Internship Report

Topic: Automation of SVN Tag Merging

Student: Prateek Bhatnagar

Matriculation ID: 4805056

Duration: 01.02.2019 – 31.08.2019

Workload: 80hours/month = 400 hours

Course: M.Sc. (Distributed System Engineering)

Company: IAV GmbH,Dresden

1. Abstract

This report is about the Automation of merge strategy when SVN tags are involved and later integrating that with Jenkins automation pipeline.

2. Introduction

SVN is a software version and revision control system from apache. silkSVN is a command line tool for accessing this repository. There are many features for version control systems but focus of this report is on Branching and tagging.

In a multi-user environment one want to isolate respective (team/respective development) changes onto a separate line of development. This isolation of a line is knows as branch.

Branch can be used to develop experiment or change existing features without disturbing the main actual line of development. Once the feature is successful and most importantly stable then this development line can be merged back to the main branch (which is often called as Trunk)

Other important features is to mark a particular revision (e.g. a so and so release version). This help us to recreate a certain build or environment when needed. This process is known as tagging.

SVN lacks special commands to differentiate branching and tagging, hence it uses what we call as 'cheap-copies'. Cheap copies are similar to hard links in UNIX , which means instead of making a complete copy in the repository ,an internal link pointing to a specific tree/revision is created , hence creating a branch or tag are easy to create and doesn't take up much space in the repository.

Once we have all the things in place one may want to merge back to main line of development. However, inside an IT company, there are multiple branches simultaneously running and each day/week new tags are created. Now the actual problem starts, when one want to merge new tags released to a specific branch.

Auto merging of respective branches is usually done with the help of Jenkins. Jenkins is an open source automation sever written in JAVA. It automates software development process with continuous integration.

Let's define the actual problem in next section.

3. Problem

3.1 Definition

1. The main integration have different PRL (Product Release Lines), which is something like a branch for software.

2. The PRL for MIB (Modularer Infotainment Baukasten i.e. Modular Infotainment Toolbox) is maintained by company X and PRL for CNS is maintained by IAV.
3. The CNS build (or PRL) is based on MIB build/PRL (which is maintained by company X), so CNS build is a copy of MIB with new features and functionalities.
4. The component which are not branched for CNS, merge can smoothly take place.
5. But the real challenge starts when someone already using CNS branched component wants to merge from MIB and parallel new features are getting developed for that component.
6. If we have one PRL to check for it's still no problem, but we have many PRL for MIB, which makes it very difficult to understand which source to take so that changes can be taken over from it.
7. Hence, we need to automatically determine the source (tag) for the merge.

3.2 High level Solution

A new version of releasenotes.txt file is generated every day along with the build. There is a 'new' flag which indicate new issues from last build to actual build. But software is based on customer release build, which is the last build of a week. Hence 'new' flag cannot be directly used as it will describe the changes from the last build, not from the build last week. Merge is scheduled every week not every day. This can be done as follows:-

1. Monitor releaseNotes.txt for configured components (this is done using Jenkins)
2. Check the tag version.
3. Then Check last integrated tag version
4. If tag has already been merged do nothing
5. If tag has not been merged yet, do the merge.

3.3 Merge Process in General

Merging is an easy process, but automation around have following challenges is not that easy:-

1. To determine correct source
2. Remembering last merged version
3. Check if merge already took place as no one wants to merge same changes again.
4. Merge should be started only when some changes occurred.
5. To merge to take place number of Jenkins jobs should be started.

Merge process in general have following steps

1. Checkout: - if one decide to merge, one need to checkout main branch (current state) which is the target and the source is the tag version determined from the release notes.
2. Merge: - Then one need to merge new tag changes to branch.
3. If merge is successful, it is done.
4. If not one needs to manually resolve all the changes by doing checkout locally
5. Test: - After merging one need to run some sanity tests to make sure merge has happened as per the expectation.

6. If test is not successful one need to repeat all the steps and see what is failing , resolve it and again try the merge
7. Commit: - Finally, if test is successful one finally commit merged changes to the branch.

4. Solution

4.1 Jenkins

For the continuous monitoring if something has been changed in SVN folder Jenkins tool can be used.If there is a change in the folder then auto launcher jobs for each PRL (maintained by IAV) is launched and hence if release notes has already been merge then nothing is triggered

This merge process is automated using Jenkins as shown in figure below

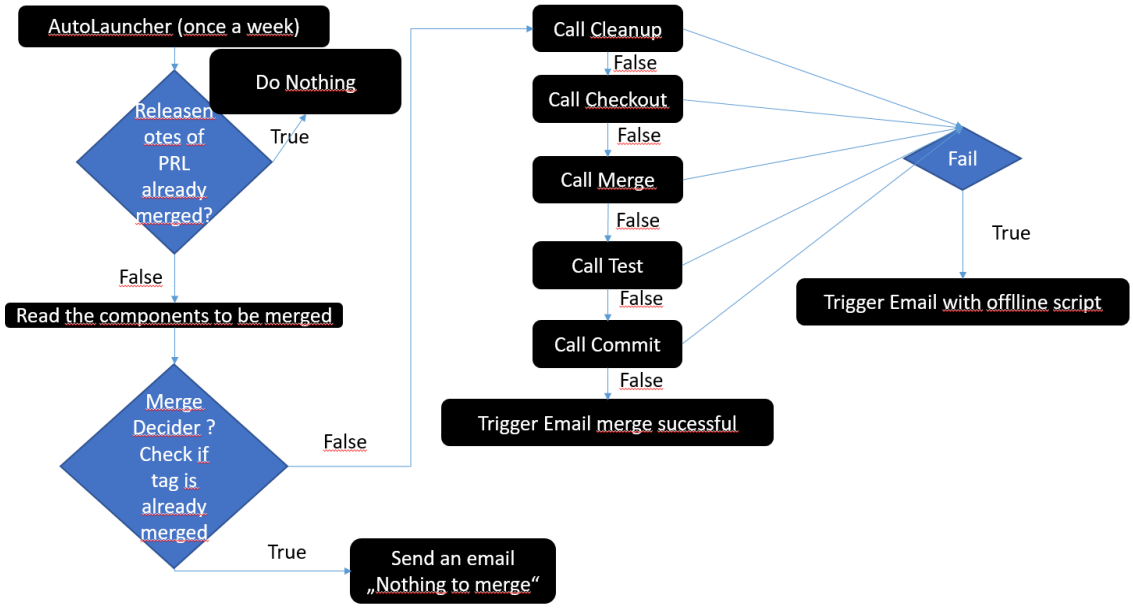

Figure 1

1. AutoLauncher is one of the Jenkins job which continuously monitor if releasenotes of a PRL are already merged.
2. If it's merged already do nothing.
3. If something is changed individual components to be merge are read from the file along with respective flags (if any for example skip cleanup , skip test etc) and then a Jenkins pipeline from cleanup to commit is executed
4. In this pipeline at each stage failure is checked , if failed then email is triggered with respective offline script attachment
5. Each stage is executed one after the other until it finishes.

Auto Launcher

S	W	Name ↓	Last Success	Last Failure	Last Duration
		MERGE Auto-Launcher CNS3	17 days - #121	21 days - #117	12 sec
		MERGE Auto-Launcher CNS3_37w_Trunk projects to 37w base	14 days - #5	14 days - #3	18 sec
		MERGE Auto-Launcher CNS_Trunk projects to 37w base	3 days 5 hr - #83	21 days - #64	17 sec
		MERGE Auto-Launcher CNS_Trunk projects to ICAS base	14 days - #5	14 days - #3	19 sec
		MERGE Test-Launcher CNS3 (merge_test_bjhommel)	1 mo 5 days - #26	N/A	11 sec
		MERGE Test-Launcher CNS_Trunk (merge_test_bjhommel)	1 mo 8 days - #4	N/A	11 sec

Figure 2

S	W	Name ↓	Last Success	Last Failure	Last Duration
		MERGE 000 Prepare	3 days 5 hr - #389	1 mo 13 days - #340	10 sec
		MERGE 010 Merge Decider	3 days 5 hr - #427	3 days 6 hr - #422	4,2 sec
		MERGE 050 Cleanup	2 days 2 hr - #650	1 mo 8 days - #551	1,4 sec
		MERGE 100 Checkout	2 days 2 hr - #634	15 days - #595	40 sec
		MERGE 200 Merge	2 days 2 hr - #530	2 days 2 hr - #528	5,9 sec
		MERGE 300 Test	2 days 2 hr - #346	2 days 2 hr - #347	1 min 16 sec
		MERGE 350 CleanMerge	2 days 2 hr - #45	1 mo 0 days - #23	9,8 sec
		MERGE 400 Commit	2 days 2 hr - #150	3 mo 23 days - #26	9 sec
		MERGE 500 Send Success Mail	2 days 2 hr - #101	N/A	0,22 sec
		MERGE 600 Remember Revision	2 days 2 hr - #94	2 mo 10 days - #2	6,1 sec

Figure 3

4.2 Parsing releaseNote.txt using python

Based on the project, location of releaseNote.txt is already known and from above step; one know something got changed.

In this step parsing of releasenote.txt takes place where in a python script is developed which accepts two arguments one is the releasenote.txt and other is config file.

Config file contain which component are to be merged.
Output of python script is a text file with :-

1. checkout path to be used to checkout (every project has different checkout path).
2. Full tag path to be checkout
3. Some extra flags to be used during merge, test and commit phase.
4. Email id/s to be triggered about the outcome.

4.3 Integration with ANT

Jenkins pipeline can be configured using many ways but here focus is to configure using ANT. Next step is to integrate ANT with python and finding a way to store python script output to workspace so that later it can be used to do merge.

ANT has a property ‘exec’, which helps us to run any executable shown below

```
<exec executable="python" failonerror="true" output="${env.WORKSPACE}/test.txt" error="error.txt">
        <arg line='merge.py "${release.notes.path}" ${config.file} ${source.url.rev} ${target.url.rev}' />
    </exec>
```

Using this property, we can execute python script and save its output in a file and save it in a workspace of respective job.

4.4 Integration with Groovy

In our Jenkins jobs now, one have the parsed output in a text file. Many things can be used in build action of a Jenkins job but here groovy is chosen because of its easiness and flexibility it provide us. With the help of groovy, text file is read and parsed, new parameterized jobs (for each component) are prepared and triggered.

4.5 Enhancement for local scripts

Merge can fail if SVN is not able to do auto merge due to conflicting changes and in this case we need to resolve merge conflicts manually. Hence, Jenkins jobs will also fail.
To ease the process of merging manually, if merge fails an email is triggered and batch scripts are attached with that email. User can execute these scripts on their respective desktops, which does the cleanup, checkout of source and target, and merge. Now the user can manually resolve the conflicts and commit those.

A similar kind of script is available for test also as many times test can also fail.

4.6 Sample Emails from Jenkins

1. Success Emails will looks like:-

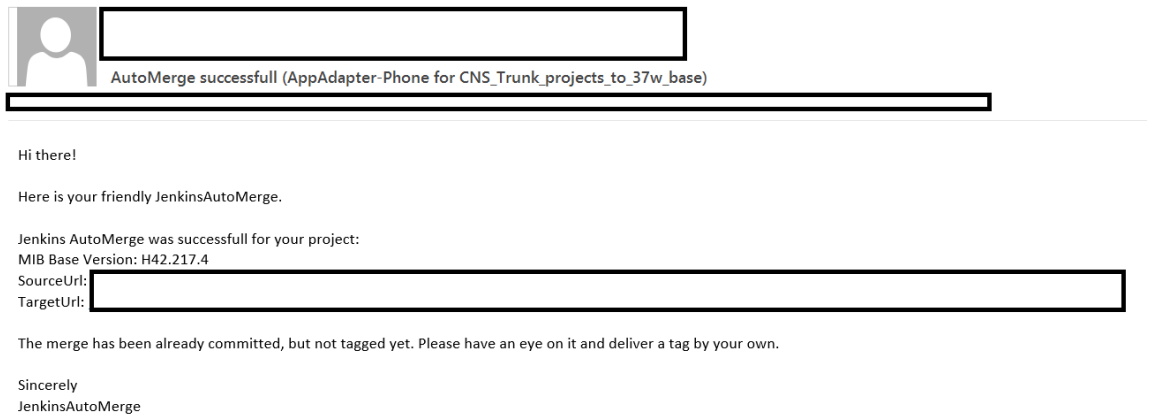

Figure 4

2. Failure Merge/Test Email looks like:-

With the emails few scripts are attached for local resolution and after resolving you can retrigger the job using the link shared in the email.

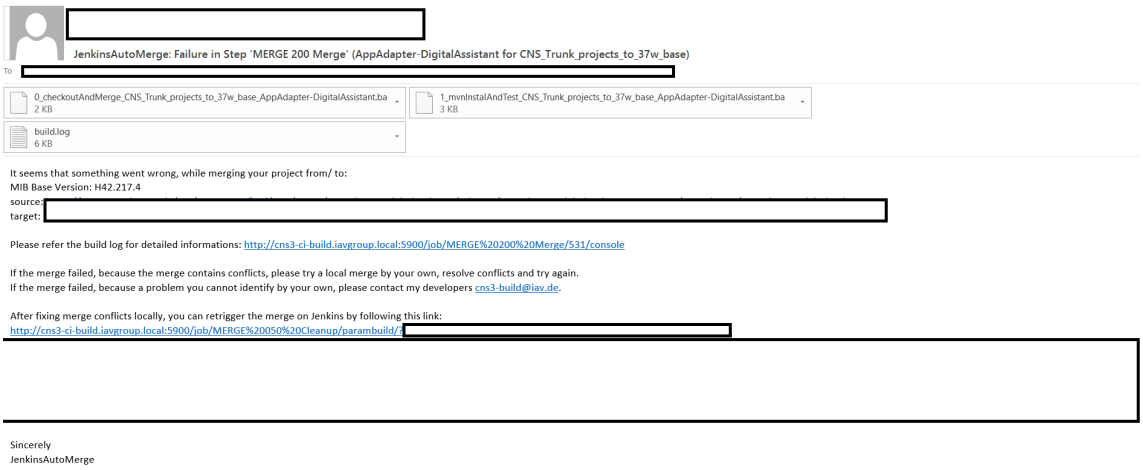

Figure 5

5. Conclusion

Earlier every week one Engineer from the respective team/project has to manually merge the components for whole team, it sometimes takes more than a day to merge successfully.

With the help of this automated merging of SVN tagging scripts on Jenkins lot of teams are able to automate their merging process, saves lot of time for an individual engineer and team as well

More such kind of enhancement with project specific requirement are work in progress on Jenkins. One such ongoing development is ‘Automatic email notification upon model change’.

6. Mentors

This work has been completed under the guidance of:-

1. Teresiak, Andreas (andreas.teresiak@iav.de)
2. Hommel, Jonas(jonas.hommel@iav.de)

7. Appendix A

7.1 Python code

```
import re
import sys

# The version number from command line Arguments
```

```
tagPart = {}
fixedSourcePart = 'https://cns3-certs.joomo.de/svn'
fixedPart = 'https://cns3-
certs.joomo.de/svn/cns3extern/hmi/tags/teams/'
componentPart = {}
folderPart = {}
sourcePath = {}
targetPath = {}
componentList = []
mPath = {}
pom = {}
email = {}
newFlag = {}
pomLevel = {}

def parseConfig():
component = ''

# f = open("CNS3.txt", "r")

f = open(sys.argv[2], 'r')
for x in f:
#To ignore comments
if (x.startswith('#', 0) != 1) and (re.match(r'\w', x)):
j = 1
for i in x.split('|'):
if j == 1:
component = i.strip()
componentList.append(component)
elif j == 2:
mPath[component] = i
elif j == 3:
pom[component] = i
elif j == 4:
email[component] = i
elif j == 5:
pomLevel[component] = i.strip()
elif j == 6:
newFlag[component] = i.strip()
j = j + 1


#printConfig()

def printConfig():
for (x, y) in mPath.items():
print (x, y)
for (x, y) in pom.items():
print (x, y)
for (x, y) in email.items():
print (x, y)
for (x, y) in newFlag.items():
print (x, y)
for (x, y) in pomLevel.items():
print (x, y)

# This function extract the component wise tag number from O drive and
```

```
populate arrays with component
# such as for ASL-Sound = ASL-Sound_2019.x.x

def get_tag():

# inputFilename = "O:\MIB-WebDAV\WebDAV\\" + version + "\ReleaseNotes-
baseline-info.txt"
sstring =""
inputFilename = sys.argv[1]
t = ""
# inputFilename = "O:\MIB-WebDAV\WebDAV\H40.73.65\ReleaseNotes-
baseline-info.txt"

for component in componentList:
f = open(inputFilename, 'r')
for x in f:
if component in x and newFlag[component] in x:
if 'ASL' in component and 'ASL-Tooling' not in component:
folderPart[component] = 'ASL'
sstring = folderPart[component] + '/' + component
elif 'AppAdapter' in component and 'ASL-Tooling' not in component:
folderPart[component] = 'AppAdapter'
sstring = folderPart[component] + '/' + component
else:
folderPart[component] = ''
sstring = component
x = x.strip()
b = re.search(r"(" + sstring + "\s+(.*)(\s:)\s(.*))", x)
#print b.groups
if b:
if 'mib2' in b.group(2):
t = re.sub('mib2', 'cns3extern', b.group(2))
elif 'mibextern' in b.group(2):
t = re.sub('mibextern', 'cns3extern', b.group(2))
tagPart[component] = t


def form_path():
for c in componentList:
if mPath[c].rfind("http") != -1 :
path=mPath[c]
else:
path = fixedPart.replace('tags', 'branches') + c \
                + '/release-branches/' + mPath[c] + '/' + folderPart[c]
\
                + '/' + c
targetPath[c] = path
i = c
if i == 'AppAdapter-Phone':
i = 'Appadapter-Phone'
if c in tagPart:
path = fixedSourcePart + tagPart[c]
else:
path = 'No New code to Merge'
targetPath[c] = path
sourcePath[c] = path
```

```
# After extracting tag this function makes URL to merge such as
# https://cns3-token.joomo.de/svn/cns3extern/hmi/tags/teams/ASL-
Sound/releases/ASL-Sound_2019.8.0/ASL/

def prepareSourcePath():
parseConfig()
#printConfig()
get_tag()
form_path()


def run():
prepareSourcePath()


run()
for component in componentList:
print component + '#' + sourcePath[component] + '#' \
        + targetPath[component] + '#' + pom[component] + '#' \
        + email[component] + '#' + pomLevel[component] +  '\n'
```

7.2 Ant Script Modification

```
<target name="GetIntegratedComponent">
        <exec executable="python" failonerror="true"
output="${env.WORKSPACE}/test.txt" error="error.txt">
            <arg line='merge.py "${release.notes.path}" ${config.file}
${source.url.rev} ${target.url.rev}' />
        </exec>
      <loadfile property="mytext" srcFile="${env.WORKSPACE}/test.txt"/>
        <echo>${mytext}</echo>
</target>
```

7.3 Groovy Script to trigger other sub-jobs

```
import groovy.util.XmlSlurper
import java.util.Map
import jenkins.*
import jenkins.model.*
import hudson.*
import hudson.model.*
import hudson.console.HyperlinkNote

def component
def source
def target
def words
def testPath
def emails
def pomLevel
def params = []

//test.txt is created by python and ant script and contains output of
source url and target url
```

```
EnvVars envVars = build.getEnvironment(listener);

filename = envVars.get('WORKSPACE') + "\\test.txt";
println filename

//Fetch current thread to get current parameters for the jenkins job
def currentBuild = Thread.currentThread().executable
def currentParams = currentBuild.getAction(ParametersAction.class)

new File(filename).eachLine {  line ->
    if (line.trim().size() != 0) {
        words = line.split('#')
        component = words[0]
        println component
        source = words[1]
        println source
        target = words[2]
        println target
        testPath = words[3]
        println testPath
        emails = words[4]
        println emails
        pomLevel = words[5]
        println pomLevel
        params = []
        // if new flag is set , then there is no new code to merge then
, we dont need to run jenkins jobs
        if (source != "No New code to Merge") {
            def job = Hudson.instance.getJob('MERGE 010 Merge Decider')
            //Extract all the current parameters to pass on to next job
in pipeline
            for (ParameterValue p in currentParams) {
                switch (p.name) {
                    case 'Local.Path':
                        println "the local path is " + p.value
                        params.add(new StringParameterValue(p.name,
p.value + "/" + component))
                        break
                    case 'skip.merge.conflicts':
                        println "the skip.merge.conflicts is " +
p.value
                        params.add(new BooleanParameterValue(p.name,
p.value))
                        break
                    case 'cleanup':
                        println "cleanup " + p.value
                        params.add(new BooleanParameterValue(p.name,
p.value))
                        break
                    case 'commit':
                        println "commit " + p.value
                        params.add(new BooleanParameterValue(p.name,
p.value))
                        break
                    case 'PRL.name':
                        println "PRL " + p.value
                        params.add(new StringParameterValue(p.name,
```

```
p.value))
                        break
                    case 'H.Version':
                        println "PRL " + p.value
                        params.add(new StringParameterValue(p.name,
p.value))
                        break
                    case 'POMs.url':
                        params.add(new StringParameterValue(p.name,
p.value))
                }
            }

            params.add(new StringParameterValue('Merge.Source.Url',
source))
            params.add(new StringParameterValue('Merge.Target.Url',
target))
            params.add(new StringParameterValue('projects.to.test',
testPath))
            params.add(new StringParameterValue('Email.List', emails))
            params.add(new StringParameterValue('Project.Name',
component))
            params.add(new
StringParameterValue('link.POMs.oneLevelDeeper', pomLevel))

            def paramsAction = new ParametersAction(params)
            def cause = new hudson.model.Cause.UpstreamCause(build)
            def causeAction = new hudson.model.CauseAction(cause)
            def future = job.scheduleBuild2(0, causeAction,
paramsAction)
            println "Waiting for the completion of " +
HyperlinkNote.encodeTo('/' + job.url, job.fullDisplayName)
            anotherBuild = future.get()
        }
    }
}
```

7.4 Groovy Script for offline merge/test

```
import groovy.util.XmlSlurper
import java.util.Map
import jenkins.*
import jenkins.model.*
import hudson.*
import hudson.model.*
import hudson.console.HyperlinkNote
EnvVars envVars = build.getEnvironment(listener);

def currentBuild = Thread.currentThread().executable
def currentParams = currentBuild.getAction(ParametersAction.class)

def component
def project_to_test
def source
def target
def path
```

```
def pri
def pomUrl
def pomLevel
def pomPath

//Get all the parameter from the job
for (ParameterValue p in currentParams) {
    switch (p.name) {
        case 'Local.Path':
            path=p.value
            break
        case 'Merge.Source.Url':
            source=p.value.replaceAll('certs','token')
            break
        case 'Merge.Target.Url':
            target=p.value.replaceAll('certs','token')
            break
        case 'Project.Name':
            component=p.value
            break
        case 'PRL.name':
            pri=p.value
            break
        case 'POMs.url':
            pomUrl=p.value.replaceAll('certs','token')
            break
        case 'projects.to.test':
            project_to_test=p.value
            break
        case 'link.POMs.oneLevelDeeper':
            println "p.value=" + p.value
            pomLevel=p.value
            break
    }
}
//Prepare the local path to checkout , merge and test
def delPath = "D:/JenkinsAutoMerge/"
path = delPath + pri + "/" + component

//Create a new file to be executed for checkout and marge at current
workspace
def newFile = new File(envVars.get('WORKSPACE') +
"\\0_checkoutAndMerge_" + pri + "_" + component + ".ba")
newFile.createNewFile()
def filename = envVars.get('WORKSPACE') +  "\\0_checkoutAndMerge_" +
pri + "_" + component + ".ba"

println
"**********************************************************************
***************"
println "Generated 0_checkoutAndMerge file at"
println filename
println
"**********************************************************************
***************"
newFile.write("\n")
```

```
if (pomLevel) {
    pomPath = delPath + pri + "/../POMs"
    println "link.POMs.oneLevelDeeper was true " + pomPath
} else {
    pomPath = delPath + pri + "/POMs"
    println "link.POMs.oneLevelDeeper was false " + pomPath
}

def sourcePath = path + "/source"
def targetPath = path + "/target"

//Batch script to cleanup ,checkout POMs , source and target and later
merge
newFile.append("echo deleting ..." + path + "\n")
newFile.append("rmdir /Q /S " + "\"" + path + "\"" + "\n")
newFile.append("svn co " + pomUrl + " " + pomPath + "\n")
newFile.append("svn co " + source + " " + sourcePath + " --ignore-
externals \n")
newFile.append("svn co " + target + " " + targetPath + " --ignore-
externals \n")
newFile.append("svn merge -r 0:HEAD " + sourcePath + " " + targetPath +
"\n")
newFile.append("PAUSE")

//Create a new file to be executed for mvn install and test
newFile = new File(envVars.get('WORKSPACE') + "\\1_mvnInstalAndTest_" +
pri + "_" + component + ".ba")
newFile.createNewFile()
filename = envVars.get('WORKSPACE') +  "\\1_mvnInstalAndTest_" + pri +
"_" + component + ".ba"

println
"**********************************************************************
***************"
println "Generated 1_mvnInstalAndTest file at"
println filename
println
"**********************************************************************
***************"
newFile.write("\n")

//Batch script to mvn install and test along with error handling
switch(project_to_test) {
    case '-self-':
        newFile.append("call mvn3.bat install -f" + targetPath +
"/pom.xml -Dmaven.test.skip=true -Dmaven.repo.local=" + path + "/.repo
\n")
        newFile.append("IF NOT %ERRORLEVEL%==0 GOTO INSTALLATION_ERRORS
\n")

        newFile.append("call mvn3.bat test -f" + targetPath + "/pom.xml
-Dmaven.repo.local=" + path + "/.repo \n")
        newFile.append("IF NOT %ERRORLEVEL%==0 GOTO TESTING_ERRORS \n")
        break
    case '-aggregates-':
        newFile.append("call mvn3.bat install -f" + targetPath + "/" +
```

```
"Aggregates/Install/pom.xml -Dmaven.test.skip=true -Dmaven.repo.local="
+ path + "/.repo \n")
        newFile.append("IF NOT %ERRORLEVEL%==0 GOTO INSTALLATION_ERRORS
\n")

        newFile.append("call mvn3.bat test -f" + targetPath + "/"  +
"Aggregates/Test/pom.xml -Dmaven.repo.local=" + path + "/.repo \n")
        newFile.append("IF NOT %ERRORLEVEL%==0 GOTO TESTING_ERRORS \n")
        break
    default:
        for (i in project_to_test.split(',')) {
            newFile.append("call mvn3.bat install -f" + targetPath +
"/" + i + "/pom.xml -Dmaven.test.skip=true -Dmaven.repo.local=" + path
+ "/.repo \n")
            newFile.append("IF NOT %ERRORLEVEL%==0 GOTO
INSTALLATION_ERRORS \n")
        }
        for (i in project_to_test.split(',')) {
            newFile.append("call mvn3.bat test -f" + targetPath + "/" +
i + "/pom.xml -Dmaven.repo.local=" + path + "/.repo \n")
            newFile.append("IF NOT %ERRORLEVEL%==0 GOTO TESTING_ERRORS
\n")

        }
}


//Error Handling if mvn install or test fails
newFile.append(":INSTALLATION_ERRORS \n")
newFile.append("echo Process stops because of maven install errors \n")
newFile.append(":END \n")
newFile.append("GOTO PAUSE_LABEL \n")

newFile.append(":TESTING_ERRORS \n")
newFile.append("echo Process stops because of maven test errors \n")
newFile.append(":END \n")
newFile.append("GOTO PAUSE_LABEL \n")

newFile.append(":PAUSE_LABEL \n")
newFile.append("PAUSE \n")
newFile.append(":END \n")
```