The following variables are available to shell scripts

* + BRANCH\_NAME: For a multibranch project, this will be set to the name of the branch being built, for example in case you wish to deploy to production from master but not from feature branches; if corresponding to some kind of change request, the name is generally arbitrary (refer to CHANGE\_ID and CHANGE\_TARGET).
  + BRANCH\_IS\_PRIMARY: For a multibranch project, if the SCM source reports that the branch being built is a primary branch, this will be set to "true"; else unset. Some SCM sources may report more than one branch as a primary branch while others may not supply this information.
  + CHANGE\_ID: For a multibranch project corresponding to some kind of change request, this will be set to the change ID, such as a pull request number, if supported; else unset.
  + CHANGE\_URL: For a multibranch project corresponding to some kind of change request, this will be set to the change URL, if supported; else unset.
  + CHANGE\_TITLE: For a multibranch project corresponding to some kind of change request, this will be set to the title of the change, if supported; else unset.
  + CHANGE\_AUTHOR: For a multibranch project corresponding to some kind of change request, this will be set to the username of the author of the proposed change, if supported; else unset.
  + CHANGE\_AUTHOR\_DISPLAY\_NAME: For a multibranch project corresponding to some kind of change request, this will be set to the human name of the author, if supported; else unset.
  + CHANGE\_AUTHOR\_EMAIL: For a multibranch project corresponding to some kind of change request, this will be set to the email address of the author, if supported; else unset.
  + CHANGE\_TARGET: For a multibranch project corresponding to some kind of change request, this will be set to the target or base branch to which the change could be merged, if supported; else unset.
  + CHANGE\_BRANCH: For a multibranch project corresponding to some kind of change request, this will be set to the name of the actual head on the source control system which may or may not be different from BRANCH\_NAME. For example in GitHub or Bitbucket this would have the name of the origin branch whereas BRANCH\_NAME would be something like PR-24.
  + CHANGE\_FORK: For a multibranch project corresponding to some kind of change request, this will be set to the name of the forked repo if the change originates from one; else unset.
  + TAG\_NAME: For a multibranch project corresponding to some kind of tag, this will be set to the name of the tag being built, if supported; else unset.
  + TAG\_TIMESTAMP: For a multibranch project corresponding to some kind of tag, this will be set to a timestamp of the tag in milliseconds since Unix epoch, if supported; else unset.
  + TAG\_UNIXTIME: For a multibranch project corresponding to some kind of tag, this will be set to a timestamp of the tag in seconds since Unix epoch, if supported; else unset.
  + TAG\_DATE: For a multibranch project corresponding to some kind of tag, this will be set to a timestamp in the format as defined by java.util.Date#toString() (e.g., Wed Jan 1 00:00:00 UTC 2020), if supported; else unset.
  + JOB\_DISPLAY\_URL: URL that will redirect to a Job in a preferred user interface
  + RUN\_DISPLAY\_URL: URL that will redirect to a Build in a preferred user interface
  + RUN\_ARTIFACTS\_DISPLAY\_URL: URL that will redirect to Artifacts of a Build in a preferred user interface
  + RUN\_CHANGES\_DISPLAY\_URL: URL that will redirect to Changelog of a Build in a preferred user interface
  + RUN\_TESTS\_DISPLAY\_URL: URL that will redirect to Test Results of a Build in a preferred user interface
  + CI: Statically set to the string "true" to indicate a "continuous integration" execution environment.
  + BUILD\_NUMBER: The current build number, such as "153".
  + BUILD\_ID: The current build ID, identical to BUILD\_NUMBER for builds created in 1.597+, but a YYYY-MM-DD\_hh-mm-ss timestamp for older builds.
  + BUILD\_DISPLAY\_NAME: The display name of the current build, which is something like "#153" by default.
  + JOB\_NAME: Name of the project of this build, such as "foo" or "foo/bar".
  + JOB\_BASE\_NAME: Short Name of the project of this build stripping off folder paths, such as "foo" for "bar/foo".
  + BUILD\_TAG: String of "jenkins-${JOB\_NAME}-${BUILD\_NUMBER}". All forward slashes ("/") in the JOB\_NAME are replaced with dashes ("-"). Convenient to put into a resource file, a jar file, etc for easier identification.
  + EXECUTOR\_NUMBER: The unique number that identifies the current executor (among executors of the same machine) that’s carrying out this build. This is the number you see in the "build executor status", except that the number starts from 0, not 1.
  + NODE\_NAME: Name of the agent if the build is on an agent, or "master" if run on master.
  + NODE\_LABELS: Whitespace-separated list of labels that the node is assigned.
  + WORKSPACE: The absolute path of the directory assigned to the build as a workspace.
  + WORKSPACE\_TMP: A temporary directory near the workspace that will not be browsable and will not interfere with SCM checkouts. May not initially exist, so be sure to create the directory as needed (e.g., mkdir -p on Linux). Not defined when the regular workspace is a drive root.
  + JENKINS\_HOME: The absolute path of the directory assigned on the master node for Jenkins to store data.
  + JENKINS\_URL: Full URL of Jenkins, like <http://server:port/jenkins/> (note: only available if Jenkins URL set in system configuration).
  + BUILD\_URL: Full URL of this build, like <http://server:port/jenkins/job/foo/15/> (Jenkins URL must be set).
  + JOB\_URL: Full URL of this job, like <http://server:port/jenkins/job/foo/> (Jenkins URL must be set).
  + GIT\_COMMIT: The commit hash being checked out.
  + GIT\_PREVIOUS\_COMMIT: The hash of the commit last built on this branch, if any.
  + GIT\_PREVIOUS\_SUCCESSFUL\_COMMIT: The hash of the commit last successfully built on this branch, if any.
  + GIT\_BRANCH: The remote branch name, if any.
  + GIT\_LOCAL\_BRANCH: The local branch name being checked out, if applicable.
  + GIT\_CHECKOUT\_DIR: The directory that the repository will be checked out to. This contains the value set in Checkout to a sub-directory, if used.
  + GIT\_URL: The remote URL. If there are multiple, will be GIT\_URL\_1, GIT\_URL\_2, etc.
  + GIT\_COMMITTER\_NAME: The configured Git committer name, if any, that will be used for FUTURE commits from the current workspace. It is read from the Global Config user.name Value field of the Jenkins Configure System page.
  + GIT\_AUTHOR\_NAME: The configured Git author name, if any, that will be used for FUTURE commits from the current workspace. It is read from the Global Config user.name Value field of the Jenkins Configure System page.
  + GIT\_COMMITTER\_EMAIL: The configured Git committer email, if any, that will be used for FUTURE commits from the current workspace. It is read from the Global Config user.email Value field of the Jenkins Configure System page.
  + GIT\_AUTHOR\_EMAIL: The configured Git author email, if any, that will be used for FUTURE commits from the current workspace. It is read from the Global Config user.email Value field of the Jenkins Configure System page.
  + SVN\_REVISION: Subversion revision number that’s currently checked out to the workspace, such as "12345"
  + SVN\_URL: Subversion URL that’s currently checked out to the workspace.
* Refer the below jenkinsfile to see the creation and usage of existing environment variables

pipeline {

agent { label 'GOL'}

triggers {

cron('H \* \* \* \*')

pollSCM('\* \* \* \* \*')

}

parameters {

string(name: 'BRANCH', defaultValue: 'master', description: 'Branch to build' )

}

environment {

CI\_ENV = 'DEV'

}

stages {

stage('scm') {

environment {

DUMMY = 'FUN'

}

steps {

git branch: "${params.BRANCH}", url: 'https://github.com/asquarezone/game-of-life.git'

//input message: 'Continue to next stage? ', submitter: 'qtaws,qtazure'

echo env.CI\_ENV

echo env.DUMMY

}

}

stage('build') {

steps {

echo env.GIT\_URL

sh 'mvn package'

}

}

}

post {

success {

archive '\*\*/gameoflife.war'

junit '\*\*/TEST-\*.xml'

}

}

}

**Flow Control Options**

* timeout: This step allows you to limit the amount of your script spends waiting for action to happen
* sleep: This is basic delay step

sleep time: 7, unit: 'MINUTES'

* retry: This closure retires the process n times if an exception

retry(3){

sh 'mvn clean package'

}

* For notification on build failures we can use the mail step
* Refer the below Jenkins file with notifications and other additions as discussed in the class

pipeline {

agent { label 'GOL'}

triggers {

cron('H \* \* \* \*')

pollSCM('\* \* \* \* \*')

}

parameters {

string(name: 'BRANCH', defaultValue: 'master', description: 'Branch to build' )

choice(name: 'GOAL', choices: ['package', 'clean package', 'install'], description: 'maven goals')

}

options {

timeout(time: 1, unit: 'HOURS')

retry(2)

}

environment {

CI\_ENV = 'DEV'

}

stages {

stage('scm') {

environment {

DUMMY = 'FUN'

}

steps {

mail subject: 'BUILD Started '+env.BUILD\_ID, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

git branch: "${params.BRANCH}", url: 'https://github.com/asquarezone/game-of-life.git'

//input message: 'Continue to next stage? ', submitter: 'qtaws,qtazure'

echo env.CI\_ENV

echo env.DUMMY

}

}

stage('build') {

steps {

echo env.GIT\_URL

timeout(time:10, unit: 'MINUTES') {

sh "mvn ${params.GOAL}"

}

}

}

}

post {

success {

archive '\*\*/gameoflife.war'

junit '\*\*/TEST-\*.xml'

mail subject: 'BUILD Completed Successfully '+env.BUILD\_ID, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

}

failure {

mail subject: 'BUILD Failed '+env.BUILD\_ID+'URL is '+env.BUILD\_URL, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

}

always {

echo "Finished"

}

changed {

echo "Changed"

}

unstable {

mail subject: 'BUILD Unstable '+env.BUILD\_ID+'URL is '+env.BUILD\_URL, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

}

}

}