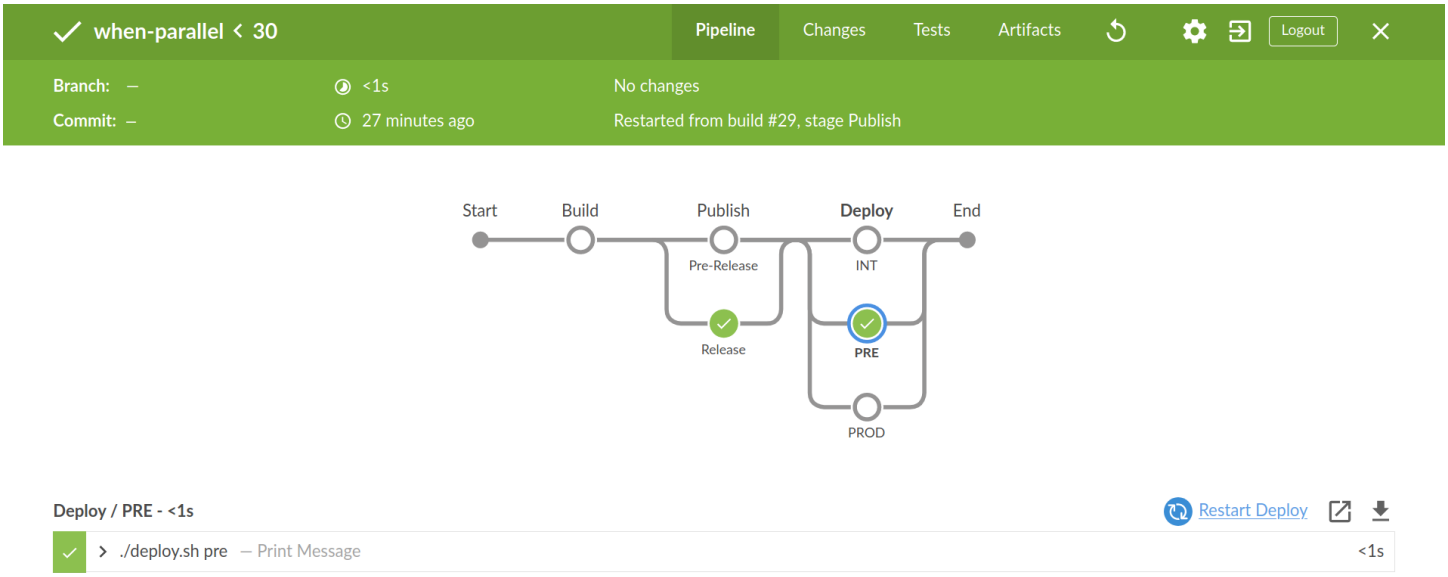
Michael Kutz

Follow 30 Followers About

Conditionals in a Declarative Pipeline Jenkinsfile

Michael Kutz Aug 3, 2020 · 4 min read



In this article I'll show how to express conditionals — like `if`, `else` or `switch` — in a Jenkinsfile using the declarative pipeline syntax.

Optional Stages (≅ if)

Generally it is possible to use Groovy's conditionals in a declarative syntax, when we use a `script` step.

So for example, if we only want a release to happen, if a certain boolean parameter `RELEASE` is set, we code it like this:

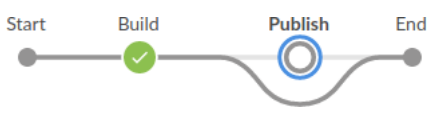
But in that case, we end up with a stage that looks like it was successfully executed, but in fact it didn't do anything.

The pipeline looks like this, no matter if we did anything in the Publish stage or not

The declarative way of expressing this is the `when` directive, which will skip or execute a whole stage according to a condition.

And then evaluate its value in a `when` directive on the release stage:

Nice. This code is even shorter and less indented. Jenkins will display the stage to be skipped or executed in the build overview, so we can find the last build that performed a release without having to check the logs.



Now the Publish stage is marked as skipped

mechanism to build a pre-release and another to build a final release.

So, here's the version using `script` with `if` and `else`:

```groovy
#!/usr/bin/env groovy
// see https://jenkins.io/doc/book/pipeline/syntax/

pipeline {

    agent any

    parameters {
        booleanParam(name: "RELEASE", defaultValue: false)
    }

    stages {

        stage("Build") {
            steps {
                sh "./gradlew build"
            }
        }

        stage("Publish") {
            steps {
                script {
                    if (params.RELEASE) {
                        sh "./gradlew release"
                    } else {
                        sh "./gradlew preRelease"
                    }
                }
            }
        }
    }
}
```

IfElseScriptedJenkinsfile.groovy hosted with ❤ by GitHub view raw

Not too bad. This is short and still readable. However, to Jenkins this is one step and to find out what happened in hindsight is not easy as we'd need to check the logs on the executions.

We can get more declarative using two successive stages with `when`, where the first when can only be true, if the second is false and vice versa.

```groovy
#!/usr/bin/env groovy
// see https://jenkins.io/doc/book/pipeline/syntax/

pipeline {

    agent any

    parameters {
        booleanParam(name: "RELEASE", defaultValue: false)
    }

    stages {

        stage("Build") {
            steps {
                sh "./gradlew build"
            }
        }

        stage("Publish Pre-Release") {
```

```
            }
        }

        stage("Publish Release") {
            when { expression { params.RELEASE } }
            steps {
                sh "./gradlew release"
            }
        }
    }
}
```

IfElseDeclarativeJenkinsfile.groovy hosted with ❤ by GitHub view raw

This is a bit more code, but still reads alright and we get a much better understanding by looking at the graph.

The two alternative stages are not logically grouped

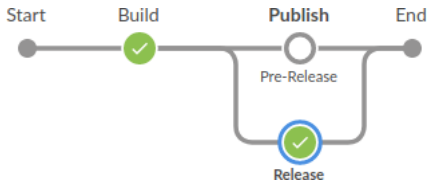
But after all, both stages are doing the same thing (publishing) and it would be nicer to have them grouped by something more then just by name.

For this we can wrap both stages in a `parallel` section:

```
#!/usr/bin/env groovy
// see https://jenkins.io/doc/book/pipeline/syntax/

pipeline {

    agent any

    parameters {
        booleanParam(name: "RELEASE", defaultValue: false)
    }

    stages {

        stage("Build") {
            steps {
                sh "./gradlew build"
            }
        }

        stage("Publish") {
            parallel {
                stage("Release") {
                    when { expression { params.RELEASE } }
                    steps {
                        sh "./gradlew release"
                    }
                }
                stage('Pre-Release') {
                    when { expression { !params.RELEASE } }
                    steps {
                        sh "./gradlew preRelease"
                    }
                }
            }
        }
```

IfElseDeclarativeParallelJenkinsfile.groovy hosted with ❤ by GitHub view raw

Now, when we look at the pipeline graph, we get a nice overview, which way the build took.



The two Publish stages are grouped by the parallel block and are clearly visible as alternatives

One of Many (≅ switch)

In some cases, we not only want to chose one out of two, but out of many alternatives. E.g. we'd like to add a deploy stage but want to choose the target stage via parameter.

Again, we can fallback to `script` and use `switch` for this:

```
#!/usr/bin/env groovy
// see https://jenkins.io/doc/book/pipeline/syntax/

pipeline {

    agent any

    parameters {
        booleanParam(name: "RELEASE", defaultValue: false)
        choice(name: "DEPLOY_TO", choices: ["", "INT", "PRE", "PROD"])
    }

    stages {

        stage("Build") {
            steps {
                sh "./gradlew build"
            }
        }

        stage("Publish") {
            parallel {
                stage('Pre-Release') {
                    when { expression { !params.RELEASE } }
                    steps {
                        sh "./gradlew preRelease"
                    }
                }
                stage("Release") {
                    when { expression { params.RELEASE } }
                    steps {
                        sh "./gradlew release"
                    }
                }
            }
        }

        stage("Deploy") {
            steps {
                script {
                    switch(params.DEPLOY_TO) {
                        case "INT": echo "./deploy.sh int"; break
                        case "PRE": echo "./deploy.sh pre"; break
                        case "PROD": echo "./deploy.sh prod"; break
                    }
                }
```

```
50  }
```

SwitchScriptedJenkinsfile.groovy hosted with ❤ by GitHub view raw

Again, this is quite short and well readable in the Jenkinsfile, but in the pipeline graph, it is hard to see, where the deployment went to.

The graph does not tell us, if a deployment happened or where it went to

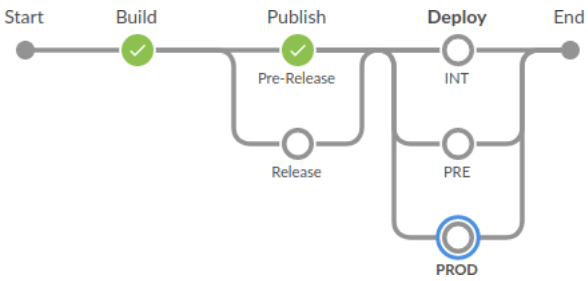
But we can apply the same trick as before, only now we use a `choice` parameter:

```groovy
#!/usr/bin/env groovy
// see https://jenkins.io/doc/book/pipeline/syntax/

pipeline {

    agent any

    parameters {
        booleanParam(name: "RELEASE", defaultValue: false)
        choice(name: "DEPLOY_TO", choices: ["", "INT", "PRE", "PROD"])
    }

    stages {

        stage("Build") {
            steps {
                sh "./gradlew build"
            }
        }

        stage("Publish") {
            parallel {
                stage('Pre-Release') {
                    when { expression { !params.RELEASE } }
                    steps {
                        sh "./gradlew preRelease"
                    }
                }
                stage("Release") {
                    when { expression { params.RELEASE } }
                    steps {
                        sh "./gradlew release"
                    }
                }
            }
        }

        stage("Deploy") {
            parallel {
                stage("INT") {
                    when { expression { params.DEPLOY_TO == "INT" } }
                    steps {
                        sh "./deploy int"
                    }
                }
                stage("PRE") {
```

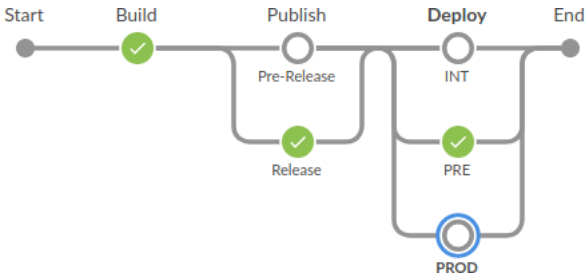
```
                    }
                }
                stage("PROD") {
                    when { expression { params.DEPLOY_TO == "PROD" } }
                    steps {
                        sh "./deploy.sh prod"
                    }
                }
            }
        }
    }
}
```

SwitchDeclarativeJenkinsfile.groovy hosted with ❤ by GitHub view raw

This is a lot more code, compared to the simple `script` solution above, but the pipeline graph gets a lot more expressive.



Without any changes, the Deploy stages gets skipped entirely, since no when directive expression evaluates to true



This graph is the result with RELEASE checked and PRE chosen for DEPLOY_TO,

Conclusion

Especially the later examples clearly show that declarative pipelines are not necessarily the best choice regarding the amount of (stupid) code. However, the resulting pipeline graphs show what happened very nicely.

Grouping alternative stages in `parallel` makes the graphs even more expressive and also binds the repetitive code together.

Jenkins Pipeline Jenkins Pipeline As Code