Auto Version Bump

Code

Groovy Script can be found at Library .

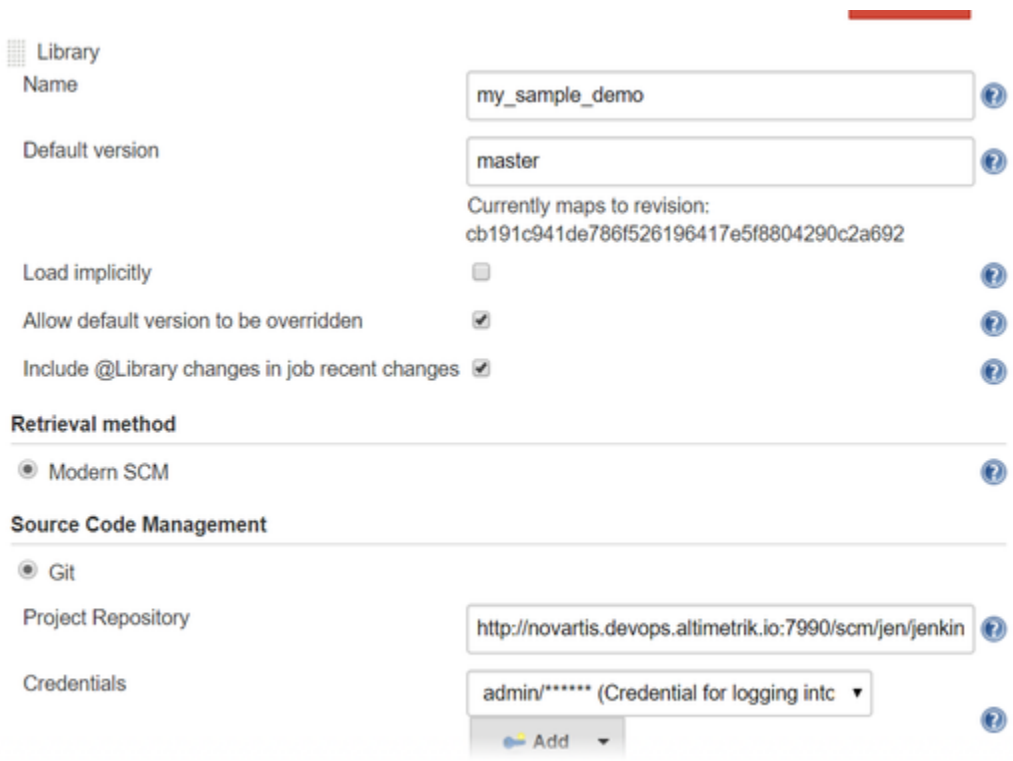
```
#!/usr/bin/groovy

import hudson.EnvVars;
import hudson.slaves.EnvironmentVariablesNodeProperty;
import hudson.slaves.NodeProperty;
import hudson.slaves.NodePropertyDescriptor;
import hudson.util.DescribableList;
import jenkins.model.Jenkins;
import groovy.transform.Field

@Field nextversion

def call(Map config = [:]) {

 if (("${env.BUILD_NUMBER}"=="1") && (env.BRANCH_NAME == 'master'))
 {
  if ((! config.first_version) )
  {
   config.first_version = "0.1.0"
  }

  createGlobalEnvironmentVariables('Current_Version',
"${config.first_version}")
  nextversion = "${config.first_version}-${env.BUILD_NUMBER}"
 } else

  {  if ((config) && (config.current_version)) {
     nextversion = nextPackageVersion("${config.current_version}")
   } else {
     nextversion = nextPackageVersion(env.Current_Version)
     }
   createGlobalEnvironmentVariables('Current_Version', nextversion)
   nextversion = "${nextversion}-${env.BUILD_NUMBER}"
  }


 print "Next Version"
 print nextversion
    }

def nextPackageVersion(String latestVersion) {
    latestVersion = latestVersion.replaceAll("\"", "");
    def (major, minor, patch) = latestVersion.tokenize('.').collect {
it.toInteger() }
```

```
    def nextVersion
    switch (env.BRANCH_NAME) {
        case 'master':
            nextVersion = "${major + 1}.${minor}.${patch}"
            break
        case 'development':
            nextVersion = "${major}.${minor + 1}.${patch}"
            break
  case ~/.*feature.*/:
   nextVersion = "${major}.${minor}.${patch + 1}"

        default:
            nextVersion = "${major}.${minor}.${patch + 1}"
            break
    }
 nextVersion

}

public createGlobalEnvironmentVariables(String key, String value){

        Jenkins instance = Jenkins.getInstance();

        DescribableList<NodeProperty<?>, NodePropertyDescriptor>
globalNodeProperties = instance.getGlobalNodeProperties();
        List<EnvironmentVariablesNodeProperty> envVarsNodePropertyList =
globalNodeProperties.getAll(EnvironmentVariablesNodeProperty.class);

        EnvironmentVariablesNodeProperty newEnvVarsNodeProperty = null;
        EnvVars envVars = null;

        if ( envVarsNodePropertyList == null ||
envVarsNodePropertyList.size() == 0 ) {
            newEnvVarsNodeProperty = new
hudson.slaves.EnvironmentVariablesNodeProperty();
            globalNodeProperties.add(newEnvVarsNodeProperty);
            envVars = newEnvVarsNodeProperty.getEnvVars();
        } else {
            envVars = envVarsNodePropertyList.get(0).getEnvVars();
        }
```

```
        envVars.put(key, value)
        instance.save()
}
```

- Integrate the Jenkins Library having above groovy file as follows,

To do so, go into **Manage Jenkins -> Configure System** and find the **Global Pipeline Libraries** section. The shared library will be loaded on the fly from a git repository, in every job. It is never cached.



The Project repository is the url of your git repo where Global library code is present, Shared Library

Usage

- You can use this Library in your Jenkinfile as follows,

```
@Library('my-sample-demo') _

autoversion(
    first_version : '',
 current_version : ''
    )
print autoversion.nextversion
```

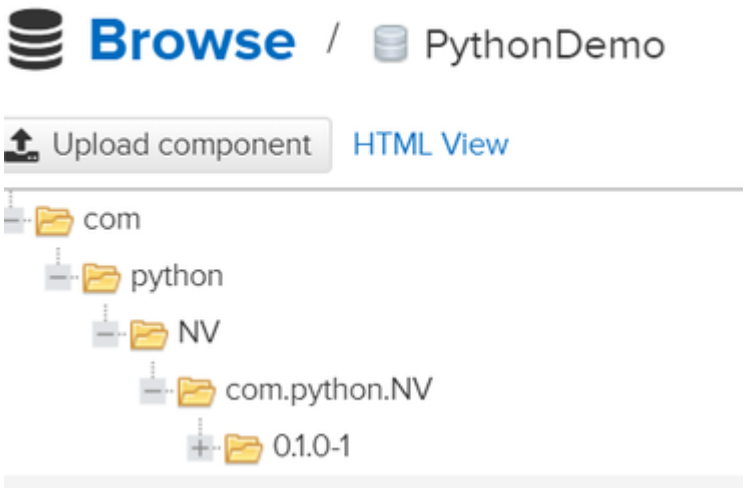
- Here, **first_version** and **current_version** both are optional variable. If you don`t provide first_version it will take **0.1.0** as default.
- If you are not giving **current_version**, it will take last version as the current version.
- **autoversion.nextversion** will give you the next version value.

Working Example

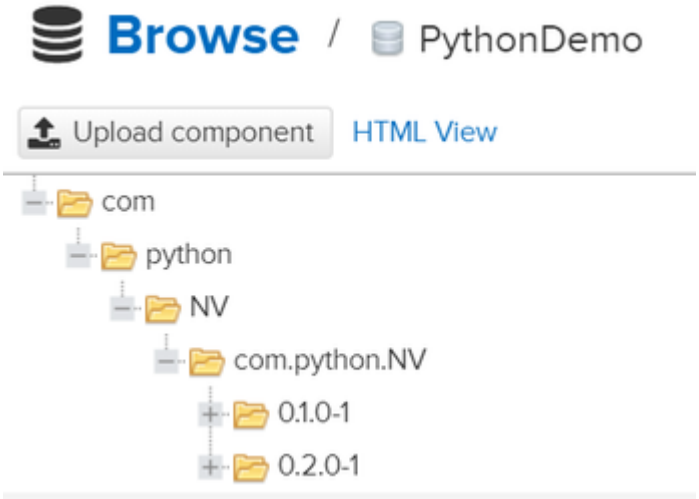
```
@Library('Auto_Version') _

autoversion()

pipeline {
    agent {label 'python-slave'}
    options {
        skipStagesAfterUnstable()
    }
    stages {
        stage('Build') {
            steps {
                sh 'python -m py_compile sources/add2vals.py
sources/calc.py'
            }
        }
        stage('Test') {
            steps {
                sh 'PYTHONPATH="$WORKSPACE/sources" python test/test.py'
            }
            post {
                always {
                    junit 'test-reports/*.xml'
                }
            }
        }
        stage('Package') {

            steps {
                sh 'python setup.py bdist'
            }

        }
  stage("Publsih to Nexus"){

   steps {

    nexusArtifactUploader artifacts: [[artifactId: 'com.python.NV',
classifier: '', file: 'dist/py-jenkins-0.1.0.linux-x86_64.tar.gz',
    type: 'tar.gz']], credentialsId: 'nexus', groupId: 'com.python.NV',
nexusUrl: 'novartis.devops.altimetrik.io:8082', nexusVersion: 'nexus3',
protocol:
    'http', repository: 'PythonDemo', version: autoversion.nextversion

   }
        }
 }
}
```

We can notice that while pushing the artifact to Nexus we are using autoversion.nextversion as the version, it will decide the version on the basis of branch , on which the build is triggered.

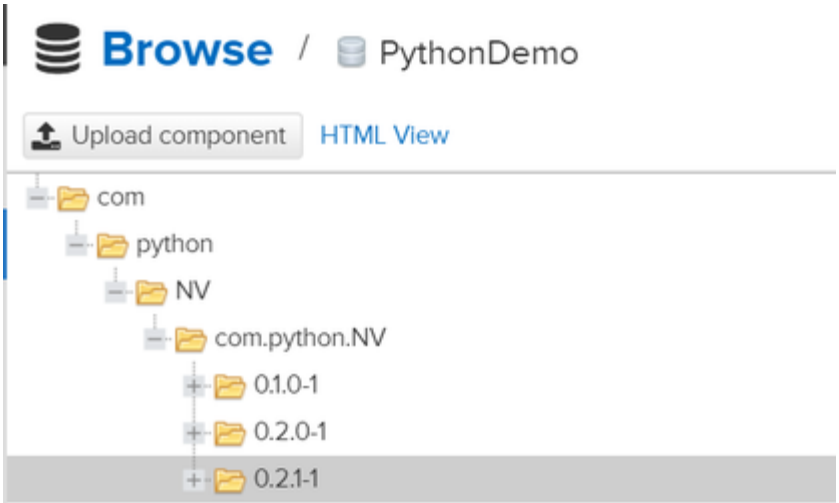
- When build is triggered for the first time on master, it will take **0.1.0** as the default version(which can be overrided by passing first_version while calling the library autoversion).



- When build is triggered on the Development branch, it will read the current version as **0.1.0** and publish the artifact using the next version as **0.2.0**



- When the build is triggered on feature branch, it will read the current version as **0.2.0** and publish the artifact using the next version as **0.2.1**



Note :- -1 in the end of version is the build number from Jenkins.