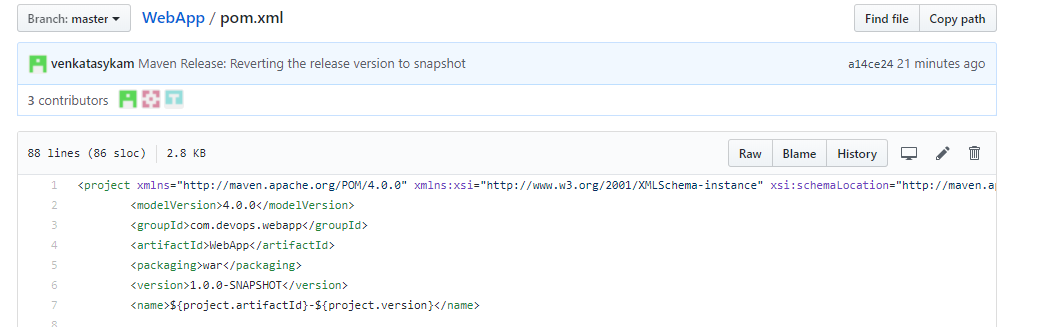
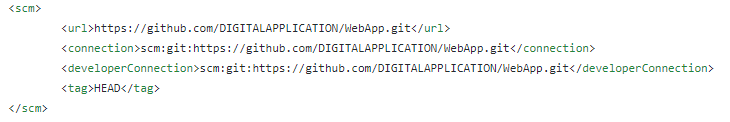
**Create a new item using freestyle or maven template job:**

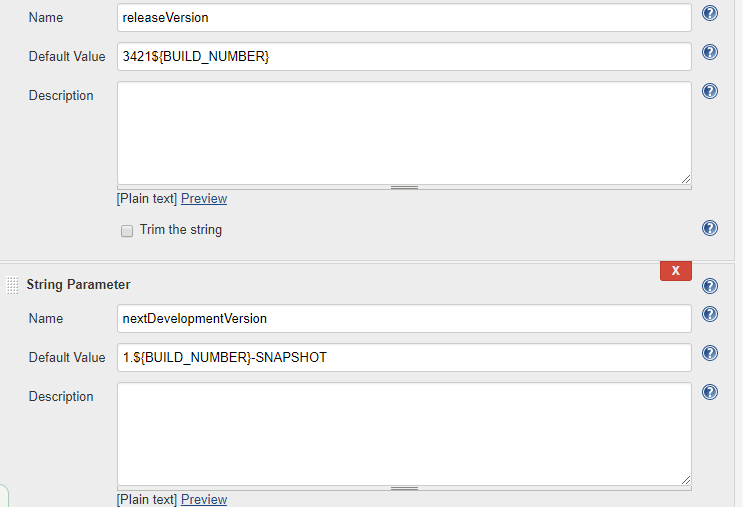
1. **POM file configuration:**

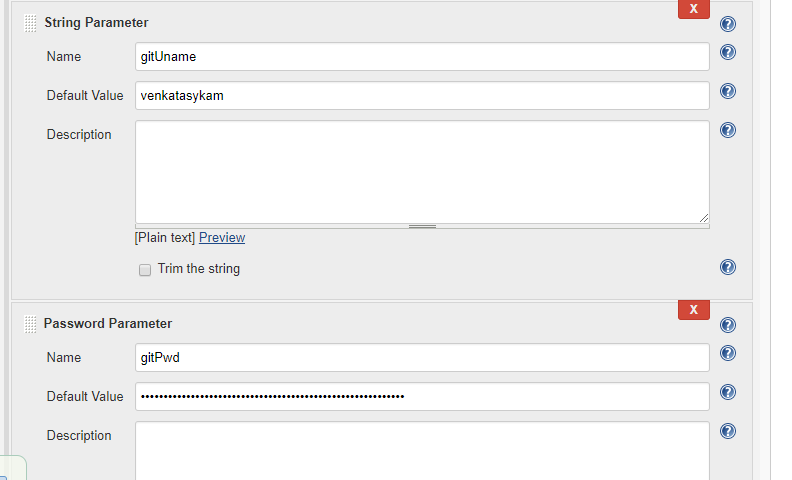
****

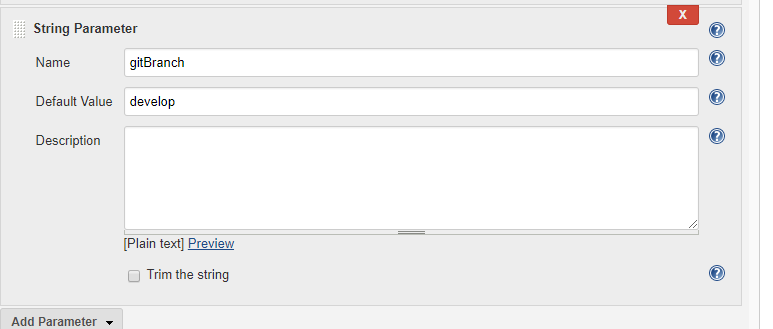
****

****

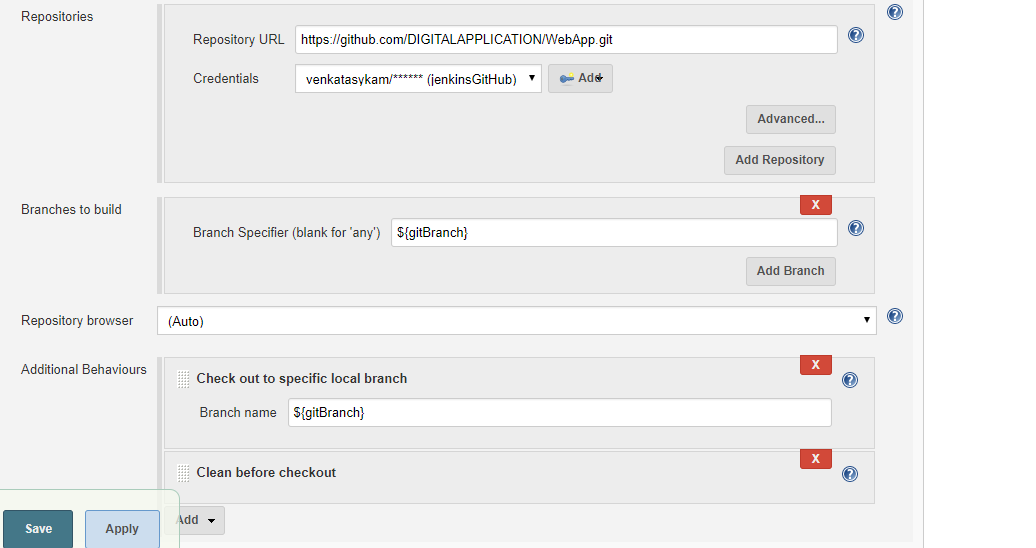
1. **Parameters sections**:



****

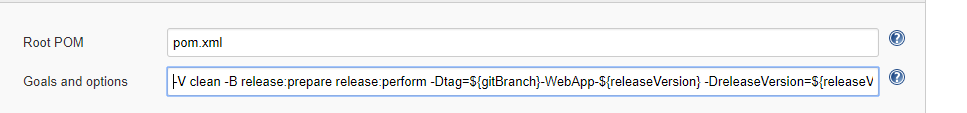
****

1. **SCM Section:**

****

1. **Build section:**

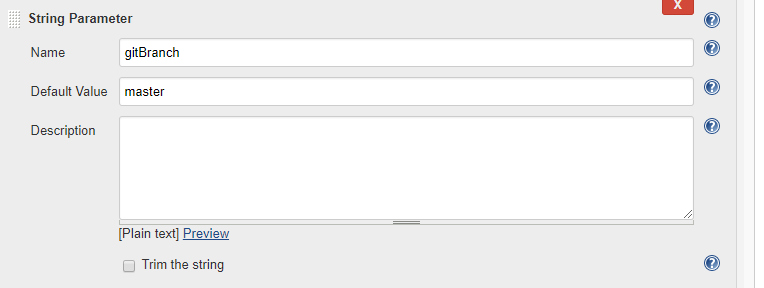
|  |
| --- |
| -V clean -B release:prepare release:perform -Dtag=${gitBranch}-WebApp-${releaseVersion} -DreleaseVersion=${releaseVersion} -DdevelopmentVersion=${nextDevelopmentVersion} -Dusername=${gitUname} -Dpassword=${gitPwd} |

****

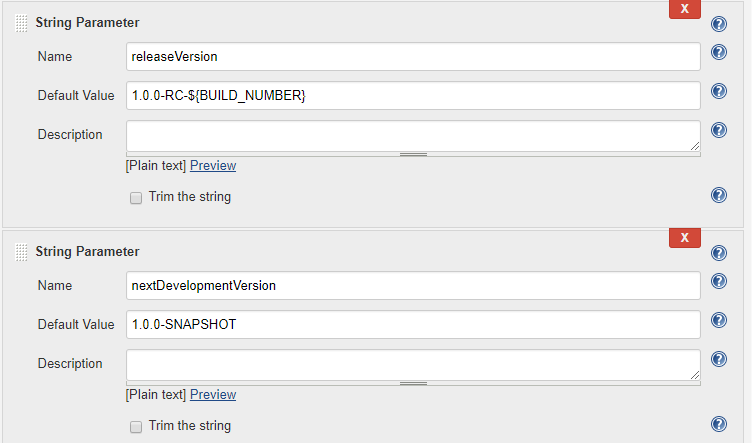
**-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Release the artefact in another way: (Both RC & SNASPHOT in single job)**

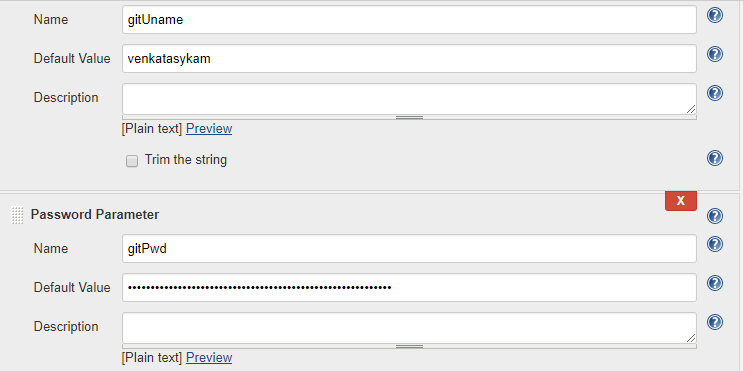
1. **Create a freestyle job.**
2. **Parameters section:**
   1. **Pass the gitBranch name to check out the code.**

****

* 1. **Pass the release version and next development version(i.e., SNAPSHOT version)**

****

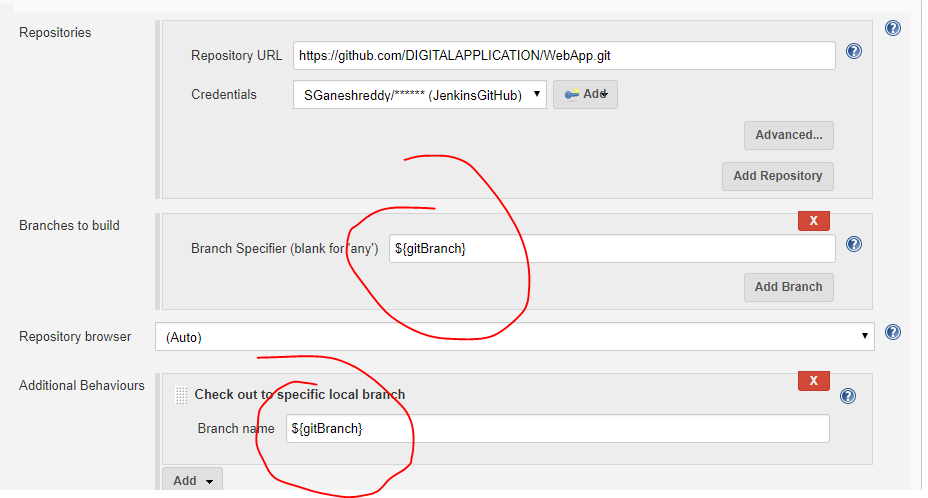
* 1. **Pass the git credentials.**

****

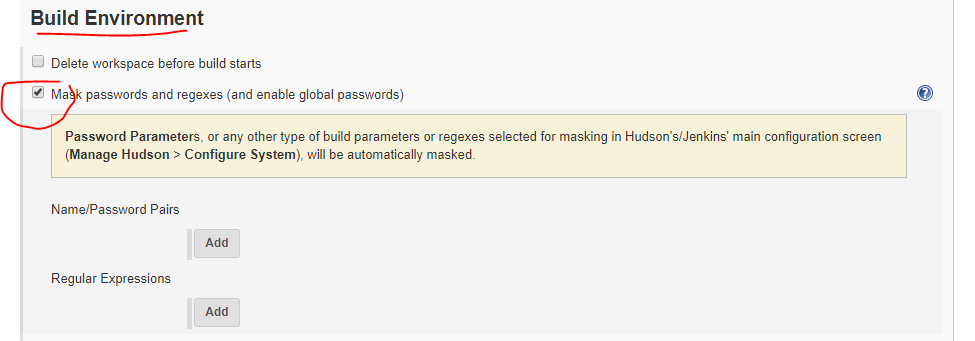
* 1. **Build type**

****

1. **SCM section:**

****

1. **Build Environment Section: Select the checkbox “Mask Passwords”, so thyat passwords will not be visible in the Jenkins build console.**

****

1. **Build section: Execute Shell.**

**1. Build 🡪 Execute Shell.**

**2. If you choose the build type parameter as “SNAPSHOT”, then maven goals will run only “deploy” goal, it will build and deploy the SNAPSHOT package to artifactory/nexus.**

**3. If you choose the build type parameter as “RC”, then the following steps 4,5,6 will be executed.**

**4. Git pull to make sure the git credentials are working file also, the code will be updated with latest changes.**

**5. To store the credentials for 60 secs in the Jenkins slave/node. (i.e., linux/unix box).**

**6. This is the release process.**

**6.1.** mvn versions:set -DnewVersion=${releaseVersion} : 🡪 Version update in all the pom file with the release version which we passed as a parameter to this Jenkins job.

**6.2.** mvn clean deploy**: 🡪 build and deploy the release versioned package to artifactory/nexus release repo.**

**6.3.** Git is always creating the tag at latest code which is exactly synch with remote repo changes. Git will **not** create a tag on staged changes or un-committed code.

To create the tag successfully, we need to commit and push the changes first.

git commit -am "Updating the poms with release version, this will be revereted once release build is completed"

git push origin ${gitBranch}

6.4. Create a tag and push to the remote repo.

git tag -a ${gitBranch}-${releaseVersion} -m "${gitBranch}-${releaseVersion}" (actual syntax is git tag -a <tagName> <commitID> -m "message here")

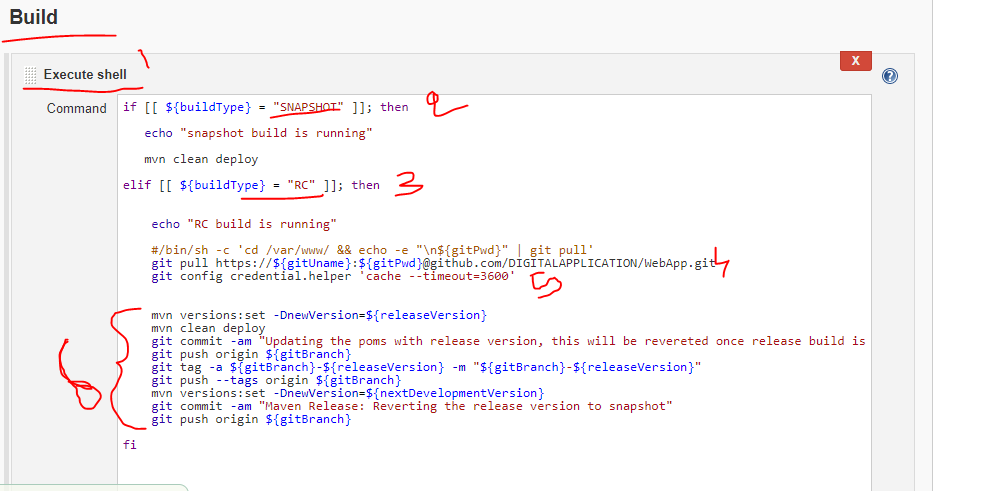
git push --tags origin ${gitBranch} (Or you can push the particular tag to remote repo: git push origin <tagName>)

6.5. Revert the version in pom files to next development version(i.e., SNAPSHOT) and then commit & push the changes to remote repo.

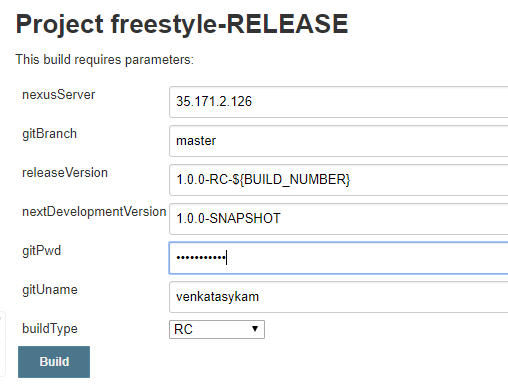
mvn versions:set -DnewVersion=${nextDevelopmentVersion}

git commit -am "Maven Release: Reverting the release version to snapshot"

git push origin ${gitBranch}

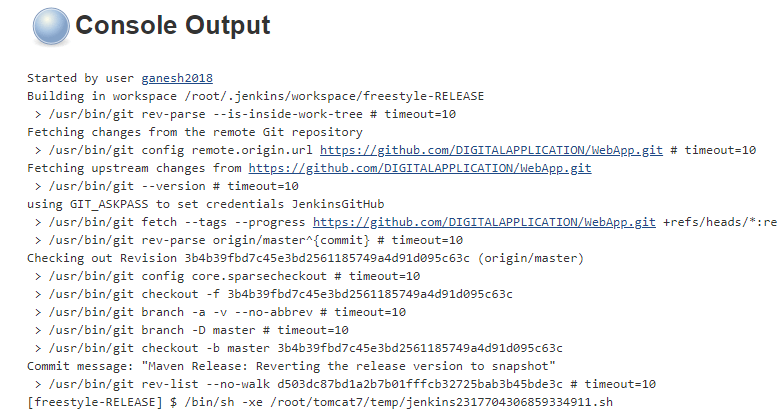
****

**Execution:** Passing the required parameters, like gitbranch, gitUname, PWD, buildType etc... And then click on build.

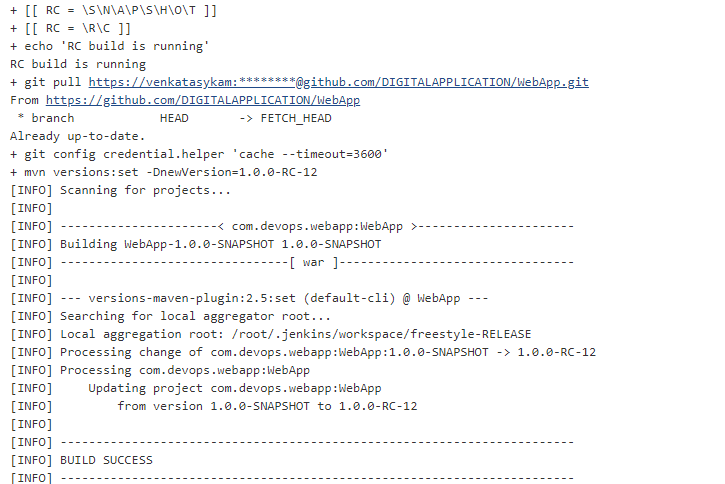
****

**Output:**

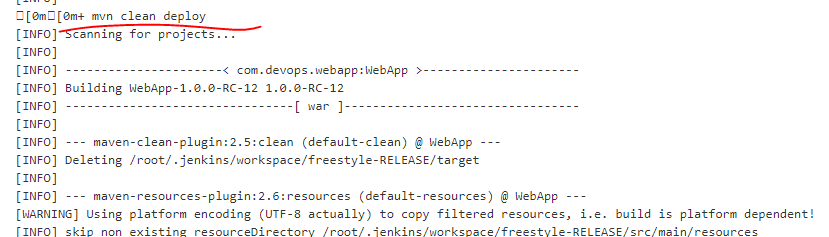
**Code checkout screen:**

****

**Condition check and version update locally (in Jenkins workspace):**

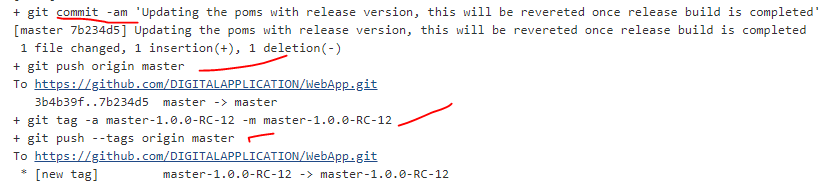
****

**Running the maven goal: mvn clean deploy**

****

****

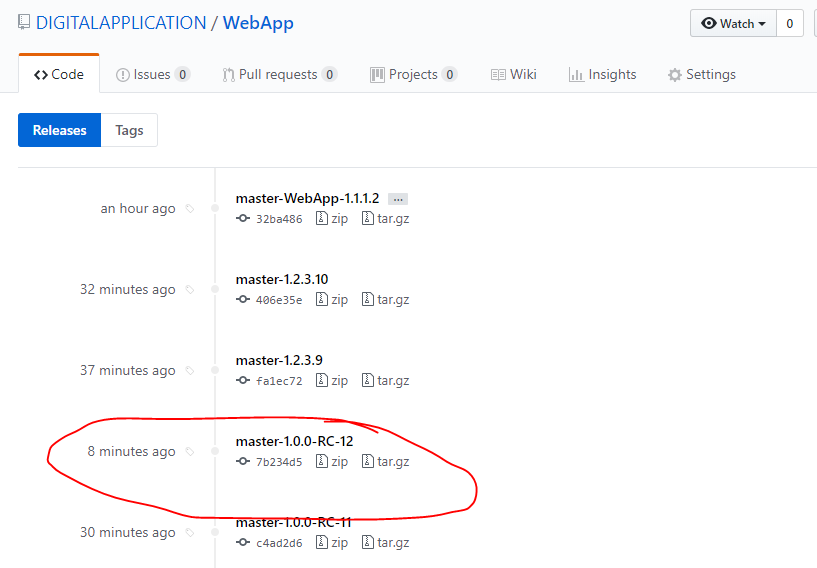
**Committing the local changes and push to github And then creating tag or push the tag top remote repo.**

****

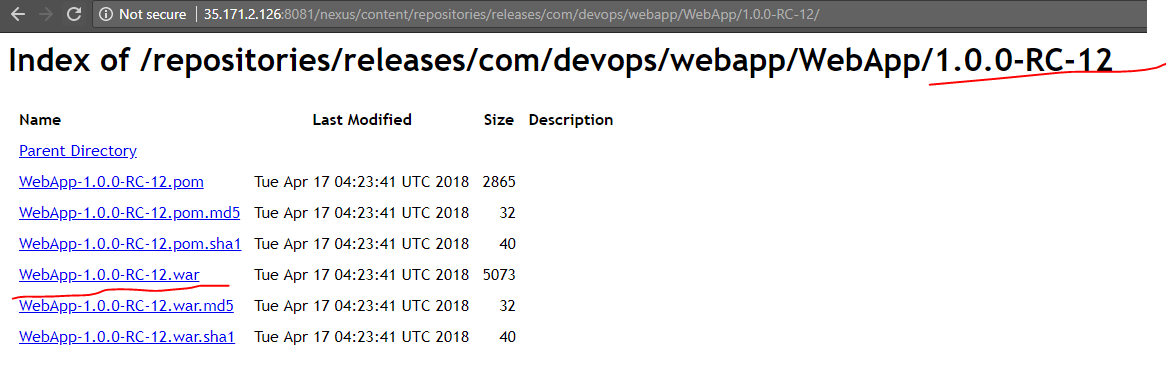
**Revert the version & commit, push to GitHub.**

****

**Tag has been pushed to GitHub:**

****

**Artifact published to nexus.**

****