Quantifying and lightening the package dependency burden

Robert Castelo
Universitat Pompeu Fabra
Barcelona
robert.castelo@upf.edu - @robertclab

BioC Developer’s Forum
February 20th, 2020
Motivation for this discussion


[Bioc-devel] how to trace 'Matrix' as package dependency for 'GenomicScores'

Robert Castelo robertc@telo@ending@rom up@edu
Thu Feb 6 12:36:08 CET 2020

- Previous message (by thread): [Bioc-devel] Account Activation Issue
- Next message (by thread): [Bioc-devel] how to trace 'Matrix' as package dependency for 'GenomicScores'
- Messages sorted by: [date] [ thread ] [ subject ] [ author ]

Hi,

when i load the package 'GenomicScores' in a clean session i see thorough the `sessionInfo()` that the package 'Matrix' is listed under 'loaded via a namespace (and not attached)'.

i'd like to know what is the dependency that 'GenomicsScores' has that ends up requiring the package 'Matrix'.
GenomicScores

R Under development (unstable) (2020-01-29 r77745)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: CentOS Linux 7 (Core)

Matrix products: default
BLAS: /opt/R-devel/lib64/R/lib/libRblas.so
LAPACK: /opt/R-devel/lib64/R/lib/libRlapack.so

locale:
LC_CTYPE=en_US.UTF-8
LC_NUMERIC=C
LC_TIME=en_US.UTF-8
LC_COLLATE=en_US.UTF-8
LC_MONETARY=en_US.UTF-8
LC_MESSAGES=en_US.UTF-8
LC_PAPER=en_US.UTF-8
LC_NAME=C
LC_ADDRESS=C
LC_TELEPHONE=C
LC_MEASUREMENT=en_US.UTF-8
LC_IDENTIFICATION=C

attached base packages:
[1] parallel  stats4   stats    graphics  grDevices  utils  datasets
[8] methods  base

other attached packages:
[1] GenomicScores_1.11.4 GenomicRanges_1.39.2 GenomeInfoDb_1.23.10
[4] IRanges_2.21.3  S4Vectors_0.25.12  BiocGenerics_0.33.0
[7] colorout_1.2.2

loaded via a namespace (and not attached):
[1] Rcpp_1.0.3
[2] shinypack_loader_0.9.0
[3] Biostrings_2.55.4
[4] digest_0.6.23
[5] BioFilecache_1.11.4
[6] R_2.4.1
[7] RSQLite_2.2.0
[8] pillar_1.4.3
[9] rlang_0.4.4
[10] data.table_1.12.8
[12] shinythemes_1.1.2
[13] BiocParallel_1.21.2
[14] htmlwidgets_1.3.1
[15] bit_1.1-15.1
[16] delayedArray_0.33.0
[17] httpuv_1.5.2
[18] pkgconfig_2.0.3
[19] tidyselect_1.0.0
[20] interactiveDisplayBase_1.25.0
[21] MatrixStats_0.55.0
[22] crayon_1.3.4
[23] dbplyr_1.1.2
[24] later_1.0.0
[25] bitops_1.0-6
[26] grid_4.6.0
[27] BH_1.1.0
[28] Vector_0.37.0
[29] VCsparse_0.2.2
[30] bit64_0.6.7
[31] Bioc-base_3.27.2
[32] BiocGenerics_0.33.0
[33] BiocGenerics_0.33.0
[34] BiocGenerics_0.33.0
[35] Bioconductor_3.9
[36] glue_1.3.1
[37] BiocVersion_3.14.4
[38] yaml_2.1.3
[39] AnnotationHub_2.19.7
[40] GenomeInfoDbData_1.2.2
[41] GenomeAlignments_1.23.1
[42] GenomicRanges_1.39.2
[43] GenomeInfoDb_1.23.10
[44] IRanges_2.21.3
[45] S4Vectors_0.25.12
[46] BiocGenerics_0.33.0
[47] colorout_1.2.2
repos <- BiocManager::repositories()[c("BioCsoft", "CRAN")]
db <- available.packages(repos=repos)
deps <- tools::package_dependencies("GenomicScores", db, recursive=TRUE)[[1]]
length(deps)
[1] 116
deps <- tools::package_dependencies(c("GenomicScores", deps), db)
g <- graph::graphNEL(nodes=names(deps), edgeL=deps, edgemode="directed")
igraph::all_simple_paths(igraph::igraph.from.graphNEL(g),
from="GenomicScores", to="Matrix", mode="out")

[[1]]
[1] GenomicScores   BSgenome     rtracklayer GenomicAlignments
[5] SummarizedExperiment DelayedArray Matrix
[[2]]
[1] GenomicScores   BSgenome     rtracklayer GenomicAlignments
[5] SummarizedExperiment Matrix
[[3]]
[1] GenomicScores   DT          crosstalk       ggplot2       mgcv
[[4]]
[4] SummarizedExperiment DelayedArray Matrix
[[5]]
[4] SummarizedExperiment Matrix
Discussion followed ..

[Bioc-devel] how to trace 'Matrix' as package dependency for 'GenomicScores'

Martin Morgan mtmorg@n@bloc @ending |rom gm@llcom
Sat Feb 8 18:01:53 CET 2020

I find it quite interesting to **identify formal strategies for removing dependencies**, but also a little outside my domain of expertise. This code

```
[...]```

shows me, via n remove, that I can remove the dependency on AnnotationHub by removing the dependency on just one package (AnnotationHub!), but to remove BioCFileCache I'd also have to remove another package (AnnotationHub, I'd guess). So this provides some measure of the ease with which a package can be removed.

I'd like a 'benefit' column, too -- if I were to remove AnnotationHub, how many additional packages would I also be able to remove, because they are present only to satisfy the dependency on AnnotationHub? More generally, perhaps there is a dependency of AnnotationHub that is only used by AnnotationHub and BSgenome. So removing AnnotationHub as a dependency would make it easier to remove BSgenome, etc. I guess this is a graph optimization problem.

Probably also worth mentioning the `itdepends` package (https://github.com/r-lib/itdepends), which I think tries primarily to determine the relationship between package dependencies and lines of code, which seems like complementary information.

Martin
Discussion followed..

[Bioc-devel] how to trace 'Matrix' as package dependency for 'GenomicScores'

Vincent Carey @vic_ending #rom ch@mnihg@be@rv@rd@edu
Sun Feb 9 13:31:24 CET 2020

- Previous message (by thread): [Bioc-devel] how to trace 'Matrix' as package dependency for 'GenomicScores'
- Next message (by thread): [Bioc-devel] how to trace 'Matrix' as package dependency for 'GenomicScores'
- Messages sorted by: [ date ] [ thread ] [ subject ] [ author ]

On Sat, Feb 8, 2020 at 12:42 PM Martin Morgan <mmorgan.bioc using gmail.com> wrote:

> I find it quite interesting to identify formal strategies for removing dependencies, but also a little outside my domain of expertise. This cedo
>
> It would be nice to collect the ideas in this thread into some recommendations. The themes I am thinking of are: how developers can make their packages robust to loss of external packages and how can the Bioc ecosystem best deal with departures of packages from itself and From CRAN. A good and well-adapted solution to the first one makes the second one moot.

Two CRAN-related events I know of that required some effort are (temporary) loss of armr and (recently) archiving of Seurat.

- Developer: How to quantify and lighten the package dependency burden.

- BioC core team: How to identify packages that compromise and are compromised by the package dependency burden.
Package itdepends - Jim Hester


- Dependencies break.
- Not all dependencies are equal (upstream dependencies, system-wide requirements, etc.).
- Users of your package: developers (lightweight and stable), end users (feature-rich).
- Widely used packages are widely tested.
- You may want to remove dependencies from your package: quantification of the “dependency weight” is critical.
- The package itdepends is meant to be a toolbox to make this type of decisions.
- The contrast between the number of functions exported from a dependency and the number of its functions that you are actually importing in your package provide a sense of how much of that package dependency you are using.
Package `itdepends`

devtools::install_github("jimhester/itdepends")

# Determine usage of dependencies for a package
itdepends::dep_usage_pkg()

# Calculate the package “weight”
itdepends::dep_weight()
Quantifying package dependency burden

```r
du <- itdepends::dep_usage_pkg("GenomicScores")

du
# A tibble: 1,333 x 2
pkg fun
<chr> <chr>
1 base standardGeneric
2 base {
3 base <-
4 utils as.person
5 base if
6 base !
7 base missing
8 base {
9 base <-
10 GenomicScores .mergeMaintainer
# ... with 1,323 more rows
```
Quantifying package dependency burden

table(du$pkg)

<table>
<thead>
<tr>
<th>Package</th>
<th>base</th>
<th>Biobase</th>
<th>BiocGenerics</th>
<th>BSgenome</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnnotationHub</td>
<td>6</td>
<td>1168</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>GenomeInfoDb</td>
<td>18</td>
<td>19</td>
<td>29</td>
<td>6</td>
</tr>
<tr>
<td>GenomicRanges</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GenomicScores</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRanges</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S4Vectors</td>
<td>57</td>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>utils</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## FROM GenomicScores/NAMESPACE
## importFrom(Biostrings, DNA_BASES)
## importMethodsFrom(Biostrings, match)

expfun <- getNamespaceExports("Biostrings")
nonstd <- expfun[grep("^\./", expfun)]
length(setdiff(expfun, nonstd))
[1] 240
Quantifying package dependency burden

gistURL <- "https://gist.github.com/rcastelo/7429d05178dddb57a38b42093c2ddfe2"
develtools::source_gist(gistURL)

g <- pkgDepGraph("GenomicScores", db)
gvtx <- graph::nodes(g)
gdep <- pkgDepGraph("Biostrings", db)
gdepvtx <- graph::nodes(gdep)
depov <- length(intersect(gvtx, gdepvtx)) / length(union(gvtx, gdepvtx))
depov
[1] 0.075
Quantifying package dependency burden

Using functionality from itdepends we can quantify the dependency burden as follows.

```r
gistURL <- "https://gist.github.com/rcastelo/7429d05178ddb57a38bd42093c2ddfe2"
devtools::source_gist(gistURL)

## build an acyclic digraph of package dependencies
pkgDepGraph()

## calculate package dependency metrics()
pkgDepMetrics()

## find what function calls in a package lead to a target dependency
funCalls2Dep()
```
Quantifying package dependency burden

```r
# Code

g <- pkgDepGraph("GenomicScores", db)
gvtx <- graph::nodes(g)
gdep <- pkgDepGraph("Biostrings", db)
gdepvtx <- graph::nodes(gdep)
depov <- length(intersect(gvtx, gdepvtx)) / length(union(gvtx, gdepvtx))
depov
[1] 0.075
```

The dependency overlap, calculated as the ratio of the intersection over the union of dependencies, tells us that the dependencies from Biostrings represent a mere 0.075 fraction of the dependencies of GenomicScores.
### Quantifying package dependency burden

```r
pkgDepMetrics("GenomicScores", db)
```

<table>
<thead>
<tr>
<th>ImportedBy</th>
<th>Exported</th>
<th>Usage</th>
<th>DepOverlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biobase</td>
<td>1</td>
<td>128</td>
<td>0.781250</td>
</tr>
<tr>
<td>BSgenome</td>
<td>1</td>
<td>93</td>
<td>1.075269</td>
</tr>
<tr>
<td>XML</td>
<td>2</td>
<td>176</td>
<td>1.136364</td>
</tr>
<tr>
<td>IRanges</td>
<td>4</td>
<td>254</td>
<td>1.574803</td>
</tr>
<tr>
<td>BiocGenerics</td>
<td>5</td>
<td>139</td>
<td>3.597122</td>
</tr>
<tr>
<td>GenomicRanges</td>
<td>4</td>
<td>104</td>
<td>3.846154</td>
</tr>
<tr>
<td>S4Vectors</td>
<td>11</td>
<td>262</td>
<td>4.198473</td>
</tr>
<tr>
<td>GenomeInfoDb</td>
<td>5</td>
<td>53</td>
<td>9.433962</td>
</tr>
<tr>
<td>AnnotationHub</td>
<td>4</td>
<td>33</td>
<td>12.121212</td>
</tr>
<tr>
<td>Biostrings</td>
<td>NA</td>
<td>240</td>
<td>NA</td>
</tr>
</tbody>
</table>

GenomicScores uses about 1% of the functionality exposed by BSgenome, while BSgenome overlaps a fraction of 0.36 of the dependencies of GenomicScores. Therefore, we would say that BSgenome is a candidate package to lighten the dependency burden of GenomicScores.
What are the function calls in GenomicScores to BSgenome?

```r
funCalls2Dep("GenomicScores", "BSgenome", db)
# A tibble: 1 x 3
# Groups:   pkg [1]
 pkg      fun                 n
<chr>    <chr>           <int>
 BSgenome referenceGenome     4
```

Further arguments to funCalls2Dep() allow to see, using the package itdepends, what are the actual lines of code doing those function calls.