How to - Build and test R repository.

Pre-requisites for Building and testing R repository.

1. Repository for hosting the code Bitbucket (or any other like gitlab/github)
2. Jenkins to build the repository.
3. R interpreter To compile,build and package the code repository.
4. Artifactory To store the R packages created by the interpreter.

Necessary folder structure needed for Repository.

Requirements of an ideal folder structure in R.

The folder structure should contain the respective folders

1. Rscripts To segregate the details of all the scripts associated with the project.
2. Data Data folder to hold the project data information such as images , excel sheets etc.
3. Reports Reports required to add the details of various reports associated with the project.
4. Library To create a virtual environment which would reserve the packages in the folder without any dependency from external resources.
5. R Basic functionality of the project.
6. man To describe the functionalities of various function.
7. tests To preserve test cases and execute the test cases over the code repositories.
8. .Rprofile It is like a configuration file which would help to redirect the packages to be installed on the repository/local folder.
9. The R project settings are provided in the repository (Rproj file).
10. DESCRIPTION The description file provides the details of package version of the code (Author,License,Type and also Encoding).

Significance of Dev tools and installation of packages.

Devtools is a package which would enable the user to install packages from github.

Example :

```
install.packages(devtools)

devtools::install_github("hadley/devtools")
```

The above would help the user to install packages from a hosted repository(github/ Bitbucket).

Work with various packages and install the packages directly from the repository including repository.

Process to test R code/functions on the repository.

The process required to test R code repository :

Functions required to test various functions and functionalities of the code repositories.

```
library  testthat

Functions available for testthat library

auto_test_package  Watches a package for changes, rerunning tests as appropriate.
capture_output     Capture output to console
expect             The building block of all expect_ functions
describe           describe: a BDD testing language
expect_setequal    Expectation: do two vectors contain the same values?
expect_success     Tools for testing expectations
compare_state      Compare two directory states
not                Negate an expectation
```

Steps to execute the testtest cases:

Step 1:

Install the respective packages associated with the library

```
install.packages("package")
```

step 2:

import the respective library to the test script

command used to import the library

```
library("testthat")
```

steps 3:

Use the respective test functions to execute the test cases on the code repositories.

Examples:

```
test_that("ouputs should match", {

expect_equal(new.function(2,3),6)

expect_equal(new.function(2,3),1)

expect_equal(new.function(2,3),2)

expect_equal(new.function(4,5),20)

expect_equal(new.function(1,1),1)

expect_equal(new.function(1,4),4)

expect_equal(tan(pi / 4), 1)

}) //expect_equal to run whether the test cases expect equal results or
not
```

End - End setup available for the R execution

The setup has various stages as mentioned below

Stages				
Poll SCM	**Compile**	**Test Execution**	**Package**	**Deploy**

Description of Various stages

Stage 1 Poll SCM Poll SCM pulls the necessary changes provided by the developer in the repository.

Poll SCM is the stage mentioned in the script to pull the latest changes in the repository and checkout the latest commit mentioned in the repository (Bitbucket)

Stage 2 Compile The code is compiled by the respective interpreter present in the respective system/Jenkins/Docker image

Compilation of the code is done by the interpreter present in the slave to compile the code and check if any syntax errors are present in the code.

Stage 3 Test Execution The tests executed needs to have a flag to determine the status of the stage to act as an intermediary for the next stages to perform.

Test Execution is part of the Code which would help in testing the efficiency of the code and publish the status through the Test Execution Stage.

Note : We have added a new feature to fail the build as soon as the stage fails.

Stage 4 Package The package stage converts the respective code to package format such as .tar file.

Command for packaging "**R CMD build Rmul/R --save**" --save to save the respective package into a specific location.

Stage 5 Deploy The Deploy stage pushes the respective package to the Artifactory.

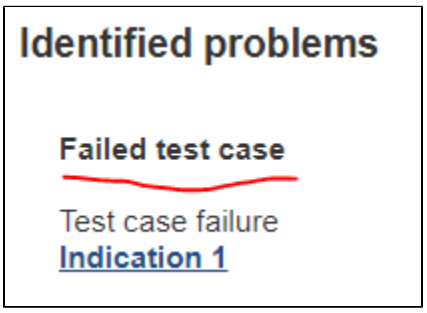
The deploy stage pushes the respective .tar file into nexus artifactory. The credentials , folder structure, version details and build number.

JENKINS SCRIPT :

```
node ('slave1') {
stage("POLL SCM") {

git credentialsId: 'fefe1963-b4e0-4d09-bd5b-f8a95b934ce0', url: 'http://novartis.devops.altimetrik.io:7990/scm/rtes/r-mul.git'
} // **To pull the new changes in the repository and checkout the latest commit **//



stage("compile") {
sh label: '', script: 'chmod 777 Rmul/R/test.R'
sh label: '', script: 'Rscript Rmul/R/test.R'
} // **To provide the respective permissions to the repository folder and also execute the script** //

stage("Test Execution") {
try { sh 'grep -f $PWD/errorfile.txt'}
catch (err) {
echo "Caught: ${err}"
if (!continueBuild) {
currentBuild.result = 'ABORTED'
error('Stopping early…')
}
}//**To provide the  details of Test execution for the next stages to proceed or to halt at the particular stage**//

stage("Package") {
sh label: '', script: 'R CMD build Rmul/R --save'
} //**To package the particular code available**//

stage("Deploy"){
nexusArtifactUploader artifacts: [[artifactId: 'com.nova', classifier: '', file: 'Rmul_0.1.0.tar.gz', type: 'tar.gz']], credentialsId: 'nexus',
groupId: 'com.nova', nexusUrl: 'novartis.devops.altimetrik.io:8082', nexusVersion: 'nexus3', protocol: 'http', repository: 'Rproject',
version: '{$BUILD_NUMBER}'
}   //** To Deploy the new artifacts to nexus artifactory**//
}
}
}
```

Special features used in the Jenkins_pipeline_R for compiling ,building and packaging R code http://novartis.devops.altimetrik.io:8084/job/POC/job/Jenkins_pipeline_R/

- Build status fails if the respective stage fails using the stage "Test Stage".
- Test failure is assessed by the Build failure Analyzer plugin such as test case failure.



- Incorporated a new file (errorfile.txt) during the Jenkins run time which would help as a temporary file to determine the status of the Job in Jenkins.
-

```
if(count<8)
{
   file.create("errorfile.txt")
   print("Job failure due to test cases")
} else
{
  print("cases passed")
}
```

Note : The above count specifies the details of test cases passed for a total of ten test cases.

The file created is a temporary variable to specify the list of test cases are passed or not.

Script used to test the functionality of the code/function.

```
count=0
a <- c(0:10)
b <- c(1:5)
c <- c(1:3)

if( (min(new.function(a,b))>0) && (min(new.function(b,c))>0) && (min(new.function(c,a)>0)) && (max(new.function(c,a)>51)) && (max(new.function(a,b)>51)) && (max(new.function(c,a)>5
{
    print("test case passed")
    count=count+1
} else
{
    print("test case failed")
}

if((max(new.function(c,a)>51)) && (max(new.function(a,b)>51)) && (max(new.function(c,a)>51)) == TRUE)
{
    print("test case passed")
    count=count+1
} else
{
    print("test case failed")
```

The above piece of script tests the functionality of the code in the repository.

The variables a,b,c are assigned with list of values between (0-10) to test the multiplication function in the code mul.R.

Count is a variable to check if each of the test case has failed or passed.