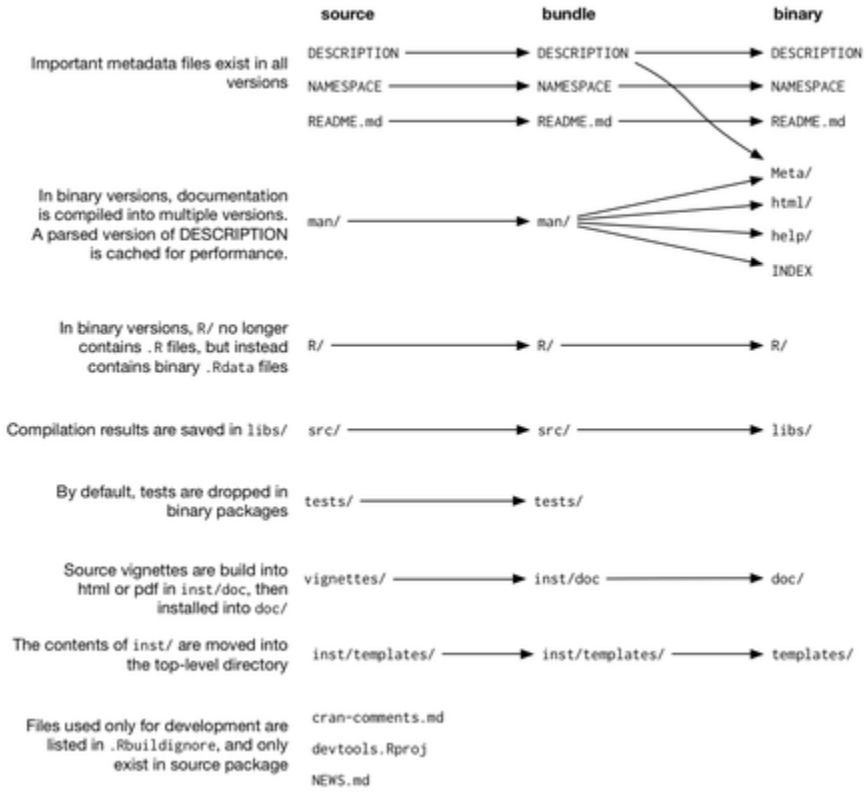
How To - Versioning Strategy for R

Before Starting the Versioning strategy, each R repo must follow the standards provided here for proper packaging. See here for more info http://r-pkgs.had.co.nz/

The following diagram shows the file/folder structure in the root directory for source, bundled and binary versions of devtools.



Each R repo has a DESCRIPTION file that stores the meta file for that given R package. The DESCRIPTION file contains the following contents.

```
Package: mypackage
Title: What The Package Does (one line, title case
required)
Version: 0.1
Authors@R: person("First", "Last", email =
"first.last@example.com",
role = c("aut", "cre"))
Description: What the package does (one paragraph)
Depends: R (>= 3.1.0)
License: What license is it under?
LazyData: true
```

Package: The package name, should be similar to the repo name.

Title: is a one line description of the package, and is often shown in package listing. It should be plain text (no markup), capitalised like a title, and NOT end in a period. Keep it short: listings will often truncate the title to 65 characters.

Version: The version of the this package.

Authors: Full name with email and role preferential.

Description: The description of a package is usually long, spanning multiple lines. The second and subsequent lines should be indented, usually with four spaces.

Depends: to require a specific version of R, e.g. Depends: R (>= 3.0.1). As with packages, it's a good idea to play it safe and require a version greater than or equal to the version you're currently using. devtools::create() will do this for you.

License: can be either a standard abbreviation for an open source license, like GPL-2 or BSD, or a pointer to a file containing more information, file LICENSE.

LazyData: makes it easier to access data in your package. Because it's so important, it's included in the minimal description created by devtools. It's described in more detail in external data.

Versioning Strategy: `<major>.<minor>.<patch>`

Formally, an R package version is a sequence of at least two integers separated by either `.` or `-`. For example, `1.0` and `0.9.1-10` are valid versions, but `1` or `1.0-devel` are not. You can parse a version number with `numeric_version`.

```
numeric_version("1.9") == numeric_version("1.9.0")
```

```
## [1] TRUE
```

```
numeric_version("1.9.0") < numeric_version("1.10.0")
```

```
## [1] TRUE
```

For example, a package might have a version 1.9. This version number is considered by R to be the same as 1.9.0, less than version 1.9.2, and all of these are less than version 1.10 (which is version "one point ten", not "one point one zero). R uses version numbers to determine whether package dependencies are satisfied. A package might, for example, import package `devtools (>= 1.9.2)`, in which case version 1.9 or 1.9.0 wouldn't work.

The version number of your package increases with subsequent releases of a package, but it's more than just an incrementing counter – the way the number changes with each release can convey information about what kind of changes are in the package.

Always use `.` to separate version numbers.

- A released version number consists of three numbers, `<major>.<minor>.<patch>`. For version number 1.9.2, 1 is the major number, 9 is the minor number, and 2 is the patch number. Never use versions like `1.0`, instead always spell out the three components, `1.0.0`.
- An in-development package has a fourth component: the development version. This should start at 9000. For example, the first version of the package should be `0.0.0.9000`. There are two reasons for this recommendation: first, it makes it easy to see if a package is released or in-development, and the use of the fourth place means that you're not limited to what the next version will be. `0.0.1`, `0.1.0` and `1.0.0` are all greater than `0.0.0.9000`.

  Increment the development version, e.g. from `9000` to `9001` if you've added an important feature that another development package needs to depend on.

  If you're using svn, instead of using the arbitrary `9000`, you can embed the sequential revision identifier.

This advice here is inspired in part by Semantic Versioning and by the X.Org versioning schemes.

CICD Recommendation

```
Package: mypackage
Title: What The Package Does (one line, title case
required)
Version: $Version Authors@R: person("First", "Last",
email = "first.last@example.com",  role = c("aut", "cre")
) Description: What the package does (one paragraph) Depe
nds: R (>= 3.1.0) License: What license is it under? Lazy
Data: true
```

For any Jira story that is attached to a new feature, the major version can be auto-incremented. i.e. <major + 1>.<minor>.<patch>

```
case 'major':
     Version = "${major + 1}.0.0"
break
For CICD pipeline of R package, the pipeline orchestrator can be set to auto-increment the minor builds for
each release. i.e <major>.<minor + 1 >.<patch> case 'minor': Version = "${major}.${minor + 1}.
break
```

For any Jira Bug story, the pipeline can be configured to increment the patch number keeping the major/minor. i.e. `<major>.<minor>.<patch + 1>`

```
case 'patch':
Version = "${major}.${minor}.${patch + 1}"
break
```

Use this sample for to get version from the tag and auto-increment it based on each case.

Error rendering macro 'code': Invalid value specified for parameter 'firstline'

```
pipeline {
    agent {label 'rimage'}
    stages {
        stage("POLL SCM") {
            steps {
            echo "Poll SCM"
            }
        }
        stage("Build") {
            steps {
                script {
                    version = nextVersionFromGit()
                    sh "sed -i '/Version/c\\Version: $version' DESCRIPTION"
                    sh "R CMD build ."
                    sh "R CMD check --no-manual ."
                }
            }
        }
        stage("Publish") {
            steps {
                script {
                    withCredentials([usernamePassword(credentialsId: 'artifactory', passwordVariable:
'pass', usernameVariable: 'user')]) {
                        echo "Publish helloworld_${version}.tar.gz"
                        //updateArtifact("helloworld_${version}.tar.gz")
                        sh "curl -u ${user}:${pass} -T helloworld_${version}.tar.gz
'http://novartis.devops.altimetrik.io:8081/artifactory/r-packages/helloworld_${version}.tar.gz'"
                    }
                }
            }
        }
        stage("Deploy") {
            agent {
                label 'slave1'
            }
            steps {
                script {
                    withCredentials([usernamePassword(credentialsId: 'artifactory', passwordVariable:
'pass', usernameVariable: 'user')]) {
                        echo "Deploy helloworld_${version}.tar.gz"
                        //updateArtifact("helloworld_${version}.tar.gz")
                        sh "curl -u ${user}:${pass} -O
'http://novartis.devops.altimetrik.io:8081/artifactory/r-packages/helloworld_${version}.tar.gz'"

                        sh """
                            set +x
                            ls -lart
                            R CMD install helloworld_${version}.tar.gz helloworld
                        """
                    }
                }

            }
        }
    }
}

def nextVersionFromGit() {
    def latestVersion = sh returnStdout: true, script: "cat DESCRIPTION | grep -i version | awk '{print
\$2}' || echo 0.0.0"
    def (major, minor, patch) = latestVersion.tokenize('.').collect { it.toInteger() }
    def nextVersion
    switch (env.BRANCH_NAME) {
        case 'master':
            nextVersion = "${major + 1}.0.0"
            break
        case 'develop':
            nextVersion = "${major}.${minor + 1}.0"
            break
        default:
            nextVersion = "${major}.${minor}.${patch + 1}-${currentBuild.number}"
```

```
            break
    }
    print "Next Version"
    print nextVersion
    nextVersion
}

def getCheckSum(fileName) {
    def checkSum = sh returnStdout: true, script: "md5sum $fileName | awk '{ print \$1 }'"
    return checkSum
}

// @Grapes([
//     @Grab('org.codehaus.groovy.modules.http-builder:http-builder:0.7'),
//     @Grab('org.apache.httpcomponents:httpmime:4.5.1')
// ])
// "curl -u <user>:<pass> -T helloworld_${version}.tar.gz
'http://novartis.devops.altimetrik.io:8081/artifactory/r-packages/helloworld_${version}.tar.gz'"
// def updateArtifact(fileName) {
//     File file = new File(fileName);

//     def http = new HTTPBuilder('http://novartis.devops.altimetrik.io:8081/')

//     http.request(Method.POST) {req->
//         requestContentType: "multipart/form-data"
//         uri.path = 'artifactory/r-packages'

//         MultipartEntityBuilder multipartRequestEntity = new MultipartEntityBuilder()
//         multipartRequestEntity.addPart('file0', new FileBody(file))

//         req .entity =  multipartRequestEntity.build()

//         response.success = { resp, data ->
//             // response text
//             println data.getText()
```

```
//          }
//      }
// }
```

Source:

http://r-pkgs.had.co.nz/

http://r-pkgs.had.co.nz/description.html#description

https://semver.org/

https://www.x.org/releases/X11R7.7/doc/xorg-docs/Versions.html