Getting Started with Pipeline

Jenkins Pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins. Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline DSL. [7: Domain-Specific Language]

This section introduces some of the key concepts to Jenkins Pipeline and help introduce the basics of defining and working with Pipelines inside of a running Jenkins instance.

Prerequisites

To use Jenkins Pipeline, you will need:

- Jenkins 2.x or later (older versions back to 1.642.3 may work but are not recommended)
- Pipeline plugin [8: Pipeline plugin]

To learn how to install and manage plugins, consult Managing Plugins.

Defining a Pipeline

Scripted Pipeline is written in Groovy. The relevant bits of Groovy syntax will be introduced as necessary in this document, so while an understanding of Groovy is helpful, it is not required to work with Pipeline.

A basic Pipeline can be created in either of the following ways:

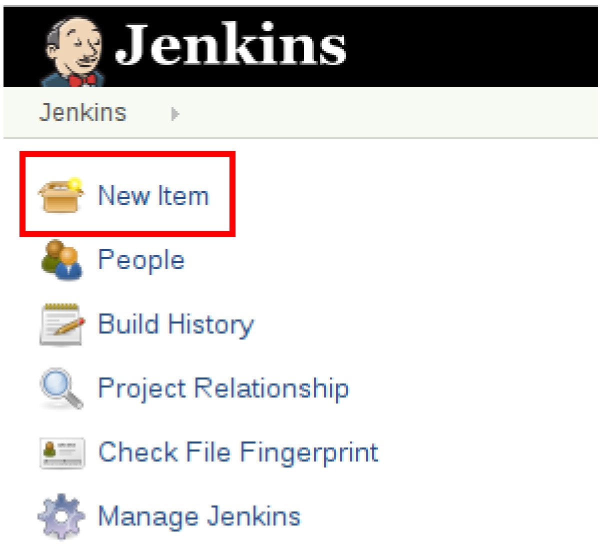
- By entering a script directly in the Jenkins web UI.
- By creating a `Jenkinsfile` which can be checked into a project's source control repository.

The syntax for defining a Pipeline with either approach is the same, but while Jenkins supports entering Pipeline directly into the web UI, it's generally considered best practice to define the Pipeline in a `Jenkinsfile` which Jenkins will then load directly from source control. [9: en.wikipedia.org/wiki/Source_control_management]

Defining a Pipeline in the Web UI

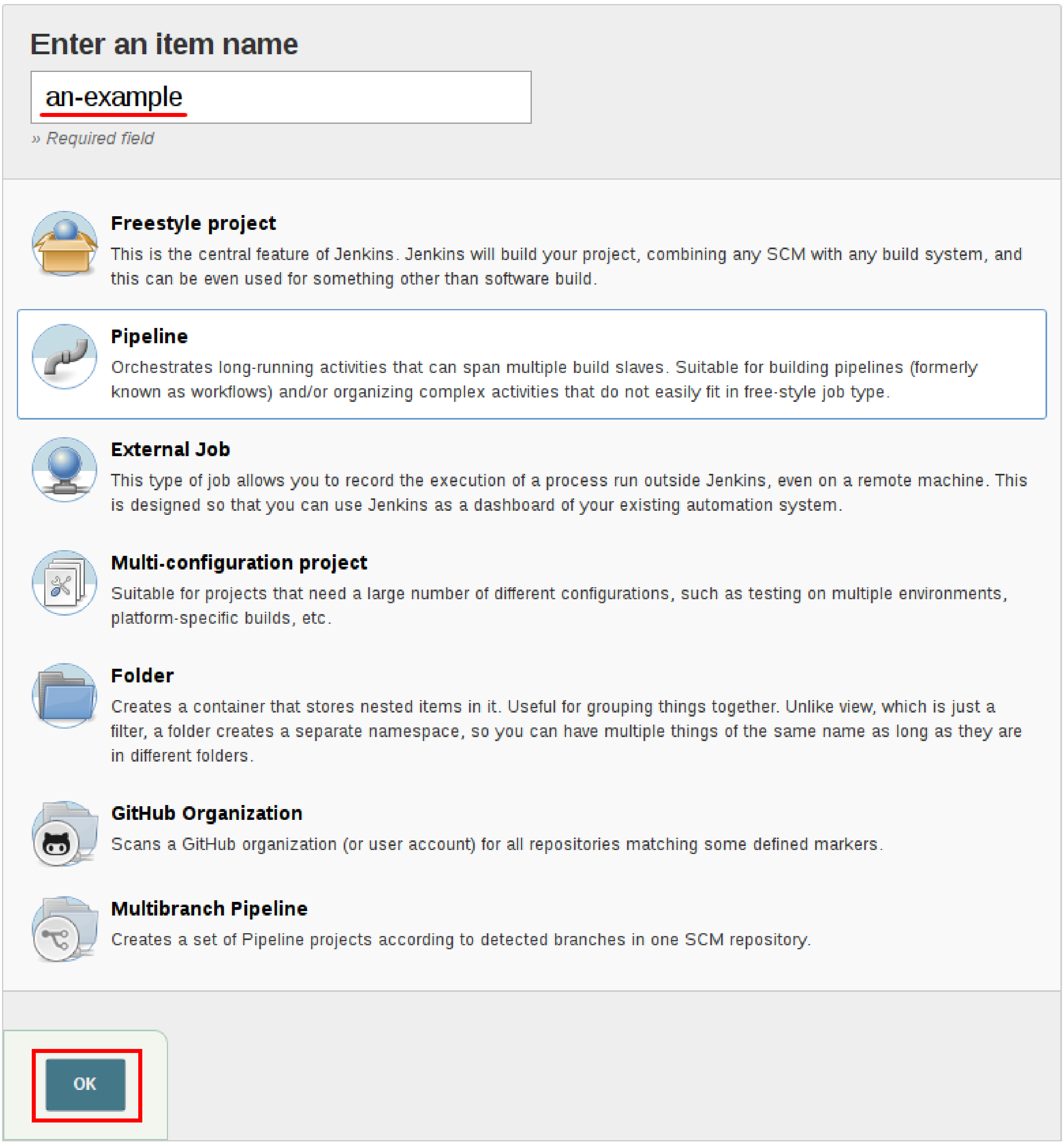
To create a basic Pipeline in the Jenkins web UI, follow these steps:

- Click **New Item** on Jenkins home page.



- Enter a name for your Pipeline, select **Pipeline** and click **OK**.

CAUTION Jenkins uses the name of the Pipeline to create directories on disk. Pipeline names which include spaces may uncover bugs in scripts which do not expect paths to contain spaces.


Enter an item name

an-example

» Required field

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

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.

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

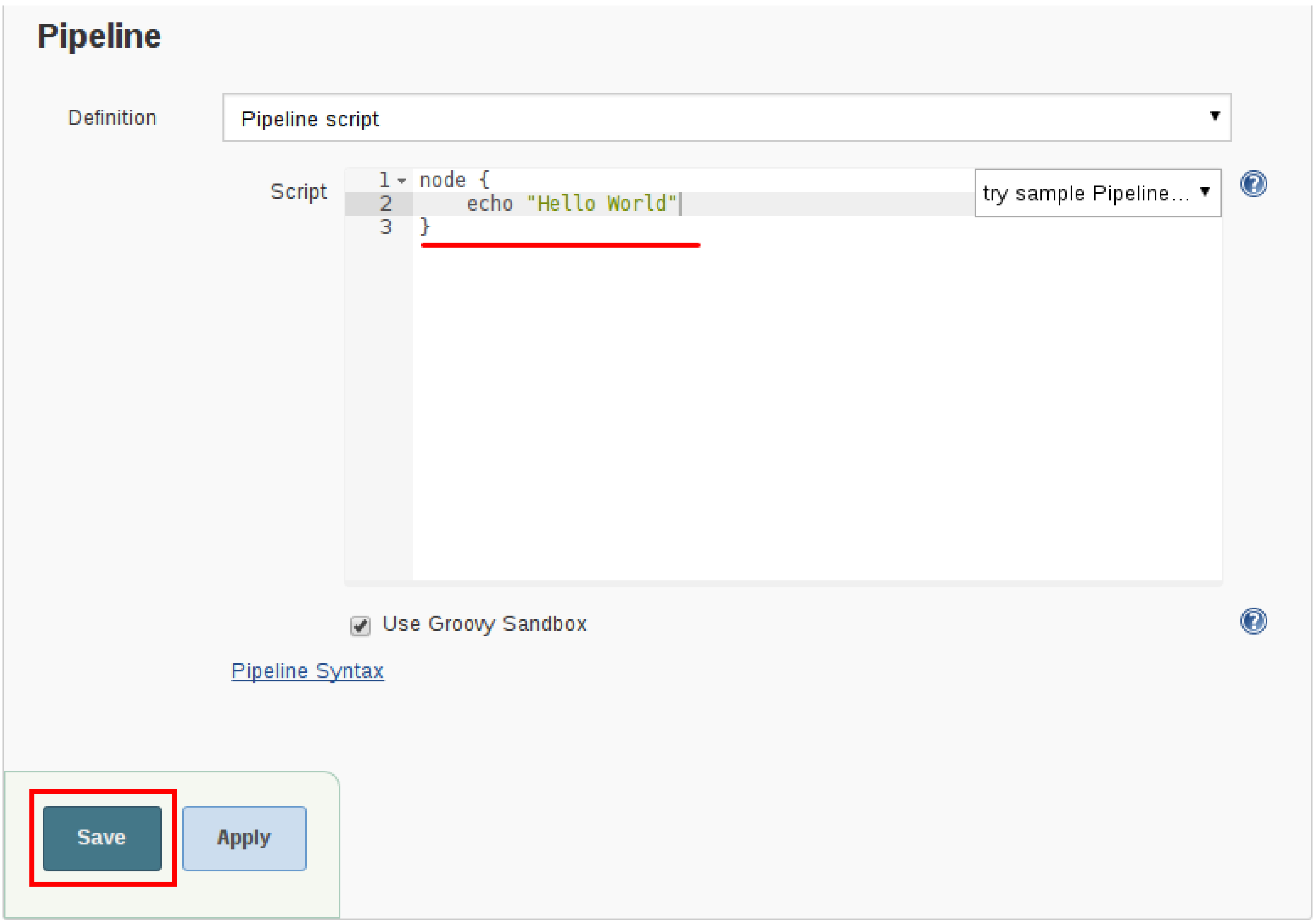
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

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

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

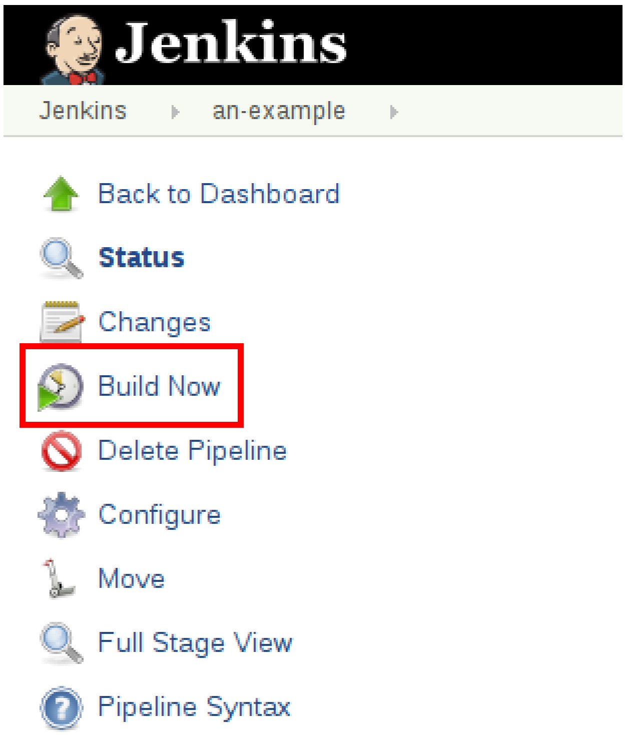
Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

- In the **Script** text area, enter a Pipeline and click **Save**.

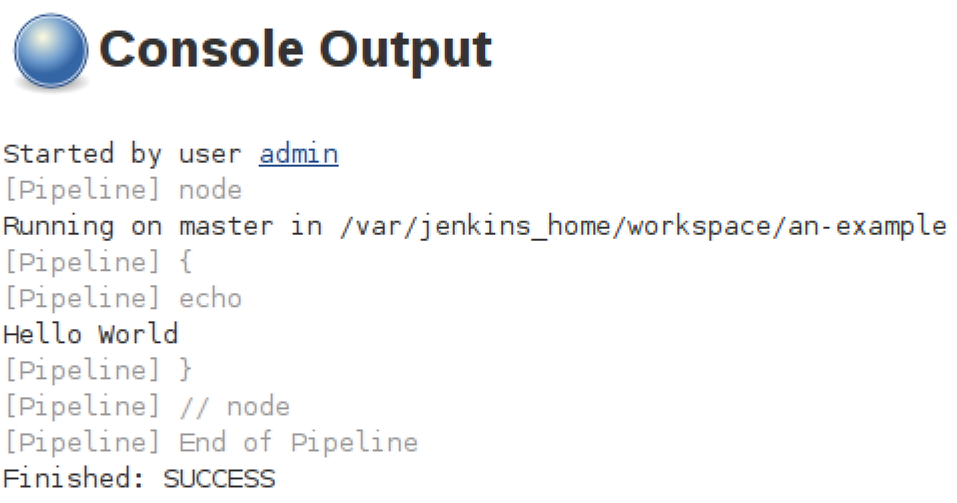


```
node {
    echo "Hello World"
}
```

- Click **Build Now** to run the Pipeline.



- Click **#1** under "Build History" and then click **Console Output** to see the full output from the Pipeline.



```
Started by user admin
[Pipeline] node
Running on master in /var/jenkins_home/workspace/an-example
[Pipeline] {
[Pipeline] echo
Hello World
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

The example above shows a successful run of a basic Pipeline created in the Jenkins web UI, using two steps.

```
// Script //
node { ①
    echo 'Hello World' ②
}
// Declarative not yet implemented //
```

① `node` allocates an executor and workspace in the Jenkins environment.

② `echo` writes simple string in the Console Output.

Defining a Pipeline in SCM

Complex Pipelines are hard to write and maintain within the text area of the Pipeline configuration page. To make this easier, Pipeline can also be written in a text editor and checked into source control as a `Jenkinsfile` which Jenkins can load via the **Pipeline Script from SCM** option.

To do this, select **Pipeline script from SCM** when defining the Pipeline.

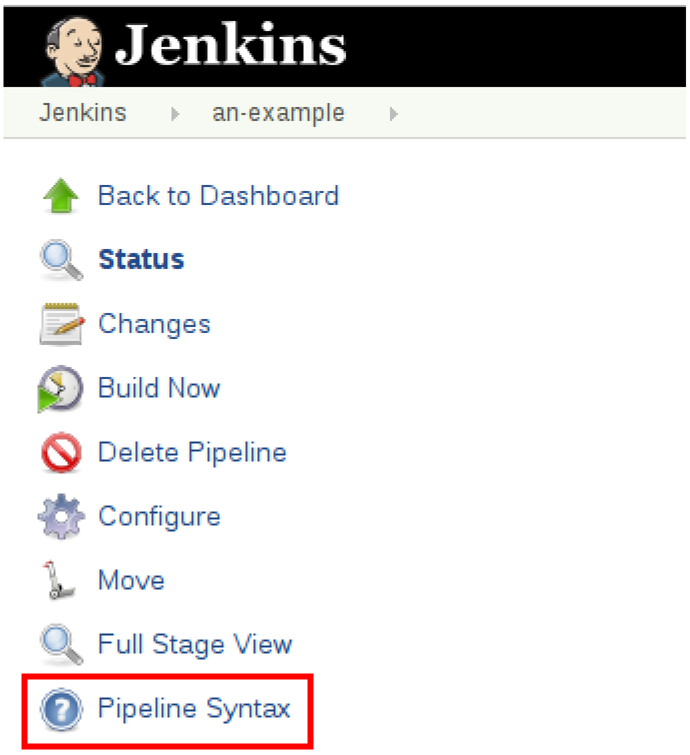
With the **Pipeline script from SCM** option selected, you do not enter any Groovy code in the Jenkins UI; you just indicate by specifying a path where in source code you want to retrieve the pipeline from. When you update the designated repository, a new build is triggered, as long as the Pipeline is configured with an SCM polling trigger.

TIP The first line of a `Jenkinsfile` should be `#!/usr/bin/env groovy` [12: en.wikipedia.org/wiki/Hashbang] [13: groovy-lang.org/syntax.html#_shebang_line] which text editors, IDEs, GitHub, etc will use to syntax highlight the `Jenkinsfile` properly as Groovy code.

Built-in Documentation

Pipeline ships with built-in documentation features to make it easier to create Pipelines of varying complexities. This built-in documentation is automatically generated and updated based on the plugins installed in the Jenkins instance.

The built-in documentation can be found globally at: localhost:8080/pipeline-syntax/, assuming you have a Jenkins instance running on localhost port 8080. The same documentation is also linked as **Pipeline Syntax** in the side-bar for any configured Pipeline project.



Snippet Generator

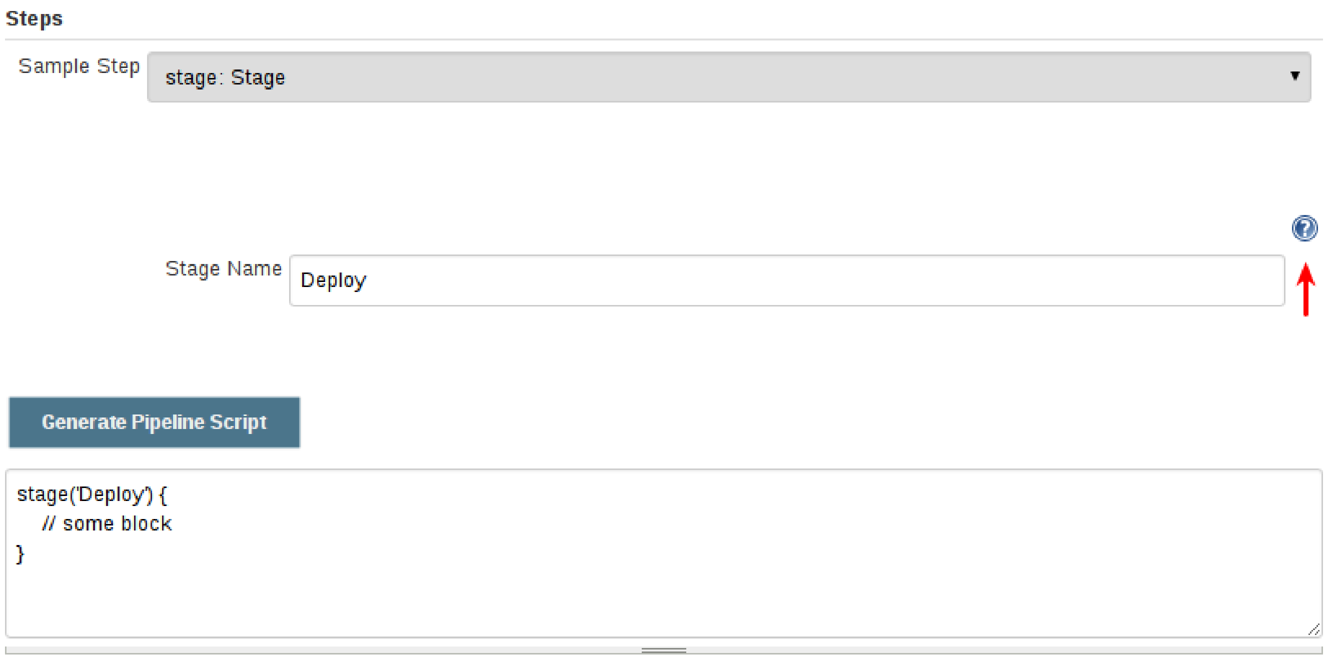
The built-in "Snippet Generator" utility is helpful for creating bits of code for individual steps, discovering new steps provided by plugins, or experimenting with different parameters for a particular step.

The Snippet Generator is dynamically populated with a list of the steps available to the Jenkins instance. The number of steps available is dependent on the plugins installed which explicitly expose steps for use in Pipeline.

To generate a step snippet with the Snippet Generator:

1. Navigate to the **Pipeline Syntax** link (referenced above) from a configured Pipeline, or at localhost:8080/pipeline-syntax.
2. Select the desired step in the **Sample Step** dropdown menu
3. Use the dynamically populated area below the **Sample Step** dropdown to configure the selected step.
4. Click **Generate Pipeline Script** to create a snippet of Pipeline which can be copied and pasted

into a Pipeline.



```
stage('Deploy') {
  // some block
}
```

To access additional information and/or documentation about the step selected, click on the help icon (indicated by the red arrow in the image above).

Global Variable Reference

In addition to the Snippet Generator, which only surfaces steps, Pipeline also provides a built-in "**Global Variable Reference**." Like the Snippet Generator, it is also dynamically populated by plugins. Unlike the Snippet Generator however, the Global Variable Reference only contains documentation for **variables** provided by Pipeline or plugins, which are available for Pipelines.

The variables provided by default in Pipeline are:

env

Environment variables accessible from Scripted Pipeline, for example: `env.PATH` or `env.BUILD_ID`. Consult the built-in Global Variable Reference for a complete, and up to date, list of environment variables available in Pipeline.

params

Exposes all parameters defined for the Pipeline as a read-only Map, for example: `params.MY_PARAM_NAME`.

currentBuild

May be used to discover information about the currently executing Pipeline, with properties such as `currentBuild.result`, `currentBuild.displayName`, etc. Consult the built-in Global Variable Reference for a complete, and up to date, list of properties available on `currentBuild`.