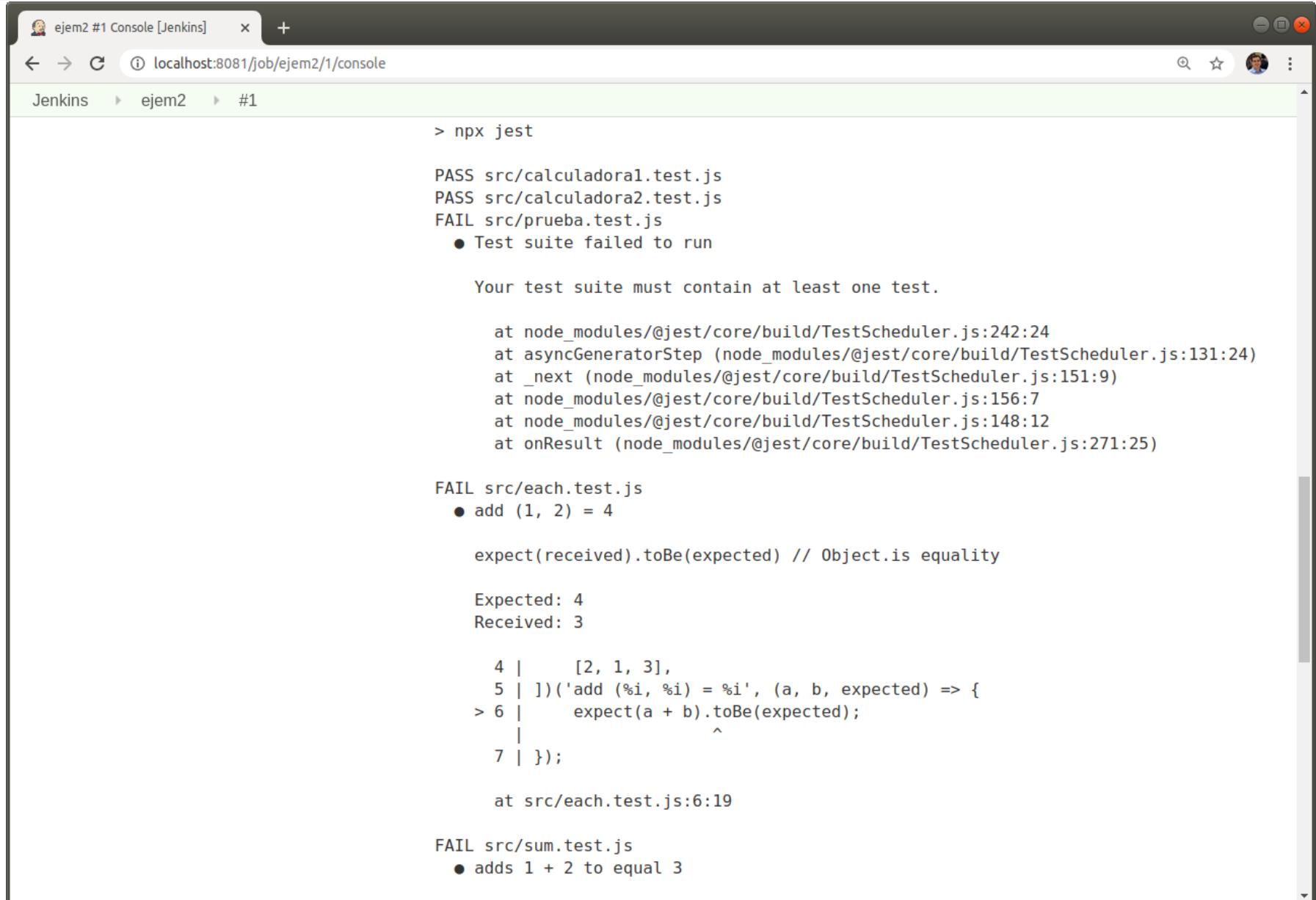
- #1 Jan 16 05:57:43 CET 2020 (No Changes)

Permalinks:

- [Last build \(#1\), 6 min 24 sec ago](#)
- [Last failed build \(#1\), 6 min 24 sec ago](#)
- [Last unsuccessful build \(#1\), 6 min 24 sec ago](#)
- [Last completed build \(#1\), 6 min 24 sec ago](#)

Buttons on the right side:

- Add description
- Disable Project



A screenshot of a Jenkins job's console output window. The title bar shows "ejem2 #1 Console [Jenkins]". The URL in the address bar is "localhost:8081/job/ejem2/1/console". The Jenkins navigation bar shows "Jenkins > ejem2 > #1". The console output displays the results of a Jest test run:

```
> npx jest

PASS src/calculadora1.test.js
PASS src/calculadora2.test.js
FAIL src/prueba.test.js
  ● Test suite failed to run

    Your test suite must contain at least one test.

      at node_modules/@jest/core/build/TestScheduler.js:242:24
      at asyncGeneratorStep (node_modules/@jest/core/build/TestScheduler.js:131:24)
      at _next (node_modules/@jest/core/build/TestScheduler.js:151:9)
      at node_modules/@jest/core/build/TestScheduler.js:156:7
      at node_modules/@jest/core/build/TestScheduler.js:148:12
      at onResult (node_modules/@jest/core/build/TestScheduler.js:271:25)

FAIL src/each.test.js
  ● add (1, 2) = 4

    expect(received).toBe(expected) // Object.is equality

      Expected: 4
      Received: 3

        4 |     [2, 1, 3],
        5 |   ])(`add (%i, %i) = %i`, (a, b, expected) => {
> 6 |     expect(a + b).toBe(expected);
          ^
        |
        7 | });

      at src/each.test.js:6:19

FAIL src/sum.test.js
  ● adds 1 + 2 to equal 3
```

Jenkins

- **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**

Jenkins

- **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

- **Ejemplo 3**

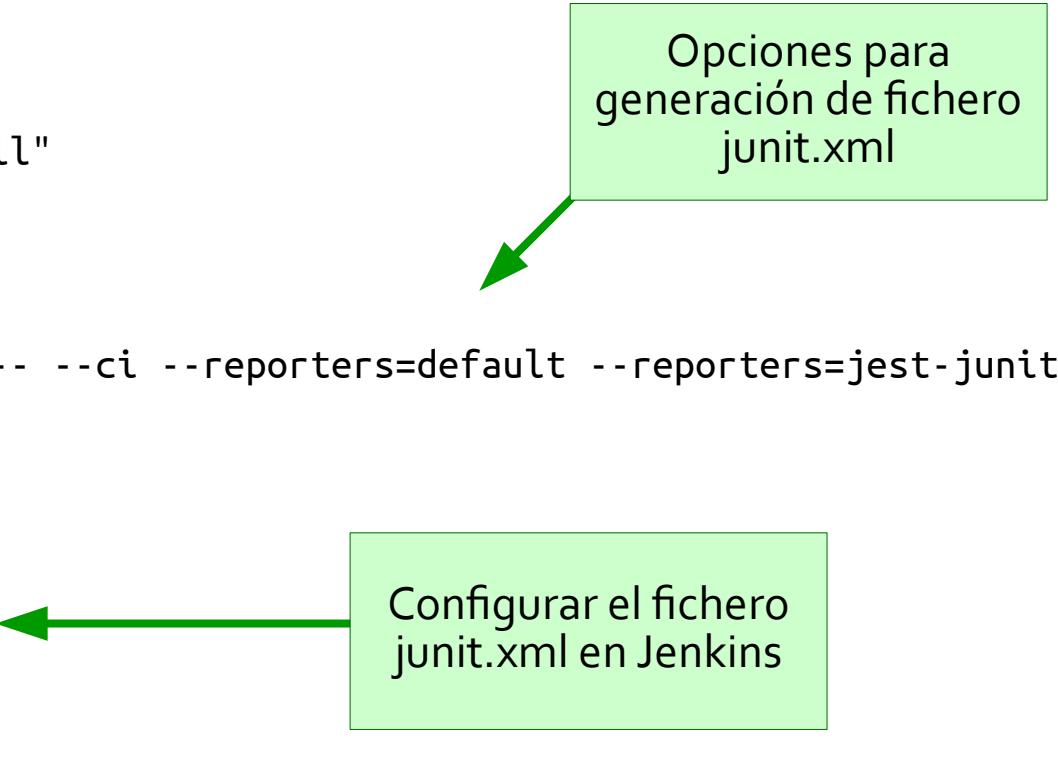
- Crearemos un nuevo job para ejecutar los tests del proyecto **tema13/ejem3** del repositorio
<https://github.com/codeurjc/curso-javascript-2019>
- Es el mismo proyecto que el ejem2, pero con la dependencia **junit-jest**
- Usando los comandos adecuados, podremos ver los tests fallidos en la **web de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

Jenkins

```
pipeline {
    agent any
    stages {
        stage("Preparation") {
            steps {
                git 'https://github.com/codeurjc/curso-javascript-2019.git'
            }
        }
        stage("Download 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("Download 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"
        }
    }
}
```



ejem3 #6 [Jenkins] +
localhost:8081/job/ejem3/6/

Jenkins

1 search admin | log out

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

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](#)

Resultados fallidos de los tests

Page generated: Thu Jan 16 06:22:10 CET 2020 [REST API](#) [Jenkins ver. 2.204.1](#)

The screenshot shows the Jenkins Test Results page for a build named ejem3 #6. The main header displays "Test Result" with "3 failures (+3)" highlighted in red. Below this, a table lists three failed test cases:

Test Name	Duration	Age
+ add(1, 2) = 4, add(1, 2) = 4	3 ms	1
+ adds 1 + 2 to equal 3, adds 1 + 2 to equal 3	2 ms	1
+ force fail, force fail	0 ms	1

A green arrow points from the text "Se muestran los tests fallidos" to the "All Failed Tests" section. The footer of the page includes the message "Page generated: Thu Jan 16 06:24:31 CET 2020 REST API Jenkins ver. 2.204.1".

Test Result

3 failures (+3)

10 tests (+8)
Took 0.58 sec.
[add description](#)

All Failed Tests

Test Name	Duration	Age
+ add(1, 2) = 4, add(1, 2) = 4	3 ms	1
+ adds 1 + 2 to equal 3, adds 1 + 2 to equal 3	2 ms	1
+ force fail, force fail	0 ms	1

All Tests

Package	Duration	Fail (diff)	Skip (diff)
(root)	10 ms	3 +3	0

Se muestran los tests fallidos

Page generated: Thu Jan 16 06:24:31 CET 2020 REST API Jenkins ver. 2.204.1

ejem3 #6 test - adds 1 + 2 to equal 3

localhost:8081/job/ejem3/6//testReport/junit/(root)/%20adds%201%20+%202%20to%20equal%203/_adds_1__2_to_equal_3/

Jenkins

1 search admin | log out

Jenkins ejem3 #6 Test Results Test Results (root) adds 1 + 2 to equal 3 adds 1 + 2 to equal 3 ENABLE AUTO REFRESH

[History](#) [Git Build Data](#) [No Tags](#) [Test Result](#) [Restart from Stage](#) [Replay](#) [Pipeline Steps](#) [Workspaces](#) [Previous Build](#)

Failed

adds 1 + 2 to equal 3. adds 1 + 2 to equal 3 (from undefined)

Failing for the past 1 build (Since #6) Took 2 ms. [add description](#)

Stacktrace

```
Error: expect(received).toBeTruthy()

Received: false
    at Object.<anonymous>
(/home/mica/.jenkins/workspace/ejem3/tema13/ejem3/src/sum.test.js:12:29)
    at Object.asyncJestTest
(/home/mica/.jenkins/workspace/ejem3/tema13/ejem3/node_modules/jest-jasmine2/build/jasmineAsyncInstall.js:102:37)
    at /home/mica/.jenkins/workspace/ejem3/tema13/ejem3/node_modules/jest-jasmine2/build/queueRunner.js:43:12
    at new Promise (<anonymous>)
    at mapper (/home/mica/.jenkins/workspace/ejem3/tema13/ejem3/node_modules/jest-jasmine2/build/queueRunner.js:26:19)
    at /home/mica/.jenkins/workspace/ejem3/tema13/ejem3/node_modules/jest-jasmine2/build/queueRunner.js:73:41
    at processTicksAndRejections (internal/process/task_queues.js:97:5)
```

Page generated: Thu Jan 16 06:25:32 CET 2020 [REST API](#) [Jenkins ver. 2.204.1](#)

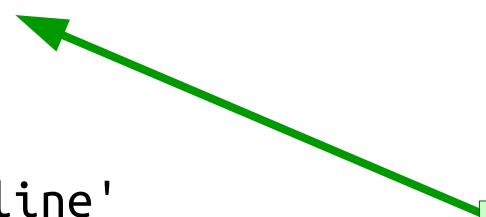
Jenkins

- **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

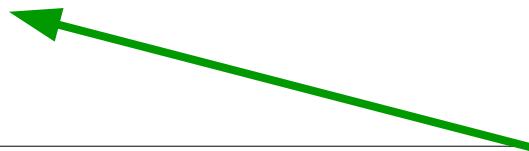
```
node {  
    stage('First sample') {  
        Build job: 'StepSyntaxArchiveArtifacts',  
        propagate: false  
    }  
  
    stage('Second sample') {  
        build 'DeclarativePipeline'  
    }  
}
```



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

```
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')]
            }
        )
    }
}
```

Ejecución de Jobs en paralelo

Ojo con los paréntesis