Using a text editor, ideally one which supports Groovy syntax highlighting, create a new `Jenkinsfile` in the root directory of the project.

The Declarative Pipeline example above contains the minimum necessary structure to implement a continuous delivery pipeline. The agent directive, which is required, instructs Jenkins to allocate an executor and workspace for the Pipeline. Without an `agent` directive, not only is the Declarative Pipeline not valid, it would not be capable of doing any work! By default the `agent` directive ensures that the source repository is checked out and made available for steps in the subsequent stages`

The stages directive, and steps directives are also required for a valid Declarative Pipeline as they instruct Jenkins what to execute and in which stage it should be executed.

For more advanced usage with Scripted Pipeline, the example above `node` is a crucial first step as it allocates an executor and workspace for the Pipeline. In essence, without `node`, a Pipeline cannot do any work! From within `node`, the first order of business will be to checkout the source code for this project. Since the `Jenkinsfile` is being pulled directly from source control, Pipeline provides a quick and easy way to access the right revision of the source code

```
// Script //
node {
    checkout scm ①
    /* .. snip .. */
}
// Declarative not yet implemented //
```

① The `checkout` step will checkout code from source control; `scm` is a special variable which instructs the `checkout` step to clone the specific revision which triggered this Pipeline run.

Build

For many projects the beginning of "work" in the Pipeline would be the "build" stage. Typically this stage of the Pipeline will be where source code is assembled, compiled, or packaged. The `Jenkinsfile` is **not** a replacement for an existing build tool such as GNU/Make, Maven, Gradle, etc, but rather can be viewed as a glue layer to bind the multiple phases of a project’s development lifecycle (build, test, deploy, etc) together.

Jenkins has a number of plugins for invoking practically any build tool in general use, but this example will simply invoke `make` from a shell step (`sh`). The `sh` step assumes the system is Unix/Linux-based, for Windows-based systems the `bat` could be used instead.