

# How to Use Conditional Constructs in Jenkins Pipeline

Internal reference: topics/05-1.md

## Introduction

Effective Jenkins pipelines rely heavily on conditional constructs to control the execution flow. Using these constructs, we get the flexibility to change the flow of execution based on dynamic criteria, such as environment variables, results from previous steps, or any other condition. The declarative nature of writing pipelines in Jenkins allows us to define the pipeline jobs in a simplified and structured way. In this section, let's explore different ways to write conditional logic in declarative pipelines.

## "test"-Command with "sh"-Step

Jenkins offers several built-in steps, such as `sh`, to facilitate writing pipelines conveniently. Further, we can use the `sh` block to write shell commands. Later, at the execution time, Jenkins executes these shell commands as shell scripts on one of the Jenkins nodes.

Let's write a simple Jenkins pipeline named `job-1` with a build stage that must execute the build steps only when the `SKIP_BUILD` variable isn't set:

```
pipeline {
  agent any

  stages {
    stage('build') {
      steps {
        sh
          test -z \${SKIP_BUILD} && echo 'starting to build ...'
      }
    }
  }
}
```

We can see that having access to shell commands allows us to use the `test` command for the conditional execution of the build steps after it. Further, we must note that the `SKIP_BUILD` variable is expected to be defined somewhere outside the steps section, so we need to use `\$` escaping so that Jenkins interprets it as a shell variable.

Next, let's save our pipeline job and execute it using the "Build Now" button in the sidebar:

```
Started by user admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/job-1
```

```
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (build)  
[Pipeline] sh  
+ test -z  
+ echo starting to build ...  
starting to build ...  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] }  
[Pipeline] // node  
[Pipeline] End of Pipeline  
Finished: SUCCESS
```

Since we didn't define the `SKIP_BUILD` variable, we expected the "starting to build ..." text in the console output, which is precisely the case.

## "if-else" with "sh" Step

Similar to the `test` command, we can use other shell commands and constructs within the `sh` block. Let's go ahead and write the job-2 Jenkins pipeline that uses an if-else logic to trigger the build stage:

```
pipeline {  
    agent any  
  
    stages {  
        stage('build') {  
            steps {  
                script {  
                    sh  
                        if [ -z \${SKIP_BUILD} ]  
                        then  
                            echo starting build ...  
                        else  
                            echo skipped build ...  
                        fi  
                }  
            }  
        }  
    }  
}
```

Moving on, let's trigger our job-2 pipeline and check its console output:

```
Started by user admin  
[Pipeline] Start of Pipeline  
[Pipeline] node  
Running on Jenkins in /home/jenkins/workspace/job-2
```

```
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (build)  
[Pipeline] script  
[Pipeline] {  
[Pipeline] sh  
+ [ -z ]  
+ echo starting build ...  
starting build ...  
[Pipeline] }  
[Pipeline] // script  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] }  
[Pipeline] // node  
[Pipeline] End of Pipeline  
Finished: SUCCESS
```

## "if-else" within Groovy Script

As an alternative Jenkins uses Groovy for writing the pipeline script. For more details read → [here](#).



Daniel Garavaldi

From:

<https://wiki.bzz.ch/> - **BZZ - Modulwiki**

Permanent link:

[https://wiki.bzz.ch/en/modul/m324\\_aws/topics/08](https://wiki.bzz.ch/en/modul/m324_aws/topics/08)

Last update: **2025/10/15 14:06**

